An Application Level Video Gateway

Elan Amir
University of California, Berkeley
elan@cs.berkeley.edu

Steven McCanne
University of California, Berkeley
mccanne@cs.berkeley.edu

Hui Zhang
Carnegie Mellon University
hzhang@cs.cmu.edu

ABSTRACT

The current model for multicast transmission of video over the Internet assumes that a fixed average bandwidth is uniformly present throughout the network. Consequently, sources limit their transmission rates to accommodate the lowest bandwidth links, even though high-bandwidth connectivity might be available to many of the participants. We propose an architecture where a video transmission can be decomposed into multiple sessions with different bandwidth requirements using an application-level gateway. Our *video gateway* transparently connects pairs of sessions into a single logical conference by manipulating the data and control information of the video streams. In particular, the gateway performs bandwidth adaptation through transcoding and rate-control. We describe an efficient algorithm for transcoding Motion-JPEG to H.261 that runs in real-time on standard workstations. By making the Real-time Transport Protocol (RTP) an integral component of our architecture, the video gateway interoperates with the current Internet video tools in a transparent fashion. We have built a prototype of the video gateway and used it to redistribute multi-megabit JPEG video seminars from the Bay Area Gigabit Network as 128 kb/s H.261 MBone sessions.

KEYWORDS

Conferencing protocols; digital video; image and video compression and processing; multicasting; networking and communication; efficient transcoding.

1 INTRODUCTION

The initial deployment of a number of multimedia applications such as nevot [13], vat [7], ivs [16], nv [5], and vic [9] over the Internet and the overlayed multicast backbone or MBone [4] has proven successful. However, a number of issues need to be resolved before the full potential of these new applications can be realized. One such issue is how to deal with heterogeneity in a conference with a large number of receivers.

Heterogeneity is intrinsic to the Internet technology. It exists both inside the network and at the end systems. Within the network, due to the mismatch between link speeds and unbalanced load distribution inside the network, the bandwidth available between different sender-receiver pairs is different. While one portion of the network may be a high speed ATM network, other portions may be ethernet, and the backbone may be composed of T1 and T3 links. In addition, different hosts have different processing power and video/audio hardware configurations. One host may be a 100 MIPS workstation without special video hardware, another host may be a PC with a JPEG board. As the Internet evolves to higher speed and larger size, the problems caused by heterogeneity will only get worse.

In existing applications, each sender transmits video at the same rate to all receivers. For a heterogeneous set of receivers, the source would have to run at a rate that matches the most constrained receiver. This is not satisfactory. Instead, receivers along high bandwidth paths should receive correspondingly higher quality.

One approach to this problem is the use of layered coding algorithms [10, 15, 17] where hosts along lower bandwidth paths receive fewer layers. While this approach is elegant and efficient, it is not yet deployed nor compatible with the installed base of JPEG hardware and existing video applications. Our goal is to create an architecture that is compatible with prevalent hardware and networking technology.

In this paper, we present the design and implementation of a video gateway that addresses the problem of heterogeneity. Our approach provides a mechanism for matching the transmission quality to the heterogeneous bandwidth constraints of distinct regions of a single logical multicast session. The gateway does so by intelligently managing incoming and outgoing video streams using transcoding and rate-control. By making the Real-time Transport Protocol (RTP) [14] an integral component of our architecture, the video gateway interoperates seamlessly with the current Internet video tools. We have demonstrated our implementation in the Bay Area Gigabit Network (BAGNet) and successfully used it to transcode high-rate JPEG [6] seminar video into

128 kb/s H.261 [18] streams for the MBone.

2 BACKGROUND AND MOTIVATION

In this section, we motivate the need for our video gateway architecture by giving several example scenarios.

2.1 BAGNet and MBone

One of our goals is to integrate the MBone applications into the BAGNet environment. We briefly describe BAGNet and MBone environments, then motivate the need for the video gateway by discussing issues of running existing MBone applications across both BAGNet and the general Internet.

BAGNet is an OC-3C (155 Mb/s) ATM network that connects 15 San Francisco Bay Area sites including universities, government and industrial research labs. One goal in the BAGNet project is to develop and deploy teleseminar applications with high quality video and audio. Currently, each site has only several workstations connected directly to the ATM network, and one of these machines is used as a gateway to connect BAGNet with other machines in the organization. Typically, ethernets are used within each organization. In a teleseminar scenario, one of the machines directly attached to BAGNet will multicast high quality thus high rate video across BAGNet. A stream of full motion JPEG compressed NTSC quality video will consume about 6 Mb/s bandwidth. While multiple 6 Mb/s video streams can be easily supported within BAGNet, transmitting even one of these streams onto an ethernet is not practical as the ethernets are shared. In addition, it may be desirable to transmit the video across the entire MBone where the aggregate bandwidth is only about 500 kb/s for all sessions. Therefore, we would like to accommodate three types of receivers in such an environment: hosts connecting directly to BAGNet, hosts connecting to BAGNet via ethernets and routers, and hosts on the global MBone.

Figure 1(a) shows such a scenario. Assume that host H0 is transmitting high quality JPEG video at 6 Mb/s. The ATM switch S forwards the transmission to BAGNet and to a router R0 connected to an ethernet, which contains among its hosts an MBone router R1. In order to not flood the ethernet, we run a video gateway on R0 that transcodes the 6 Mb/s JPEG down to 1 Mb/s. Thus, ethernet hosts such as H1 still receive the video transmission at a reasonably high quality. We place an additional video gateway between R1 and the MBone. This gateway further reduces the bandwidth requirements by transcoding the JPEG stream to H.261 and limits its output rate to 128 kb/s.

2.2 Linking Remote MBone Sites

Another application of our video gateway architecture is to link several remotely located MBone sites together. Consider the following example: a seminar at Berkeley is multicast locally to the campus, and a group of researchers at Carnegie Mellon University wants to tune in and participate. Unfortunately, a scenario like this — where small groups communicate via long haul networks — is not effectively supported by the current IP Multicast infrastructure.

The major mechanism of limiting the scope of multicasting traffic over today's MBone is use of TTL. The mechanism works as follows. Each multicast packet is assigned a Time-To-Live number in the IP header and each link is assigned a static threshold value. If the packet's time-to-live value is less than the threshold, the packet is not forwarded. By placing larger thresholds on the links that connect remote sites and using smaller TTL numbers for each multicast packet, the multicast traffic can be contained in a local region.

In the example mentioned above, in order for researchers at CMU to receive the multicast video, the video has to be sent out from Berkeley using a TTL value that is higher than the threshold value of cross-country links, which means that the video will be distributed across the entire country. This is not satisfactory.

With our video gateway, this problem can be easily solved by placing one video gateway at each site, and linking the two video gateways by unicast connections. Figure 1(b) illustrates this scenario. Multicast video from CMU is forwarded by the local video gateway via unicast connection to the remote video gateway, which in turn multicasts the video to the remote site at UCB.

2.3 Multicast Video across ISDN Links

Most MBone transmissions use a target rate of 128 kb/s for video and 64 kb/s for audio, precluding simple bridging of MBone sessions across a 128 kb/s ISDN line. By transcoding a 128 kb/s H.261 coded video stream into a 64 kb/s H.261 stream (and by similarly adapting the audio bit rate), an MBone session can be bridged over an ISDN link, as depicted in Figure 1(c).

2.4 Multicast Video over Wireless Links

Wireless links are characterized by relatively low bandwidth and high transmission error rates. Figure 1(d) illustrates the role of the video gateway in a topology containing mobile wireless hosts. By placing a video gateway at the basestation router (BS), we can transcode the incoming video stream to a lower bandwidth stream and control the rate of output transmission over the wireless link.

3 THE VIDEO GATEWAY

We now address the design of a video gateway architecture that will flexibly support the configurations discussed in the previous section. In order to interoperate with the existing MBone video tools, the video gateway must be RTP compatible. In this section, we give a brief overview of RTP, discuss the ramifications of transcoding an RTP data stream, and propose an architecture that can perform this transcoding in a protocol-consistent fashion.

3.1 Real-time Transport Protocol

The Real-time Transport Protocol [14] is an application-level protocol that is designed to satisfy the needs of multi-party multimedia applications. In the IP protocol stack, RTP lies

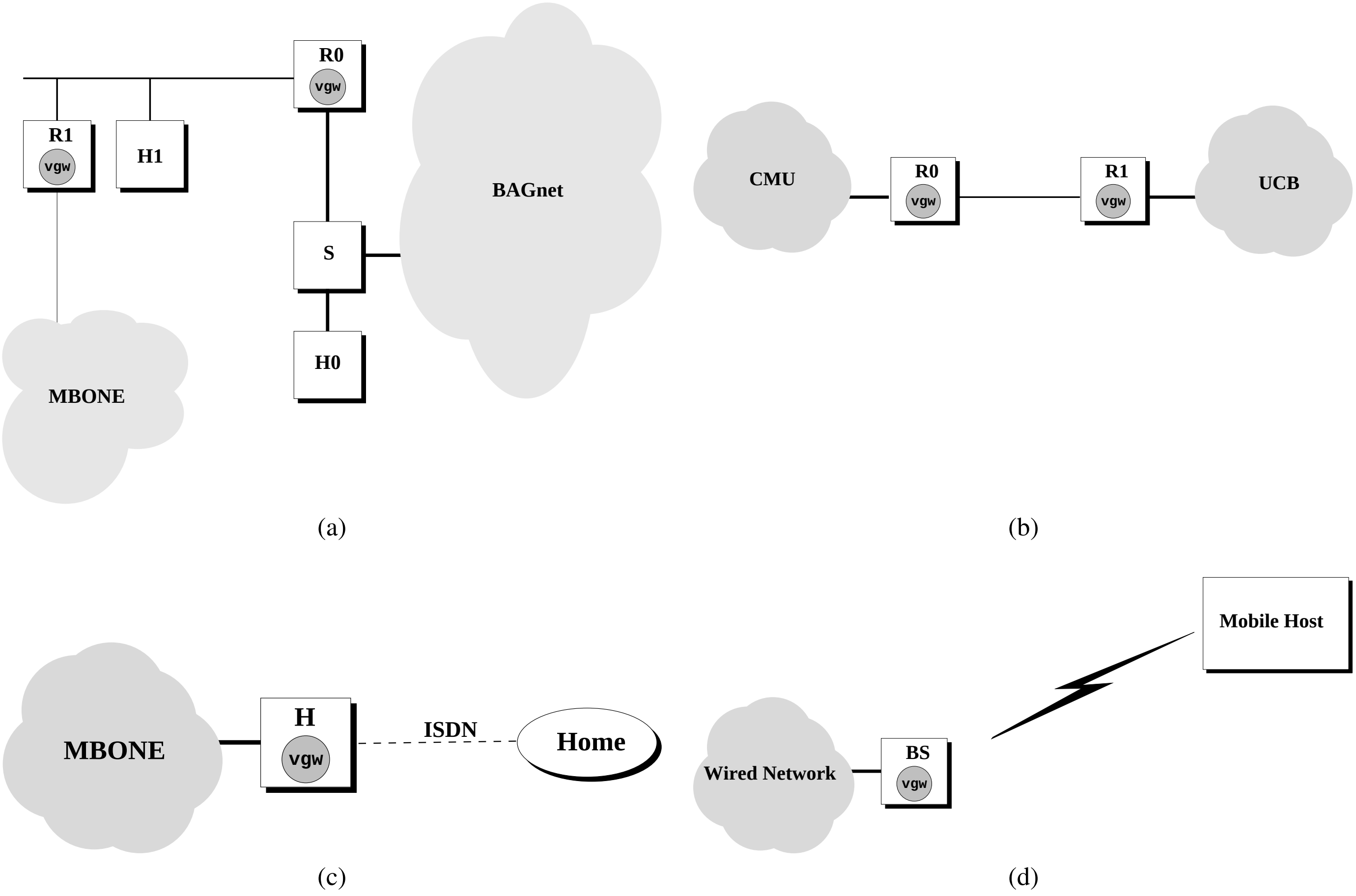

Figure 1: Four example scenarios for a video gateway.

just above UDP. The protocol consists of two parts: the data transfer protocol RTP and the control protocol RTCP.

Each RTP data packet consists of an RTP packet header and the RTP payload. The packet header includes a sequence number, a media-specific timestamp, and a synchronization source (SSRC) identifier. The SSRC provides a mechanism for identifying media sources in a fashion independent of the underlying transport or network layers. End-hosts allocate their SSRC randomly and because the SSRC's must be globally unique within an RTP session, a collision detection algorithm is employed to avoid conflicts. All packets from an SSRC form part of the same timing and sequence number space. Receivers group packets by SSRC identifiers for playback.

The RTP control protocol (RTCP) provides mechanisms for data distribution monitoring, cross-media synchronization, and sender identification. This control information is disseminated by periodically transmitting control packets to all participants in the session, using the same distribution mechanism as for data packets. The transmission interval is randomized and adjusted according to the session size to maintain the RTCP bandwidth below some configurable limit.

Distribution Monitoring. The primary function of RTCP is to provide feedback to the session on the quality of the data distribution. This information is critical in diagnosing failures and monitoring performance, and can be used by applications to dynamically adapt to network congestion [3]. Monitoring statistics are disseminated from active sources via RTCP sender reports (SR) and from receivers back to the entire session via receiver reports (RR). The SR statistics include, among other things, the sender's cumulative packet count and sender's cumulative byte count. Each receiver generates a separate reception report for each active source. The RR statistics include a cumulative count of lost packets, a short-term loss indicator, an estimate of the jitter in data packets, and timestamps for round-trip time estimation.

Synchronization. Since media streams are distributed on distinct RTP sessions (with distinct SSRCs), the protocol provides a mechanism for synchronizing media across sessions, which relies on canonical-name (CNAME) identifiers and SRs. Each SR contains that source's CNAME and the correspondence between local real-time and the media-specific time units. By matching sender reports across different media according to their CNAME, a receiver can align the time offsets of different media streams to reconstruct the original synchronization.

Sender Identification. Each session participant identifies itself by binding text descriptions to their SSRC using "source description" (SDES) messages.

In addition to the base protocol semantics, the RTP specification defines two types of "intermediate systems": *mix-*

ers and *translators*. Mixers concentrate multiple incoming streams into a single outgoing stream. In order to combine streams, a mixer might inject buffering delay to counteract network jitter, and thus must resynchronize the streams. The mixer therefore has its own SSRC identifier and appears explicitly in the session. Translators, on the other hand, do not resynchronize the data stream and do not necessarily appear in the session.

3.2 The Gateway Architecture

In theory, the basic gateway architecture is straightforward: we merely establish an application-level path that joins two separate RTP sessions and properly transforms the RTP data and control streams. RTP data packets are especially easy to handle. We can simply forward a data stream from one session to the other, either unmodified or transcoded to adapt the transmission bandwidth. The bandwidth can be further regulated by applying a rate-limiter to the output. Since RTP data streams are self-describing and "stateless", there is no special processing or external signaling to be carried out in order to instantiate the transformed stream. By directly bridging the two sessions, we implicitly merge the SSRC identifier space and the collision resolution algorithm penetrates the gateway.

However, a problem surfaces when we consider RTCP packets. While SDES messages can flow through the gateway unmodified, synchronization information and SR/RR distribution statistics are problematic. When a gateway transcodes a stream, the packet counts, byte counts, and timing information will all be affected. Hence, RTCP packets cannot be simply forwarded across the gateway — they must be modified in a self-consistent fashion.

When we first began the design of our video gateway, we encountered an inadequate treatment of this problem in the protocol definition. Revision 6 of the RTP draft specification accounted only for mixers that resynchronized streams and translators that did not modify the data packets. Because we wanted our architecture to admit data transcoding, we were required to use a mixer design. However, because a mixer becomes the new point of synchronization, a video gateway based on this model could not preserve the semantics of the RTP cross-media synchronization information. That is, the newly generated video stream could no longer be synchronized with an audio stream generated at the original source. One immediate solution to this problem is to implement a combined audio/video gateway that preserves synchronization across the system, but this complicates the design and introduces unnecessary resynchronization delay.

As a result of our interactions with the RTP designers, new definitions of translators and mixers appeared in revision 7 of the protocol draft. The modified specification relaxes the constraint on translators, leaving more freedom to the application-gateway designer.

The revised RTP specification allows us to build a transcoding gateway that:

- preserves cross-media synchronization information without the need to re-time the media;
- offers minimal delay, since data packets can often be forwarded as soon as they arrive; and,
- operates in a session-transparent fashion.

We achieve these goals with an architecture that effectively passes all RTCP source identification information directly though the gateway, but decouples the flow of sender and receiver reports. Instead, sender and receiver reports are modified to reflect the distribution quality as seen by the gateway.

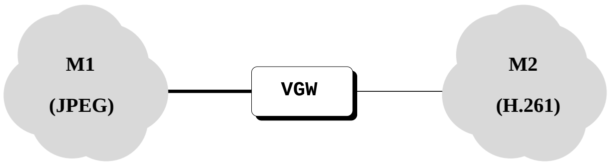

Figure 2: A video gateway between two RTP multicast groups. Group M_1 is transmitting JPEG video whereas group M_2 has capability to only receive or transmit H.261.

More specifically, consider the scenario in Figure 2, in which two multicast groups M_1 and M_2 are joined by the video gateway, VGW. Assume that the M_1 session contains high-rate JPEG sources as well as low-rate H.261 sources, while the M_2 session contains only H.261 sources. Further assume that the gateway is configured to transcode JPEG to H.261 in the M_1/M_2 direction. In our architecture, the gateway simply forwards H.261 packets to the alternate session (in either direction), but transcodes JPEG packets to H.261. While data packets are transcoded in a straightforward manner, RTCP sender and receiver reports must be handled as follows:

- On receiving an SR from a JPEG source in M_1, VGW modifies the sender statistics to reflect the H.261 statistics of the transcoded stream originating from VGW.
- On receiving an SR from an H.261 source in either M_1 or M_2, VGW forwards the packet unmodified.
- On receiving an RR from a receiver in M_2, which is reporting on a transcoded source from M_1, VGW modifies the reception statistics to reflect the JPEG reception statistics seen by VGW.
- On receiving an RR from a receiver in M_1 or M_2, which is reporting on a non-transcoded source, VGW forwards the packet unmodified.

In all cases, the modified (or unmodified) RTCP packet is immediately forwarded to the other session.

By decoupling the flow of distribution reports across the gateway, we prevent a source in one half of the session from learning about distribution problems in the other half of the session. This might raise concerns over the impact on congestion avoidance algorithms where a source relies on such information to adjust its transmission rate. However, our explicit decoupling is in fact a viable configuration even in the presence of adaptive rate control. Recall that the original motivation of the transcoding operation is to provide bandwidth adaptation. Rather than adjust the rate at the source, it would be better to adjust the rate at a point closer to the congested receiver, thereby preventing unnecessary rate backoff for well-connected receivers. Thus, it makes more sense for the gateway source to run its own congestion control loop.

3.3 Gateway Control

The gateway architecture described in the previous section provides a general mechanism for bandwidth adaptation and session aggregation. This mechanism must somehow be controlled and configured. Hence, we export an interface that allows remote agents to dynamically configure the gateway. The control interface allows agents to:

- establish rate controls for each stream as well as the global session,
- define rules for selecting output formats from input formats,
- define compression parameters for each transcoded stream (e.g., quantizers), and
- install filters (e.g., to forward only the "lecturer's video").

This decomposition separates mechanism from policy. By defining a control interface in this fashion, we displace the policy level functionality associated with conference control to external agents. The design of such external control agents are beyond the scope of this paper.

4 TRANSCODING

We now discuss the transcoding process by which the video gateway maps a compressed video bit stream into a different (lower-rate) bit stream either by employing the same compression format with alternate parameters or by employing another format altogether. We call the agent that performs this transcoding algorithm a *transcoder*.

4.1 Design

A high-level depiction of our transcoder design is given in Figure 3. The basic model involves transcoding from some set of supported input formats to a (possibly different) set of output formats. Each input format is handled by a module that decodes the incoming bit stream into an intermediate representation. This representation is transformed and delivered to an encoder, which produces a new bit stream in a new (or possibly same) format.

If we restrict our design to rely on a single format for the intermediate representation, we would be forced to adopt a pixel representation since this is the least common denominator for all compression schemes. But requiring every transcoder configuration to decompress all the way to the pixel domain, and then re-encode the stream from scratch, would impose a substantial performance penalty. Instead, we allow for multiple intermediate formats. In this way, encoder/decoder pairs can optimize their interaction by choosing an appropriate intermediate format. For example, DCT-based coding schemes like JPEG and H.261 could be more efficiently transcoded using DCT coefficients, bypassing the forward and reverse transform of each block.

Because we have decomposed the transcoder into separate encoder and decoder stages, the system is easily extensible. To support a new input format, for example, only a single encoder module needs to be implemented. Additionally, to support multiple intermediate formats in a flexible way, we define format translators that map between intermediate formats. For example, we might have a JPEG decoder that produced only a DCT output representation, which needed to be encoded by a wavelet-based coding scheme that accepted only pixel input. Rather than implement a separate JPEG decoder that produces pixels directly, we implement a generic DCT to pixel converter. If the performance of this generic approach is unsatisfactory, it can be optimized by implementing matched encoder/decoder pairs.

By decoupling the encoder and decoder through an intermediate format, we can additionally perform generic transformations on the canonical image data. Such transformations are vital to the video gateway because large reductions in bandwidth between the input format and output format cannot be attained exclusively with encoder parameter and format choices. For example, bit rates below 64 kb/s are more or less infeasible for full-sized NTSC streams at 30 f/s. Instead, using our transformation model, we can apply aggressive temporal and spatial decimation on top of a format conversion. We also might need to perform a frame geometry conversion since not all compression schemes support the same resolutions (for example, H.261 supports only the "Common Interchange Format"), or color decimation conversion since different formats downsample the chrominance planes differently (e.g., 4:1:1 versus 4:2:2 YUV).

4.2 A JPEG/H.261 Transcoder

Two principal factors determined the focus of our initial transcoder design: (1) the widespread deployment of Motion-JPEG hardware and the decision to utilize JPEG as the primary compression scheme for BAGNet, and (2) the large installed base of MBone applications that can decode H.261 (vic and ivs) as well as the low bit-rate requirements of H.261. In the rest of this section, we describe the design of an efficient JPEG to H.261 transcoder.

The H.261 specification defines two types of video blocks: *intra*-mode blocks which are independent of past blocks, and *inter*-mode blocks which are predicted from previous blocks. Because inter-mode blocks are coded as differences, a lost update will cause a reconstruction error that persists until an intra-mode block refresh. At low bit rates, this interval can be substantial.

Intra-H.261. A solution to this problem is to avoid inter-mode blocks altogether and instead code every block in intra-mode. This approach, called Intra-H.261 [9], relies on "conditional replenishment" instead of motion compensation to reduce the bit rate. In conditional replenishment, the video image is partitioned into small blocks and only the blocks

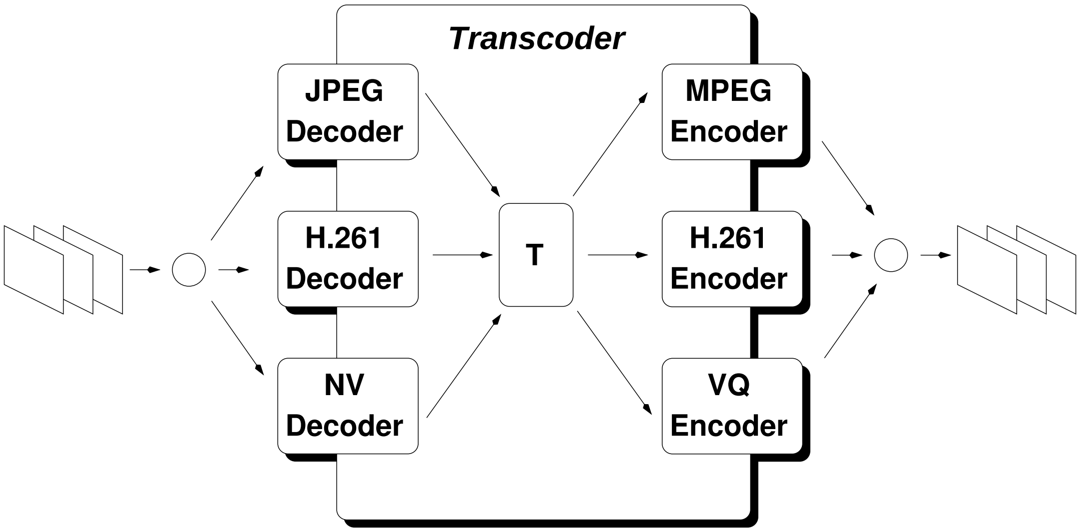

Figure 3: The basic design of the transcoder. Any of several supported input formats can be converted into any supported output format. The intermediate stage denoted by **T** performs a transformation on the internal representation, when necessary, to match the conventions of the decoder and encoder. In addition to any bandwidth reduction inherent in format conversion, the output can be rate-controlled by decoupling the generation of output frames from the arrival of input frames.

that change are transmitted. Intra-H.261 carries out conditional replenishment by using H.261 macroblock addressing to skip over unreplenished blocks.

Since an Intra-H.261 coder never utilizes inter-mode blocks, it need not perform motion compensation, which substantially reduces the run-time complexity. Moreover, we can easily build a coder that takes DCTs instead of pixels as input. In this way, the encoder complexity is further reduced because it need not compute DCTs. This property is exploited in the data-flow path optimization which is detailed in the next section.

4.2.1 Data Flow Optimizations

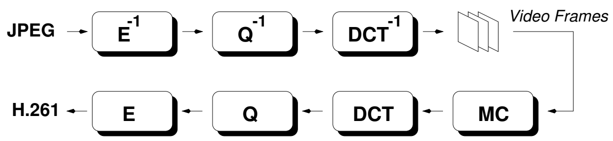

Figure 4: A generic JPEG/H.261 transcoder.

Figure 4 shows the basic data flow of a generic implementation of a JPEG/H.261 transcoder. The input to the transcoder is a stream of JPEG-encoded frames. Each frame is decoded into pixels, which involves entropy decoding, reverse quantization, and reverse DCT. Once in the pixel domain, the frames are input to an H.261 encoder, which performs motion compensation on each block and codes the motion residual using an approach similar to JPEG.

This decomposition is straightforward to implement since it can be built simply by interfacing existing coders. However, the resulting performance, for a software implementation, is unsatisfactory. We experimented with exactly this configuration using the fairly well-tuned JPEG decoder and (Intra-)H.261 encoder from vic. Even on a fast workstation (SGI Indy, 133MHz MIPS R4600SC), the naive transcoder configuration operated only in the 6 to 13 f/s range, well below real-time performance.

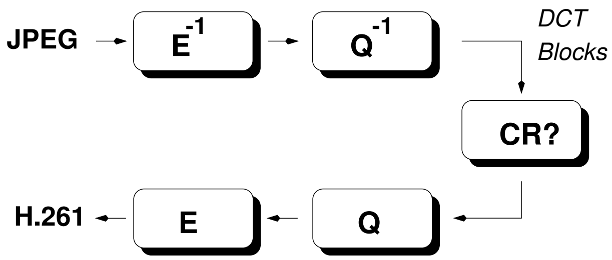

Figure 5: An optimized JPEG/H.261 transcoder.

The two primary bottlenecks in the transcoder are the DCT computations and the memory traffic that results from manipulating uncompressed images. Reducing or eliminating these bottlenecks would dramatically improve the transcoding performance. By carrying out conditional replenishment in the DCT domain and by utilizing an Intra-H.261 that takes DCTs as input, we can do just this. The modified data flow path is shown in Figure 5. Here the output of the JPEG inverse quantizer is redirected to the input of the H.261 quantizer thereby bypassing the two transform computations. Furthermore, since conditional replenishment is carried out in the DCT domain, we can prune much of the memory traffic by deciding early that a block is to be skipped.

There are two complications with our modified data path: resolution conversion and chrominance subsampling.

Resolution Conversion. JPEG is typically transmitted in either PAL or NTSC format, while H.261 is CIF (or 1/4-CIF) resolution. Instead of employing an expensive scale operation, we simply downsample the input (if necessary) to approximately match CIF or 1/4-CIF resolution. Then, for NTSC, the frame is embedded in a CIF geometry using a gray border, and for PAL, a few pixels at the vertical edges of the frame are cropped away.

Chrominance Subsampling. Because most hardware-based JPEG codecs support only 4:2:2 YUV decimated video while H.261 requires 4:1:1 decimated video, we must subsample the JPEG chrominance planes vertically by two. This is trivial in the pixel domain: simply drop every other pixel (after optionally low-pass filtering to avoid aliasing). A brute force approach is to compute the reverse 8x8 2D DCT of two vertically adjacent blocks, downsample the resulting 16x8 block to an 8x8 block, and forward transform this 8x8 block. However, this defeats our fast-path optimization, which relies on avoiding DCT computations. It turns out that pixel subsampling can be carried out by operating on the DCT coefficients directly.

Since the 2D DCT is a separable transform, we need only consider the one dimensional downsampling problem; that is, we can operate on each column of the 2D DCT independently. The appendix contains a derivation of an (approximate) algorithm for computing the of downsampling a 16-point signal using operations performed directly on the two 8-point DCT coefficients of the two halves of the 16-point signal. The output is an 8-point DCT of the subsampled signal. By running this algorithm over each of the columns of two vertically adjacent 8x8 DCTs, we effectively subsample the underlying signal. Though our algorithm is only approximate, it is fast and produces good qualitative results.

4.3 Addressing Some Weaknesses

Two commonly cited weaknesses of transcoding systems are increased end-to-end delay and an incompatibility with encryption. Increased delay is incurred, for example, when encoding JPEG to MPEG (with B frames) because the encoder must collect several frames into a "group of pictures" before generating new output. However, for the compression formats commonly used in the Internet, this is not a problem. The packet formats of the schemes based on conditional replenishment are intentionally designed so that each packet may be processed independently of the rest. Hence, the theoretical transcoding delay is zero (modulo processing and store and forward delays) because we can transcode each packet as soon as it arrives. In practice, our JPEG/H.261 transcoder waits until an entire JPEG frame arrives. Since traffic is usually smoothed at the source, we incur one frame interval of delay. On the other hand, if the JPEG packets arrive in order, they could be decoded incrementally and output packets generated "on the fly".

The other complaint against transcoding systems is their incompatibility with encryption. This argument is based on the view that the transcoding agent would be integrated into the network architecture and that privacy would be compromised by entrusting the network with the encryption key. However, in our model, the transcoder is an application level agent that is deployed by the user, within that user's administrative domain. Thus, the user will be able to configure the transcoder with the session key(s) in a secure fashion.

5 IMPLEMENTATION

We have implemented our gateway and transcoder architectures in a prototype application called *vgw*. We leveraged off the flexible code base in *vic*, by incorporating vic's H.261 encoder, JPEG decoder, and networking implementation into vgw. While our eventual goal is to support several combinations of efficient transcoding operations, the current implementation supports only the JPEG/H.261 transcoding model described in the previous section. Streams that are not JPEG (or not intended to be transcoded) are simply forwarded across the gateway.

JPEG/H.261 Transcoder. By modifying vic's H.261 encoder and JPEG decoder to support a DCT based call interface, we were able to easily implement the transcoder architecture described in the previous section. The JPEG decoder performs table-driven Huffman decoding of the DCT transform coefficients in parallel with conditional replenishment. We optimize the case of an unreplenished block by Huffman decoding the first six DCT coefficients and comparing them to what has already been transmitted, as described in [9]. If they are "similar enough", we skip the current block by parsing the rest of the coefficients in the block without any additional processing. Hence, in the case where there is little motion in the scene, the system runs at the Huffman decoding speed (which just uses table lookups).

A single frame buffer of DCT coefficients is maintained by the decoder, which is used as a reference for the conditional replenishment decision. As the decoder encounters replenishable blocks, it replaces the corresponding block in the reference frame and sets a flag in bit vector to indicate that the block has changed. After a whole frame is decoded, the bit vector and DCT frame buffer are passed to the H.261 encoder. The encoder then codes each H.261 macroblock that contains an altered block, using an index table to relate H.261 macroblock and "Group of Block" addresses to NTSC block numbers. The encoder skips over unreplenished blocks with macroblock addressing. To guarantee that blocks are eventually replenished if lost (or if a receiver joins an in-progress transmission), an update process forces the transmission of a few extra background blocks for each frame.

In addition to performing conditional replenishment, the H.261 encoder requantizes the coefficients using scalar quantization. Because the full dynamic range of the DCT coefficients is not achievable for small quantizers, we dynamically adjust the quantizer of each macroblock to guarantee that the largest DCT coefficient in the block is representable.

Finally, before encoding a chrominance plane, it must be subsampled as discussed earlier. This subsampling is carried out using the code in the appendix, which transforms two blocks from the DCT frame buffer into a single DCT stored in a scratch buffer. This temporary result can then be encoded as usual.

Rate-controller. A fundamental function of the gateway is to regulate the output bandwidth used by the transcoded packet streams. Since our codec implementations use fixed quantizers, we cannot rely on adaptive quantization to fine tune the output bit rate. Instead, we generate variable frame rate output, dropping frames as necessary from the input stream to meet a given rate constraint. Most compression schemes are stateful, which means that we must continually decode the input stream into an intermediary frame buffer, and then re-encode that buffer at the lower rate. On the other hand, with a stateless format like Motion-JPEG, we can decode only the frames we forward and throw away all others at the input, thereby saving computation time.

For example, let's suppose that we want to scale an H.261 stream to a lower rate (i.e., transcode H.261 to H.261). We can do this by employing both a more aggressive quantizer (to trade off quality for bit rate) and output frame rate control (to fine tune the resulting rate). Since H.261 is block oriented, we can combine several input frames into a single output frame by encoding all the blocks that have changed since the previous output frame. Thus, the output frame rate is made independent of the input frame rate and along with the traffic shaping mechanism described below, we achieve the required bandwidth limit.

We implement the variable frame rate scheme by scheduling frame times. When a frame from a given source is transmitted, the next frame time is chosen far enough into the future so that the resulting bit rate matches the rate allocated to that source. However, the scheduled output frame time will not exactly match the arrival of a new input frame. But if we include the lag time between the scheduled time and actual time in our computed bit rate, then the resulting output rate will exactly match the target rate. Additionally, the rate controller performs traffic smoothing by pacing out all the packets of a single frame over the entire frame time.

Because the gateway supports an arbitrary number of active video sources, we must extend our rate control algorithm to multiple sources. We do so in the most straightforward fashion possible — a separate instance of the rate controller is run for each active source such the sum of the individual rates is limited by the configured session bandwidth. The fraction of bandwidth allocated to each source is dynamically specified via an external conference coordination protocol. If no allocations are explicitly assigned, the bandwidth is divided equally among all active sources.

6 PERFORMANCE

We measured the performance of *vgw* on an SGI Indy (133MHz MIPS R4600SC) running IRIX 5.3. The JPEG quantization tables were generated using the Independent JPEG Group's formula with an input value of 30, and the H.261 quantizer was set to 10. For a low-motion scene, the JPEG/Intra-H.261 transcoder uses about 60% of the CPU to process 1/4-NTSC at 30 f/s. For a very high-motion scene, the performance drops to 15 f/s.

We also compared the relative performance of the optimized and brute force data paths described in Section 4.2. The optimizations account for about a factor of three improvement in run-time performance in both the low and high motion scenes:

motion	*slow-path*	*fast-path*
low	12.9 f/s	33.3 f/s
high	6.9 f/s	21.4 f/s

7 FUTURE WORK

Currently, the video gateway supports only JPEG to H.261 transcoding. We intend to implement additional formats including a vector quantization scheme developed for the InfoPad project at U.C. Berkeley [1].

For non-transcoded streams, the rate-controller adapts only the packet rate, not the frame rate. This leads to degraded quality because packets are dropped randomly across the bit stream. A more effective model is to adjust the bandwidth by controlling the frame rate. In other words, placing a rate-limiter on the output requires transcoding all of the incoming streams. Thus, we need to develop support for transcoding between identical formats. In particular, *vgw* will be much more useful if we supported the H.261 and nv compression algorithms, which are in widespread use in the Internet.

The external control interface to the video gateway has not yet been implemented. We plan to realize it with the "Conference Bus" abstraction described in [9]. The Conference Bus provides a high-level coordination protocol for real-time multimedia applications. Simple control scripts can be easily prototyped using a Tcl [11] interface to the Conference Bus. In this way, users can easily configure the gateway for unforeseen applications. For example, a script could configure and dynamically control the gateway to switch among conference participants.

8 SUMMARY

We have presented an application level gateway for adapting packet video transmissions to heterogeneous environments. By incorporating an RTP compatible framework, our video gateway immediately becomes an integral component of the existing application and networking infrastructure of the Internet. We have introduced a flexible architecture for transcoding multiple video formats. In particular, we have developed and implemented an efficient algorithm for real-time transcoding of Motion-JPEG to Intra-H.261. Finally, we have successfully deployed the system in a production environment in which high-bandwidth seminars from BAGNet were bridged to the MBone.

9 ACKNOWLEDGMENTS

We thank Rahul Garg, Deana Goldsmith, and Hoofar Razavi, who provided helpful comments on drafts of this paper. Our architecture derives from the design framework in the LBL Mbone conferencing tools, developed with Van Jacobson. Van provided constructive input on the design of our gateway architecture. Finally, we thank Stephen Casner, Henning Schulzrinne, Ron Frederick, and Van Jacobson for the email exchange in November 1994 that influenced our design decisions and clarified the RTP specification.

APPENDIX

This appendix describes an efficient algorithm for (approximately) subsampling a signal by operating directly on the DCT coefficients of the transformed signal. We use a matrix decomposition as in [8]. Instead of judicially factoring the matrices to reduce the number of multiply operations, we use approximate multipliers in order to reduce them to fixed point shifts and adds.

Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be two 8-point signals (represented as column vectors) and $\mathbf{X}_1$, $\mathbf{X}_2$ be their DCTs. Let $\mathbf{x}$ be the signal that results from downsampling the signal $(\mathbf{x}_1\mathbf{x}_2)^T$. Our problem then is to compute $\mathbf{X}$, the DCT of $\mathbf{x}$. We will express this computation in matrix form.

Let D_N be the matrix operator for an N-point DCT, and let G be a 16×8 downsampling matrix,

$$G_{ij} = \begin{cases} 1 & j = 2i \\ 0 & \text{o.w.} \end{cases}$$

We can then express the pixel downsampling operation as an operation on DCT coefficients:

$$\mathbf{X} = D_8 G \begin{bmatrix} D_8^{-1} & \mathbf{0} \\ \mathbf{0} & D_8^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{bmatrix}$$

To avoid aliasing, we must low-pass filter $\mathbf{x}_1$ and $\mathbf{x}_2$. Let $h(n)$ be our anti-aliasing filter, and $\mathbf{H}$ be its column vector DCT. We take $\mathbf{H} = (1,1,1,1,0,0,0,0)^T$ for a very simple low pass filter. Then, by the DCT convolution property [12], we can multiply $\mathbf{X}_k$ and $\mathbf{H}$ to have the effect of (approximately) convolving $h(n)$ and $\mathbf{x}_k$. We can write $\mathbf{H}$ as a matrix operator F:

$$F = \begin{bmatrix} I_4 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_4 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where I_4 is the 4×4 identity and $\mathbf{0}$ is a 4×4 matrix of zeros. Now, let

$$M = D_8 G \begin{bmatrix} D_8^{-1} & \mathbf{0} \\ \mathbf{0} & D_8^{-1} \end{bmatrix} F$$

Then,

$$\mathbf{X} = M \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{bmatrix}$$

This result shows that we can compute the DCT of the decimated signal as a linear combination of the DCTs of the input signals. The linear equations are determined by M, which is easily computed with the S [2] program shown in Figure 6. However, direct implementation of this algorithm on an 8x8 2D matrix would require 8^3 multiplies, which is higher in complexity than a brute force approach using forward and reverse DCTs. Fortunately, many of the values in M are zero and by using approximate arithmetic the computation runs very fast. On a RISC processor, each 8-pt signal can be loaded into registers, and each output point computed in a small number of shifts and adds. The code for this scheme, which produces good qualitative results, is given in Figure 7.

```
dctMatrix _ function(N)
{
        m _ matrix(0,N,N)
        v _ 0:(N-1)
        for (i in v) {
                k _ (2 * v + 1) * i
                m[i+1,] _ cos(k * pi / (2 * N))
        }
        m _ m * sqrt(2 / N)
        m[1,] _ m[1,] / sqrt(2)
        m
}
D8 _ dctMatrix(8)
D8i _ solve(D8)
L _ rbind(cbind(D8i, matrix(0,8,8)),
        cbind(matrix(0,8,8), D8i))
G _ matrix(0,8,16)
for (i in 1:8) G[i,2*i-1] _ 1
H _ matrix(0,16,16)
for (i in c(1:4,9:13)) H[i,i] _ 1
M _ D8 %*% G %*% L %*% H
```

Figure 6: An S program to compute M.

```
void
dct_decimate(const short* in0, const short* in1, short* o)
{
    for (int k = 0; k < 8; ++k) {
        int x00 = in0[0];
        int x01 = in0[1];
        int x02 = in0[2];
        int x03 = in0[3];
        int x10 = in1[0];
        int x11 = in1[1];
        int x12 = in1[2];
        int x13 = in1[3];
#define X_N 4
#define X_5(v)  ((v) << (X_N - 1))
#define X_25(v)  ((v) << (X_N - 2))
#define X_125(v)  ((v) << (X_N - 3))
#define X_0625(v)  ((v) << (X_N - 4))
#define X_375(v) (X_25(v) + X_125(v))
#define X_625(v) (X_5(v) + X_125(v))
#define X_75(v) (X_5(v) + X_25(v))
#define X_6875(v) (X_5(v) + X_125(v) + X_0625(v))
#define X_1875(v) (X_125(v) + X_0625(v))
#define X_NORM(v) ((v) >> X_N)

        o[0] = X_NORM(X_5(x00 + x10) + X_0625(x01 + x11) +
                      X_125(x03 + x13));
        o[1] = X_NORM(X_5(x00 - x10) + X_25(x01) +
                      X_0625(x03) + X_125(x11 + x12));
        o[2] = X_NORM(X_5(x01 - x11) + X_1875(x02 + x12));
        o[3] = X_NORM(X_1875(x10 - x00) + X_375(x01 + x02) +
                      X_5(x11) - X_125(x13));
        o[4] = X_NORM(X_5(x02 + x12) + X_25(x03 + x13));
        o[5] = X_NORM(X_125(x00 - x10) - X_1875(x01 + x11) +
                      X_25(x02) + X_5(x03 - x12));
        o[6] = X_NORM(X_625(x12 - x02) + X_375(x03 + x13));
        o[7] = X_NORM(X_125(x01 - x00 + x11 + x10 + x12) +
                      X_1875(x02) + X_25(x03) + X_5(x13));
        o += 8;
        in0 += 8;
        in1 += 8;
    }
}
```

Figure 7: A fast approximation function for subsampling in the DCT domain. Two 8x8 input DCTs are horizontally subsampled into a single 8x8 output DCT.

References

[1] BARRINGER, B., ET AL. InfoPad: A system design for portable multimedia access. In *Proceedings of Calgary Wireless '94 Conference* (Calgary, Canada, 1994).

[2] BECKER, R. A., CHAMBERS, J. M., AND WILKS, A. R. *The New S Language*. Wadsworth & Brooks, Pacific Grove, CA, 1988.

[3] BOLOT, J.-C., TURLETTI, T., AND WAKEMAN, I. Scalable feedback control for multicast video distribution in the Internet. In *Proceedings of SIGCOMM '94* (University College London, London, U.K., Sept. 1994), ACM.

[4] ERIKSSON, H. Mbone: The multicast backbone. *Communications of the ACM 37*, 8 (1994), 54–60.

[5] FREDERICK, R. Experiences with real-time software video compression. In *Proceedings of the Sixth International Workshop on Packet Video* (Portland, OR, Sept. 1994).

[6] ISO DIS 10918-1 Digital compression and coding of continuous-tone still images (JPEG). CCITT Recommendation T.81.

[7] JACOBSON, V., AND MCCANNE, S. *Visual Audio Tool*. Lawrence Berkeley Laboratory. Software on-line.[1].

[8] KOU, W., AND FJÄLLBRANT, T. A direct computation of DCT coefficients for a signal block taken from two adjacent blocks. *IEEE Transactions on Signal Processing* (July 1991).

[9] MCCANNE, S., AND JACOBSON, V. *vic:* a flexible framework for packet video. In *Proceedings of ACM Multimedia '95* (Nov. 1995), ACM.

[10] MCCANNE, S., AND VETTERLI, M. Joint source/channel coding for multicast packet video. *IEEE International Conference on Image Processing* (Oct. 1995).

[11] OUSTERHOUT, J. K. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[12] RAO, K. R., AND YIP, P. *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Academic Press, Inc., 1990.

[13] SCHULZRINNE, H. Voice communication across the Internet: A network voice terminal. Technical Report TR 92-50, Dept. of Computer Science, University of Massachusetts, Amherst, Massachusetts, July 1992.

[14] SCHULZRINNE, H., CASNER, S., FREDERICK, R., AND JACOBSON, V. *RTP: A Transport Protocol for Real-Time Applications*. Internet Engineering Task Force, Audio-Video Transport Working Group, Mar. 1995. Internet Draft expires 9/1/95.

[15] TAUBMAN, D., AND ZAKHOR, A. Multi-rate 3-D subband coding of video. *IEEE Transactions on Image Processing 3*, 5 (Sept. 1994), 572–588.

[16] TURLETTI, T. *INRIA Video Conferencing System (ivs)*. Institut National de Recherche en Informatique et an Automatique. Software on-line.[2].

[1] ftp://ftp.ee.lbl.gov/conferencing/vat
[2] http://www.inria.fr/rodeo/ivs.html

[17] TURLETTI, T., AND BOLOT, J.-C. Issues with multicast video distribution in heterogeneous packet networks. In *Proceedings of the Sixth International Workshop on Packet Video* (Portland, OR, Sept. 1994).

[18] Video codec for audiovisual services at p*64kb/s, 1993. ITU-T Recommendation H.261.