Video Coding for Streaming Media Delivery on the Internet

Gregory J. Conklin, *Member, IEEE*, Gary S. Greenbaum, *Member, IEEE*, Karl O. Lillevold, Alan F. Lippman, *Member, IEEE*, and Yuriy A. Reznik, *Member, IEEE*

Invited Paper

***Abstract*—We provide an overview of an architecture of today's Internet streaming media delivery networks and describe various problems that such systems pose with regard to video coding. We demonstrate that based on the distribution model (live or on-demand), the type of the network delivery mechanism (unicast versus multicast), and optimization criteria associated with particular segments of the network (e.g., minimization of distortion for a given connection rate, minimization of traffic in the dedicated delivery network, etc.), it is possible to identify several models of communication that may require different treatment from both source and channel coding perspectives. We explain how some of these problems can be addressed using a conventional framework of temporal motion-compensated, transform-based video compression algorithm, supported by appropriate channel-adaptation mechanisms in client and server components of a streaming media system. Most of these techniques have already been implemented in RealNetworks® RealSystem® 8 and its RealVideo® 8 codec, which we are using throughout the paper to illustrate our results.**

***Index Terms*—Internet media delivery networks, scalable video coding, streaming media, video compression.**

I. Introduction

SINCE its introduction in early 1990s, the concept of *streaming media* has experienced a dramatic growth and transformation from a novel technology into one of the mainstream manners in which people experience the Internet today. For example, according to recent statistics cf., [1], over 350 000 hours of live sports, music, news, and entertainment are broadcast over the Internet every week, and there are also hundreds of thousands of hours of content (predominantly in RealAudio® or RealVideo® formats) available on-demand.

Indeed, such a phenomenal growth would not be possible without adequate progress in the development of various core technologies utilized by streaming media software, and in particular, video coding. In this paper, we briefly review some of the important stages in the development of this field, explain various specific requirements that streaming poses for video coding algorithms, and describe solutions to some of these problems in today's industry-standard streaming media delivery systems.

Manuscript received June 15, 2000; revised December 7, 2000. This paper was recommended by Guest Editors M. R. Civanlar, A. Luthra, S. Wenger, and W. Zhu.

The authors are with RealNetworks, Inc., Seattle, WA 98121 USA (e-mail: gregc@real.com; garyg@real.com; karll@real.com; yreznik@real.com, alanl@real.com).

Publisher Item Identifier S 1051-8215(01)02235-2.

II. Evolution of Streaming Media Technologies

The concept of *streaming media* came at a time when basic multimedia technologies had already established themselves on desktop PCs. Audio and video clips were digitized, encoded (e.g., using MPEG-1 compression standard [2]), and presented as files on the computer's file system. To view the information recorded in such files, PC users ran special software designed to decompress and render them on the screen.

The first and most natural extension of this paradigm on the Internet was the concept of *downloadable media*. Compressed media files from the Web were expected to be downloaded on local machines, where they could be played back using the standard multimedia software. However, this was not a satisfactory solution for users with limited amounts of disk space, slow connection speeds and/or limited patience. This essentially created the need for *streaming media*, a technology that enabled the user to experience a multimedia presentation on-the-fly, while it was being downloaded from the Internet.

A. HTTP-Based Streaming

The design of some early streaming media programs, like VivoActive 1.0 [3], was based on the use of the standard (HTTP-based [4]) Web servers to deliver encoded media content. Since all HTTP server-client transactions are implemented using a guaranteed-delivery transport protocol, such as TCP [5], the design of these programs was very simple. For example, VivoActive used a combination of the standard H.263 [6] video and G.723 [7] audio codecs, and a simple multiplexing protocol to combine the audio and video streams in single file. These codecs came from desktop video conferencing, and only minor algorithmic changes (mostly related to rate control) were required to make such a system work.

However, being originally designed for serving static documents, HTTP protocol was not particularly suited for real-time streaming. For example, the lack of control over the rate at which the Web server pushes data through the network, as well as the use of the guaranteed-delivery transport protocol (TCP), caused substantial fluctuation in the delivery times for the fragments of the encoded data. This is why the VivoActive player used a quite large (5–20 s) *preroll buffer* that was meant to compensate for the burstiness of such a delivery process. Nevertheless, if for some reason the delivery of the next fragment of data was delayed by more than the available *preroll time*, the

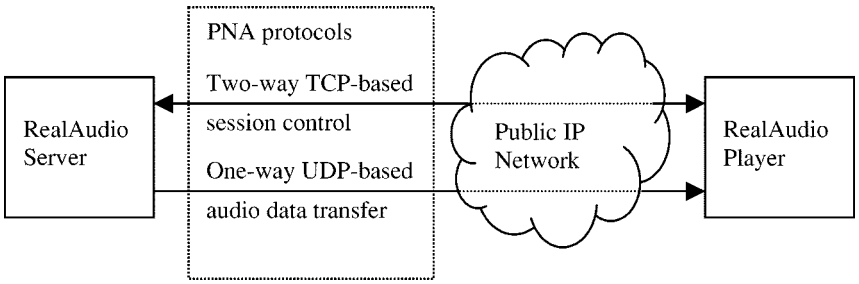

Fig. 1. Communication between RealAudio server and RealAudio player.

player had to suspend rendering until the buffer was refilled. This so-called *rebuffering* process was a frequent cause of diminished user experience.

Some other challenges of using standard Web servers were streaming of live presentations and implementing VCR-style navigation features such as seek, fast-forward, and rewind for on-demand streaming.

B. First Servers and Protocols for Streaming Media

The first complete streaming media package that featured both server and client components was RealAudio 1.0, introduced in March 1995 [8]. As shown on Fig. 1, the process of communication between RealAudio server and RealAudio player was based on a suite of dedicated, TCP- and UDP-based network protocols, known as Progressive Networks Architecture (PNA). These protocols allowed the transmission of the bulk of compressed audio packets to be done via a low-overhead, unidirectional UDP transport, and reduced the use of TCP to basic session control needs.

While the use of UDP transport enabled a better utilization of the available network bandwidth and made the transmission process much more continuous (compared to TCP traffic), it also introduced several problems such as *lost*, *delayed*, or *delivered out of order* packets.

To combat the damage caused by these effects, RealAudio used several mechanisms. First, both client and server implemented the Automatic Repeat-Request (ARQ) mechanism. This procedure allowed the client to re-request missed packets, and if they were successfully delivered within the available preroll time, the loss was recovered. Second, in case ARQ failed, a frame *interleaving* technique was used to minimize the *perceptual damage* caused by the loss of packets [9]. For example, before interleaving, a network packet may contain 10 continuous audio frames representing $1/4$ second of audio signal. The loss of such packet results in the loss of $1/4$ second of audio, which is clearly noticeable. On the other hand, after interleaving, this packet may contain 10 audio frames randomly collected from the last 10 seconds of audio. If such a packet is lost, the damage is spread across these 10 seconds, leading to a much less noticeable type of disruption.

C. First Video Codecs for Streaming Media

As we already mentioned, video codecs in VivoActive and some other early programs were directly derived from the ITU-T H.261 [10] or H.263 [6] standards. These codecs have been originally designed for low-latency, bit-level transmission scenarios in POTS-based desktop videoconferencing. However, the need to address many different requirements specific to UDP-based streaming delivery have resulted in the proprietary design of the RealVideo codec [11].

One of the most problematic initial requirements was the need to produce compressed data that can be streamed at some *fixed bit rate*. Normally, dynamic video clips have fragments that are hard to encode, such as scene changes, transitions, etc., interleaved with more or less static or slow-motion scenes, that can be compressed efficiently. This results in very unequal distribution of bits between frames when they are encoded with the same level of distortion.

In order to maintain a constant bit rate, the encoder has either to introduce unequal distortion when encoding frames, or skip encoding some frames (which, again increases distortion, unless the video is still), or do both. This appears to be an interesting optimization problem on its own, and while a dynamic programming-based algorithm for solving it has recently been found (cf. [12]), such a solution may still leave substantial distortions in the reconstructed signal.

Fortunately, with the availability of the preroll buffer, the constant bit rate requirement can be substantially weakened. Thus, we now only need to maintain the required bit rate *on average*, allowing the actual number of bits per frame to fluctuate within the bounds provided by the space available in the preroll buffer. These considerations have eventually led to the design of the variable-bit-rate (VBR) rate-control algorithm in RealVideo codec. A variant of such technique, called *bandwidth smoothing*, has been recently studied in [13].

Another difficult requirement was the need to support *random access* to video frames in a compressed file. Such access was needed to let users join live broadcasts or to let them rewind, seek, and fast forward on-demand video clips. To solve this problem, RealVideo codec periodically inserted Intra frames and modified the rate-control mechanism in a way that fluctuations in the quality of the encoded frames were minimized.

On the channel side, RealVideo codec had to deal with packetization and packet loss. Due to the large sizes of video frames, simple loss-distribution techniques such as *interleaving* could not be applied directly. Instead, RealVideo codec used a combination of *forward error correction codes* to protect the most sensitive parts of the compressed bitstream and various built-in *error concealment* mechanisms. Such combination of techniques is commonly referred to as *unequal error protection* [14].

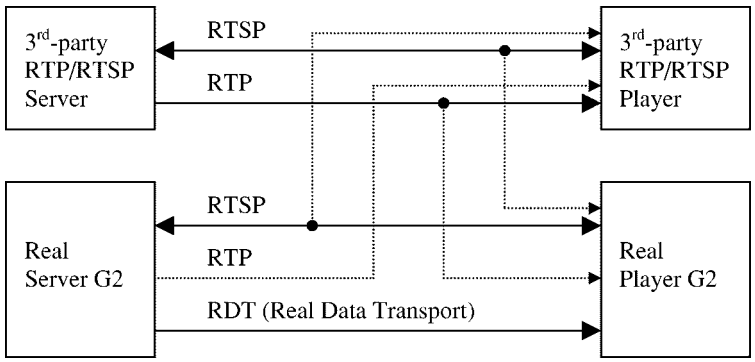

Fig. 2. Streaming protocols used by RealSystem G2, and its interoperability with other standards-based systems.

D. The Need for Scalable/Adaptive Streaming

Since the first programs for streaming media were only concerned with delivering streaming media content at some fixed bit rate (e.g., 14 or 28 kbits/s), it was only sufficient to serve the needs of users with identical speeds of Internet connection. Users with faster connections could not experience any benefits of extra bandwidth, while those with slower connections were not been able to view the content at all.

A preliminary solution to this problem was to create and serve multiple versions of the same content, encoded for some specific classes of the Internet audience. For example, a publisher of a streaming media presentation had to create separate versions of it for users of 28K and 56K modem connections, ISDN lines, etc.

Undoubtedly, this solution had many obvious problems. First, it was based on the assumption that the actual bandwidth of the channel between server and client is bounded only by the last link in the chain (i.e., client's connection to the ISP), which is not always true. Also, it did not address the possibility of dynamic changes in channel bandwidth and loss statistics.

To maintain connection when bandwidth changes early streaming media servers implemented so-called stream *thinning* mechanism. When a server was notified that packets were delivered to the client slower than real-time, it began skipping transmission of some of the packets. Such a technique certainly introduced the loss of data, but it was sufficient to prevent players from rebuffering or losing connections.

E. RealSystem G2

A much more comprehensive solution to the problem of serving multiple audiences and making such serving *adaptive* was provided by RealSystem G2 and its SureStream™ technology, introduced in 1998 [15], [16].

The key idea of SureStream is to use the encoder to produce multiple representations (or *streams*) of the original content, optimized for various channel conditions. These encoded streams are then stored in single *SureStream* file, in a form that facilitates their efficient retrieval by the server. During the streaming session, a client (RealPlayer G2) monitors the actual bandwidth and loss characteristics of its connection, and instructs the server to switch to the stream, whose transmission over the current channel would yield the minimum distortion in the reconstructed signal.

The use of client-side processing offered at least two major benefits. First, it greatly reduced the complexity of server-side processing needed to support stream selection, and thus, increased the number of simultaneous connections the server would be able to maintain. Second, it allowed a very simple extension of SureStream mechanism for *multicast* delivery: if encoded streams are assigned to different multicast addresses, all the client has to do is to subscribe and unsubscribe them dynamically using the same rate distortion minimization process.

It is important to note that the implementation of SureStream services in RealSystem G2 is not tied to any particular file format or video coding algorithm. In Section VI, we will provide a more detailed description of the SureStream framework, and will show how it can be used to take advantage of various *scalable video coding* techniques, for channel adaptation.

In addition to many other technological advances, RealSystem G2 marked an important phase in the development of Internet streaming infrastructure, being the first system built on the IETF and W3C standards for Internet multimedia. As illustrated in Fig. 2, in place of the proprietary PNA protocol, RealSystem G2 used the standard RTSP protocol [17] for session control, and supported the RTP standard [18] for framing and transporting of data packets. RealSystem G2 was also one of the first systems that embraced the W3C SMIL standard [19] for multimedia presentations.

F. Distributed Media Delivery Networks

In spite of the dramatic progress in improving the performance of software and hardware for streaming media servers, it become apparent that a single server is capable of serving only a very limited subset of the potential Internet audience. Moreover, a single server-based delivery system faces several major problems from network utilization point of view. The amount of traffic it pushes through the public IP network is always a linear function of the number of subscribed clients. Even if the information sent to all clients is the same (e.g., transmission of a live video event), it still has to be sent individually. Besides of generating large quantities of redundant IP packets, this also creates a strongly asymmetric (centered around the server) distribution of load on local network infrastructure. Under certain circumstances, all these factors can cause network congestion, which in turn, degrades the quality of service provided by such a system, or even worse makes it completely nonfunctional.

In certain cases, such as distribution of live content, the load on the streaming media server can be reduced if the network supports *multicast routing*. The server only sends a single stream, and multicast routers replicate it to all subscribed clients. Unfortunately, in spite of being an area of very active and challenging research in the past few years, the practical use of multicast still remains very limited. In part this could be explained by the costs and slow deployment rates of the required multicast equipment, as well as by the existence of various reliability problems related to this distribution method.

This motivated the development of so-called *application-level* multicast networks, that use multiple intermediate servers that re-broadcast incoming packets to their respective clients. A well-known early example of such a network used for videoconferencing was the Multicast Backbone (MBone) [20].

Similar to MBone, today's *streaming media delivery networks* employ multiple, geographically distributed servers. They differ however, in the ways they implement the distribution of the encoded content between these servers, and the mechanisms they use for redirecting clients to their local (and/or least busy) servers.

Thus, in the simplest case, a delivery network may be composed of various distributed servers, which do not have any information about each others' existence. Such servers are typically installed by ISPs and large corporations to minimize the amount of traffic coming into their local networks. This is achieved either by *splitting* incoming live streams to all subscribed clients on the local network, or by *caching* most frequently used on-demand content on local storage.

A more comprehensive (and more commonly used) solution is provided by delivery networks that use *dedicated* (guaranteed-bandwidth) connections between their servers. Typically, there are certain costs associated with usage of dedicated channels, and the minimization of traffic in such networks becomes a very important problem.

Another (and relatively new) way of building streaming media delivery networks is based on the use of multiple-access transmissions over the public Internet. In its simplest form, such a delivery system sends the requested information from several different locations concurrently, and collects packets that arrive first (or arrive at all) at the receiver end. In a more general case, such a system may employ special distributed coding, such that receiving and joint decoding of the information from multiple transmitters yields a lower level of distortion than any one of the individual streams.

G. Video Coding and the Next-Generation Media Delivery Systems

Overall, today's streaming media distribution involves transfers of audio and video information through a number of intermediate servers before it reaches subscribed clients. The final stage in this process, the server–client transmission, has long been a central problem for streaming media systems. However, with the growth of the Internet infrastructure and intensive deployment of streaming media delivery networks the focus is now shifting toward optimizing the overall quality of service provided by such delivery systems. Typical constraints for such optimization are topology of the network, bandwidth and maintenance costs of its internal channels, storage capacities of its servers, etc.

Under certain restrictions (such as the use of lossless channels, use of intermediate servers only to replicate the incoming data, etc.), analysis of flows in such networks can be viewed as one of the well known graph-combinatorial problems [21]–[23].

A more complete settlement of this problem involves the study of the *information flow* in such networks [24]. Some recent results in this theory [25] explain, for example, that simple multicast routing is not sufficient for achieving optimal bandwidth usage. On the other hand, networks that employ transcoding of the information at every node (router), can potentially be optimal [25].

All these factors highlight some new ways video coding can be used in the next generation streaming media delivery systems. Finding efficient solutions of the corresponding coding problems will be increasingly important for further progress in this field.

III. Streaming Media Delivery Mechanisms

It is important to distinguish between two modes in which video information can be distributed over the Internet, namely, *live broadcasting* and *on-demand streaming*. Below, we consider each of these models and the corresponding *delivery mechanisms* used by modern streaming media systems.

A. Distribution of Live Video

A diagram illustrating various steps in the distribution of live content is presented in Fig. 3. The source of live video information (such as any standard analog video recorder) is connected to the *encoder*. The encoding engine is responsible for capturing and digitizing the incoming analog video information, compressing it, and passing the resulting data down to the *server*. Alternatively, the server can receive such information from a *Simulated Live Transfer Agent* (*SLTA*), a software tool that reads pre-encoded information from an archive and sends it to a server as if it has just been encoded from a live source.

The server is responsible for dispersing the compressed information from the encoder to all connected *splitters* and/or *clients* who have joined the broadcast. Splitters are additional servers that can be either part of a dedicated media delivery network, or a public-IP-based multiple-access delivery network, or can be embedded in network traffic caches, which in case of live streaming broadcasts just pass the information through.

In its simplest form, the server (or splitter) *unicasts* the encoded video information to each of the clients individually using a one-way data stream (combined with two-way RTSP session control). In this case, the parameters of the connection between server and each client can be estimated at the beginning of each session and can be systematically monitored during the broadcast.

In the case where a network is equipped with multicast-enabled routers, the server needs to send only one *multicast* stream, which is automatically replicated to all subscribed clients on the network. Important limitations of multicasting

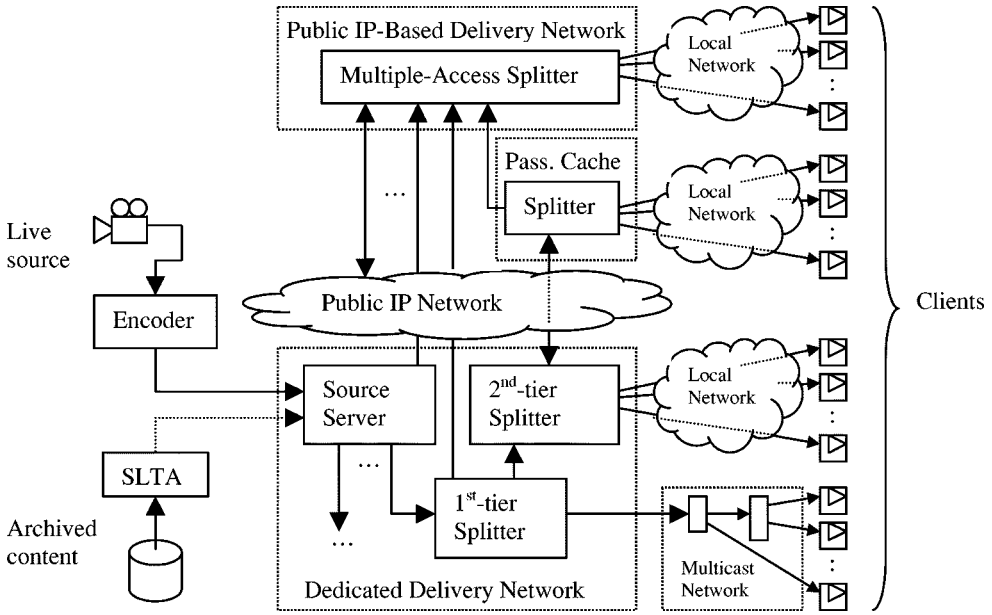

Fig. 3. Delivery of live and/or simulated live content.

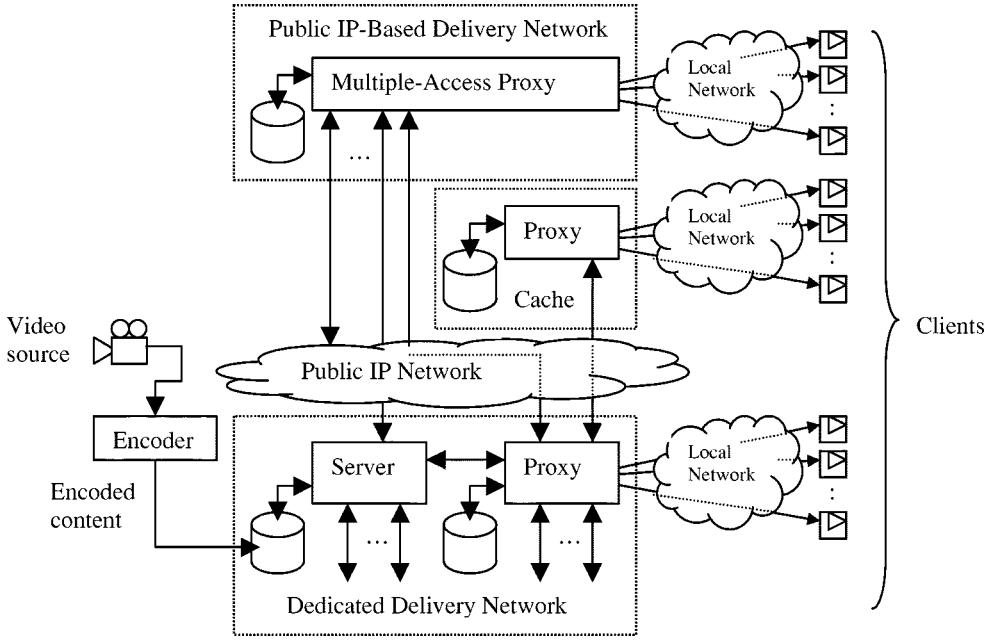

Fig. 4. Delivery of on-demand content.

are one-way transmission and nonguaranteed delivery of information. In addition, the server does not typically know how many clients are subscribed to the broadcast and/or their actual connection statistics. A possible way to serve clients with different connection speeds is to *simulcast* several independent encoded versions (streams) of the source targeted for different bit rates, and let clients decide which stream to use.

In addition to the server-client transfers, streaming media networks also have to distribute encoded video information between their splitters. There are several possible ways such distribution can be implemented by the network. In one possible implementation, splitting is initiated by the source server, which broadcasts information to all directly connected splitters, and so on. We call such process *push splitting*. Alternatively, splitting can be initiated by a client connecting to a local splitter (or network cache acting as a splitter) which, if not active, transfers request to an upper tier splitter, and so on, until it reaches the nearest active splitter. Once such a splitter is found, it can start transmission of the requested information down through the chain of intermediate connections to the client. We call this model *pull splitting*. In the case where a splitter is used as part of a *multiple-access* delivery network, it can establish connections to several geographically distributed upper-tier splitters. We call such a delivery process *multiple-access* splitting.

B. On Demand Distribution

We illustrate the steps in another distribution model, on-demand streaming, in Fig. 4. One of the major differences between

this diagram and the one for live broadcast (see Fig. 3) is that there is no direct connection between the encoder and the server. Instead, a compressed video clip has to be recorded on disk first, and then the server will be able to use the resulting compressed file for distribution. This also allows remote *proxy* servers to use their local storage to *cache* the most frequently used media clips.

Server–client communication for delivering on-demand content is essentially the same as unicast streaming of live content. The main difference is that with on-demand content user is allowed to rewind and/or fast forward the presentation, while such controls are not available for live (or simulated live) broadcasts.

Unlike push- and pull-splitting of live content, the server–proxy transfers can only be initiated by the client. Moreover, at the time of the transfer, the proxy may already have some information about the requested clip in the local storage. Using proper coding techniques, such information can be used to reduce the rate of the requested additional stream to the proxy.

TABLE I
COMMUNICATION PROCESSES IN STREAMING MEDIA

Delivery mechanism	Structure of the communication process		
	transmitter(s)	receiver(s)	side info.
unicast streaming	server or splitter or proxy	player	–
multicast streaming	server or splitter	multiple players	–
splitting	server or splitter	splitter	–
multiple-access splitting	multiple servers and/or splitters	splitter	–
proxy update	server or proxy	proxy	cached data
multiple-access update	multiple servers and/or proxies	proxy	cached data

IV. PROBLEMS IN VIDEO CODING FOR STREAMING MEDIA DELIVERY

As we have already described, today's streaming media delivery is a complex process, involving various types of transmission of video information over distributed, heterogeneous networks. Based on the number of transmitters, receivers, and the availability of the correlated information at the receiver's end, such transmissions may lead to different problems from source coding and communication (characterization of the capacity region of the network) points of view. In turn, since all encoded data are carried over an IP network, the characteristics of these transmissions will depend on the actual parameters and state of such network. Some other factors that pose additional constrains on video coding are usage model (e.g., availability of rewind, fast-forward, and seek functions), and processing power available to senders and receivers.

Below, we explain each of these aspects of streaming media delivery, and describe specific problems they pose for video coding.

A. Problems Imposed by the Type of Communication

We summarize some of the properties of the communication processes introduces in Section III in Table I.

In the case of *unicast streaming*, we clearly have a classic *point-to-point communication*, which leads to (separable, under certain conditions) *source* and *channel* coding problems.

In the case of *multicast streaming*, we have to deal with unknown multiple receivers that may have different loss characteristics of their connections. This problem is commonly known as communication over the *broadcast channel* [24].

Similar to multicast, information distributed via *splitting* is intended to be received by a number of clients with various (and unknown to the sender) types of connections. However, the intermediate transfers between splitters are the standard point-to-point communication processes. In other words, if splitters are not allowed to transcode the data they receive, splitting requires a coding technique that is optimal for both broadcast and point-to-point communication.

An even more interesting coding problem arises with *multiple-access splitting*. Such splitters request and receive information from multiple sources, which can be considered a form of *multiple-access* channel [24]. In turn, the information received by splitters is intended for multiple clients. The combination of both requirements leads to a variant of the problem of *source coding for multiterminal networks* [26], [24].

Considering *server-proxy communication*, it is important to study a case when the proxy contains some pre-cached information about the requested video clip, or some other correlated sequence. In the lossless case, the corresponding source coding problem can be treated using the *Slepian-Wolf* theorem [27]. In a lossy case, we have a problem of *source coding with side information* [28].

Finally, we need to consider communication between *multiple-access proxies*. Thus, if a multiple-access proxy needs to get new data, it initiates several connections to two or more other (geographically distributed) proxies, and hopes to use the data it receives from all channels to minimize the distortion in the reconstructed signal. This is a well known *multiple description* coding problem [29]–[31].

B. Problems Associated with IP-Based Delivery

The heterogeneous and time-variant nature of today's IP networks presents a number of challenges for implementing real-time communication systems. First, depending on the actual network path, characteristics of routers and communication channels used to transmit packets from one point to another, parameters of such connection can vary by several orders of magnitude. For example, the bandwidth can vary from hundreds of bits to megabits per second, the packet loss probabilities can vary from near zero to tens of percent, and the delivery delay can vary from milliseconds to seconds. Second, all the above parameters can vary in time, depending on the current distribution of load in the network.

A possible way to address some of these problems is based on the idea of *adaptive client-driven serving* (or *receiving* for multicast delivery) of streaming media content. Thus, in the ma-

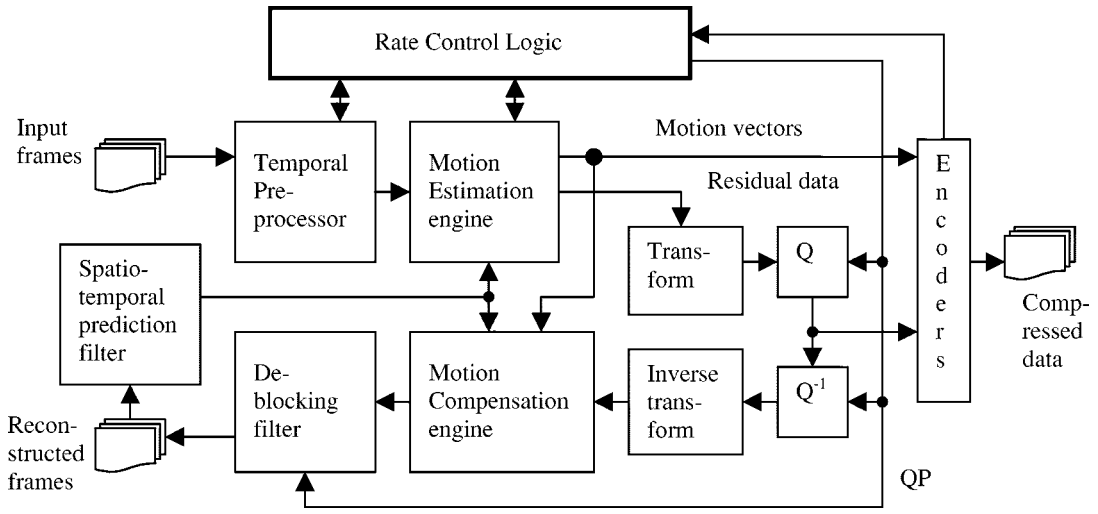

Fig. 5. Motion-compensated hybrid coder.

jority of cases clients can monitor the rate and loss statistics for the arriving packets, and instruct the server on how to adjust encoding and/or transmission rate.

In the simplest case, used only for transmission rate adjustments, such a technique may have a function of *congestion control*. This is a well known problem, and examples of works in this direction include control schemes for layered multicast [32] and TCP-friendly transmissions [33], [34].

The *error control* is another problem arising when the transmission rate is given by the congestion control algorithm, and the goal is to use this bandwidth to minimize effects of the packet loss. A variant of this problem, in a context of the adaptive layered multicast system has been recently studied in [35].

Dynamic prediction of bandwidth and loss parameters is an important integral component of both congestion- and error-control schemes. Given sufficient temporal window (preroll time), such algorithms can use a broad set of statistical techniques, and/or some known empirical phenomena. For example, it is well known that fluctuations of the Internet traffic have a fractal-like scaling behavior over time scales [36].

C. Random Access

Both on-demand and live streaming require random access to compressed video information. For on-demand video content, the end-user should be able to fast forward, rewind, and seek through the presentation. For live content, the end-user must be able to connect to the broadcast throughout the event.

Random access may also be needed to implement adaptive serving. For example, a server may have several pre-encoded versions of the same content, and when it receives the request to change transmission rate, it simply starts streaming one that fits in the requested data range.

In the simplest case, random access capability can be implemented by independent encoding of relatively small (1–5 seconds) blocks of the video content. In the framework of a motion-compensated transform-based video coding, this translates in the insertion of I-frames at the boundaries of such intervals. Unfortunately, this technique has a negative impact on the achievable compression rates, and prevents the use of many powerful *universal* and *asymptotically optimal* data compression schemes.

An alternative to the use of I-frames is an architecture proposed in [37]. This system uses special "S"-frames to implement joining and switching between (continuously) encoded streams.

D. Heterogeneity of Processing Resources

Another, and not-yet mentioned aspect of the Internet is the *heterogeneity of the processing power* available at its terminals. Streaming media presentations can be received on variety of computing devices, ranging from powerful workstations and desktop PCs to set-top boxes and low-power handhelds, such as cellular phones and PDAs.

This creates the need for coding techniques that support *complexity-scalable decoding*.

Availability of the processing power is also an important factor for the design of the encoding algorithms. For example, encoding of the on-demand presentations can typically be done off line and the use of high complexity encoding algorithms is possible and desirable in such a scenario. On the other hand, content encoding for live broadcasts must be done in real-time, frequently on a computer with limited resources, and the encoding algorithms must be able to scale its complexity to deliver best possible quality under such constrains.

V. Motion-Compensated Hybrid Video-Coding Algorithms for Streaming Media

During the last two decades, the problem of source coding of video information has been an area of extremely active research, leading to various successful deployments of such algorithms in practice, and industry-wide standardization activities. A practically important result of these efforts was the selection of a motion-compensated hybrid transform-based compression scheme as a basis for all currently adopted standards on video coding (such as MPEG-1 [2], H.261 [10], H.263 [6], etc.).

We present the structure of a generic single-rate motion-compensated hybrid codec in Fig. 5. Input video frames are passed to the *temporal preprocessor*, which decides if a frame should be

coded, detects scene cuts, selects a prediction model (unidirectional, or bi-directional), etc. The *motion estimation engine* normally performs an RD-constrained block-based motion search in the field generated by the *spatio-temporal prediction filter*. The block sizes used for motion search vary in different standards from 4×4 to 16×16 pixels. Likewise, implementations of prediction filter can vary from a simple 1/2-pixel accurate bilinear spatial interpolation, to sophisticated spatio-temporal filtering techniques [38]. The residues after motion estimation are passed to an energy-compacting *orthogonal transform*. To implement such a transform, most of the standard video codecs use the 8×8 DCT kernel [39]. The resulting transform coefficients are quantized and sent to a lossless statistical *encoder*.

To stay synchronized with the decoder, the encoder replicates parts of the decompression loop, namely, dequantization, inverse transform, motion compensation engine and an adaptive *deblocking filter*.

The *rate-control* algorithm is used to select the step of quantization and also to provide input information for the temporal pre-processor and motion compensation engine in order to maintain output bit rate within certain limits.

A. Scalability Modes

It should be stressed that the first motion-compensated hybrid video coding algorithms have been designed to produce a single-rate encoded version of the input signal. Being suitable for point-to-point transmissions over stationary channels, such algorithms do not address the needs of other communication scenarios, such as multicast or multiple-access transmission. To extend the range of applicability of these schemes, a number of special *scalability* modes have been proposed.

In the simplest case, a communication system may encode a given content several times, producing redundant, independently encoded streams, specifically optimized for several possible types of channels. If the goal is to implement a scalable multicast system, all these streams can be sent via multicast channels simultaneously, and the clients would have to decide which of the streams will work in the best way for their current channel conditions. This technique is known as *simulcast*.

However, such a coding scheme may not be optimal for splitting, where it is also necessary to have the combined representation of all streams as small as possible. These requirements can be addressed by using codes based on the principle of *successive refinement* [40]–[42].

In the context of motion-compensated hybrid video coding, this idea led to the development of a *scalable* coding technique, based on spatio-temporal pyramid decompositions of the source signal [43], [44], or factorization of the quantizer step sizes used. For example, the H.263+, Annex 0 specification [45] describes separate temporal (B-frames), spatial (resampling), and SNR (quantizer size) scalability modes.

In practice, simple SNR scalability techniques of [45] do not attain the performance of the redundant, multiple bit rate encodings [46]. However, with the use of more advanced quantization schemes proposed in [44], such differences can be made less noticeable.

To address the needs of multiple-access communication, it is necessary to employ multiple-description coding. Using the motion-compensated hybrid video-coding framework, such a goal can be achieved by replacing its scalar quantization with an appropriate *multiple-description quantization* scheme [47], [48].

B. Emerging Video-Coding Standards for Streaming Media

As we discussed earlier, streaming media poses various additional problems for video coding. Problems associated with IP-based delivery, availability of preroll delay, random access, processing power scalability, etc., have not yet been addressed by the existing standards for video communication.

One of the emerging standards that has a potential to cover these issues is the ITU-T SG16/Q15 H.26L project [49]. The corresponding requirements for this standard have already been provided in [50].

Improved video coding technologies (cf. [51]–[53]) and tools for supporting streaming media applications (Streaming Media Profile) are also the focus of the ISO/IEC JTC1/SC29/WG11 MPEG Group.

VI. Video Processing in RealSystem 8

As we have already pointed out in Sections II-D and IV-B, it is necessary to perform an *adaptive* channel (or joint source/channel) encoding of the streaming content in order to match the actual bandwidth and loss statistics of the channel. In turn, since the channel information becomes available only at the time of serving of the content, such adaptive source/channel coding could only be done in the server.

However, there are certain restrictions on the complexity associated with maintaining each of the connections in the server. For instance, today's streaming media servers are designed to be capable of serving thousands of clients per CPU, and thus, only very simple types of processing can be done on the bitstream level.

The last set of requirements is satisfied by distributing the majority of the actual source/channel coding work to the *encoder*, and leaving the server only to complete the final stages of this process. Some of the possible ways to accomplish partition of the encoding process are:

- the use of *redundant, independently encoded streams*, specifically optimized for several possible channel conditions, and the bandwidth/error rate prediction logic in the client and/or the server that selects the stream to transmit based on the actual behavior of the channel in the past;
- the use of *scalable source coding techniques* in the encoder, leaving the server an opportunity to trim the code to the appropriate bit rate before the transmission;
- the use of the *multiple-description codes* for multiple-access distribution of the content.

To support efficient implementation of such types of adaptive video encoding/serving processes, RealSystem 8 offers an extensive set of tools and public APIs, known as SureStream technology [54]. The key components of this framework are:

- *Adaptive Stream Management (ASM)* protocol;
- *SureStream* file format access and rendering mechanisms;
- actual *source and channel coding algorithms* implemented by a set of *plug-ins* (such as RealVideo 8 codec plug-in) that can be attached to the system.

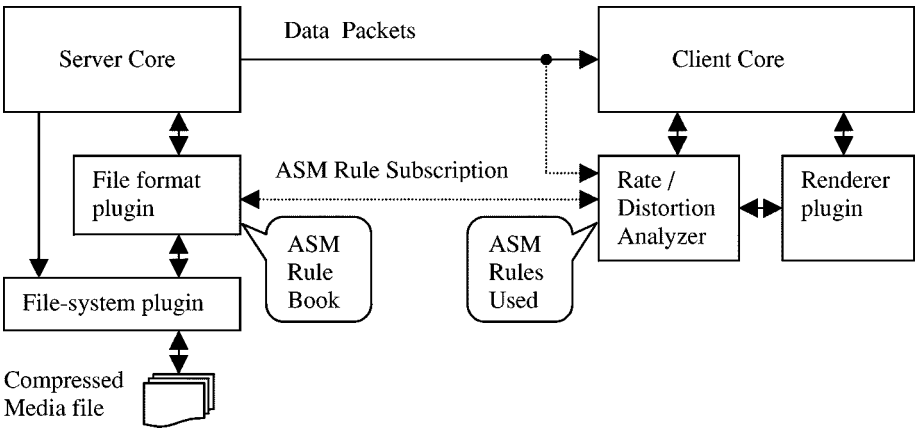

Fig. 6. Adaptive Stream Management in RealSystem 8.

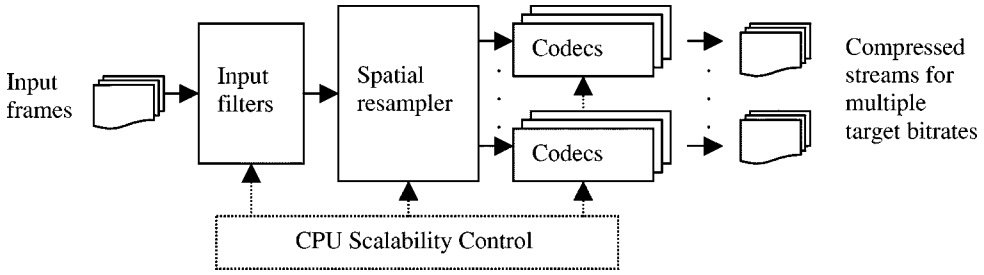

Fig. 7. The structure of the RealVideo 8 encoding module.

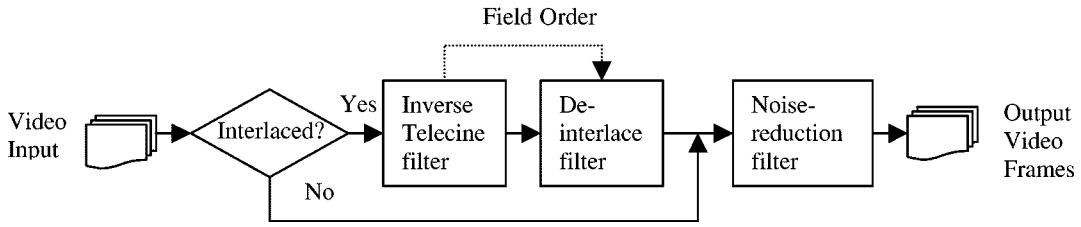

Fig. 8. Input filters in RealVideo 8.

A. ASM

ASM is a mechanism that allows the client (RealPlayer) to communicate efficiently the type of the encoding that should be "synthesized" by the server in order to minimize the distortion of the received information.

We present the structure of the server's and client's components involved in the ASM process in Fig. 6. Compressed media files are accessed by server with the help of the *file system* and *file format* plug-ins. The file format plug-in has knowledge about the way data are compressed and stored in the media file, and is capable of producing various combinations of the encoded streams as they are requested by the client.

To produce such combinations, the file format plug-in uses so-called *ASM rules*. These rules are based on a sophisticated, fully programmable syntax and can be used to describe various means of channel adaptation ranging from simple priorities assigned to different packets, to expressions describing various combinations of bandwidth, packet loss, and effects of loss on the reconstructed signal that can be measured by the client.

The complete set of the *ASM rules* is stored in the compressed media file as the *ASM rule book*. At the initial phase of the communication, the ASM rule book is transferred to the client. In turn, the client collects the information about the channel, parses the AMS rule book, and sends the server a request to *subscribe* to a rule or combination of rules that match current statistics in the channel.

When the server receives the request to subscribe to a rule, it passes it to the file format plugin, which in turn begins to mix data according to its knowledge of their structure.

It should be noted that ASM is a general and format-independent technique. The actual syntax of ASM rules can be defined differently for various datatypes, and the actual logic of using them can be fully defined in their respective file format and rendering plug-ins.

B. The Structure of the RealVideo 8 Algorithm

The overall structure of the RealVideo 8 encoding process is presented in Fig. 7. Digitized and captured video frames (or fields) along with their timestamps are passed to a set of *input filters*. These filters are mainly needed to remove the noise and some specific artifacts that could have been introduced by edits and conversions of the video signal.

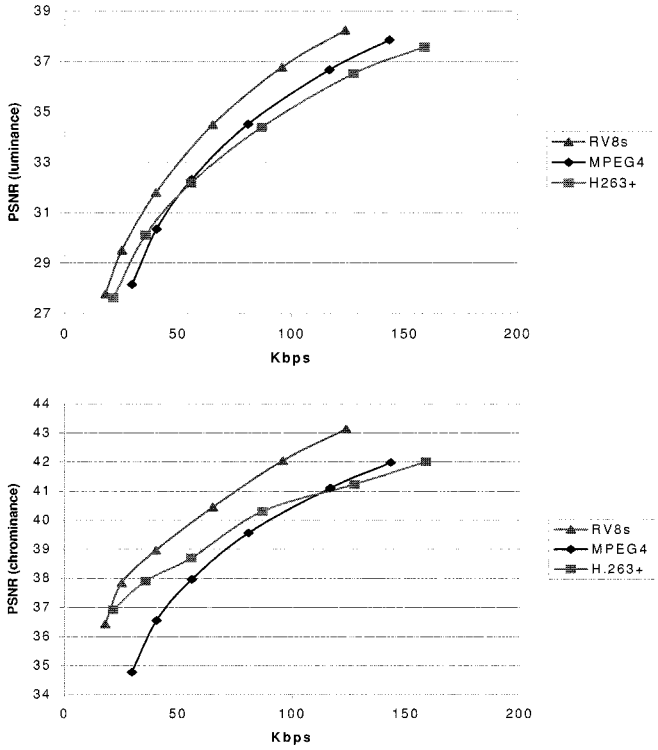

Fig. 9. RD comparison charts for clip "Foreman," QCIF, 7.5 fps.

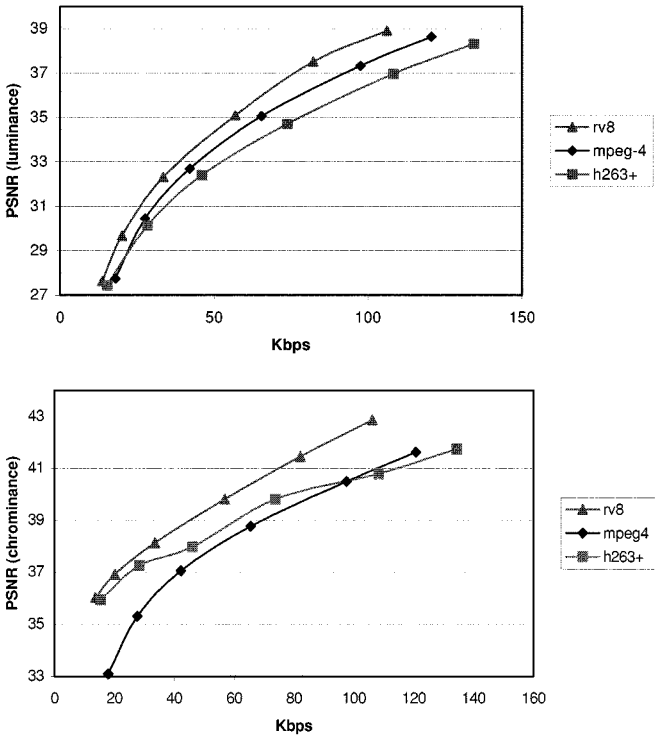

Fig. 10. RD comparison charts for clip "News," QCIF, 15 fps.

The output of the filtering engine is connected to a *spatial resampler*. The purpose of this block is to *downscale* input frames to a set of spatial resolutions that are suitable for encoding at various output bit rates. The optimal selection of such resolutions depends not only on the set of target bit rates (which is typically known), but also on the type of the content and type of distortions (e.g., smoothness versus clarity of individual frames) that the expected audience will be more willing to tolerate. For this reason, RealProducer 8 allows content creators to select one of the four modes: "smooth motion," "normal motion," "sharpest image," and "slide show" when encoding a clip. Later, this selection is used to deduct the desired tradeoff between spatial and temporal resolutions for each target bit rate.

After passing the *resampler*, video frames of various resolutions are forwarded to a set of actual video *codecs*, which are configured to produce encoded *streams* for a given set of target bit rates. Potentially, after receiving the data, each codec can work independently from the others. However, since some of the operations (such as *scene-cut detection*, *motion compensation*, etc.) that are performed for each stream may have the same or very similar results, these codecs are designed to share their intermediate data, and use them to reduce the complexity of the overall encoding process.

The last element on Fig. 7 is the *CPU scalability control* module, which tracks the actual processing times in various points of the RealVideo 8 engine, and sends signals to the appropriate algorithms if they should switch to lower complexity modes. This type of control is essential to maintain the best possible quality level for encoding live presentations, especially when the encoding machine is being used for other needs. Below, we describe some of the components of the RealVideo 8 encoding algorithm in greater details.

1) Pre-Filtering: The primary purpose of input video filters in RealVideo 8 is to remove noise and other artifacts that lower the perceptual quality of the original signal. In doing so, these filters can actually remove special types of *irrelevant* (and, in some cases, also *redundant*) information, thus helping the main video compression algorithm to achieve its goals.

As presented in Fig. 8, the actual sequence of filters used depends on the type of the video information that is being processed. If a signal is captured from a digital source, such as a USB camera, it is already presented in *progressive* form, and we may only want to pass it through a filter that removes low-energy spatial noise. On the other hand, if the signal is being from an analog NTSC (or PAL/SECAM) source, such as a TV tuner, camcoder, or VCR, we are dealing with an *interlaced* and potentially *edited* video signal, and thus, additional filters can be applied.

Several of our filters are designed to remove artifacts of the capture process that become obvious when the captured frames are displayed in a progressive mode (such as on a computer monitor). The *De-interlace* filter is designed to intelligently combine information from odd and even *fields* of NTSC (or PAL/SECAM) video signal, such that the shapes of moving objects are preserved as continuous. In effect, this filter outputs complete, progressive video frames without introducing "jaggy"-shaped artifacts, common for most of the simple NTSC converters. The *Inverse Telecine* filter is designed to remove the effects of the telecine process. Film is a progressive media that is composed of 24 frames per second (fps). NTSC video is an interlaced media at a rate of 29.97 frames (or 59.94 fields)

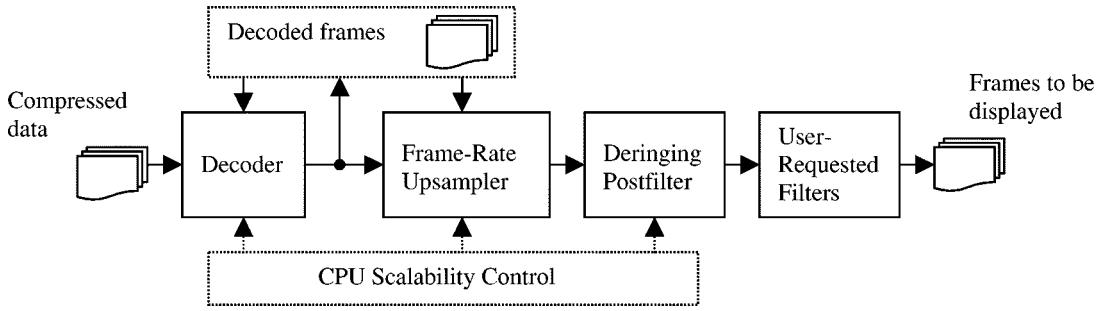

Fig. 11. Client-side video processing in RealSystem 8.

per second. The telecine process injects the additional 5.97 frames (or 11.94 fields) per second. These frames are clearly redundant and should be removed before the encoding process. Also, due to possible edits of the NTSC-converted film, the regular pattern of the inserted frames (fields) can be changed. For this reason, our *Inverse Telecine* filter detects all changes in order of fields and passes this information to the de-interlace filter to make sure that it combines them in proper order.

2) Core Algorithm: The core, single-rate video compression algorithm in RealVideo 8 is essentially a motion-compensated hybrid scheme, similar to ones we have described in Section V.

It is somewhat more sophisticated in various respects (such as motion prediction [55], adaptive transform sizes, and advanced statistical models) and demonstrates better coding gain compared to well-known standard codecs. We illustrate this in Figs. 9 and 10, where we compare the RealVideo 8 with MPEG-4 and H.263+ algorithms. Here we can observe improvements on the order of 0.5–2.0 dB relative to H.263+, and around 0.5–1.0 dB compared to MPEG-4 codec.

In the above tests, the MPEG-4 bit streams were created using the latest MoMuSys reference software (Version FPDAM1-1.0-000 608) with all implemented and relevant tools of the Advanced Coding Efficiency profile: quarter-pel motion compensation, ac/dc prediction, 4-MV, unrestricted motion vectors. To produce H.263+ bit streams, we used the TMN10 model with enabled Annexes I, J, and T. All codecs used fixed frame rate and fixed quantization (no rate control) mode, and motion search was restricted to ± 16 pixels range. Both MPEG-4 and H.263+ codecs used exhaustive search.

3) Scene Detection and Rate Control: Rate control determines which frames are coded and the number of bits or quality level of the encoded frames. At lower bit rates, one of the greatest impacts on the perceived quality of coded video is the relationship between framerate and frame quality. At higher bit rates it is essential to maintain full framerates, and the difficulty is to to maintain the appearance of uniform (high) quality as well.

Rate control for RealVideo has two major modalities: single-pass and two-pass rate control. In the single-pass mode (which is always used for live encoding), rate control can be described as trying to pick the number of frames to skip until the next frame is encoded and the "correct" quality and type (for example, intra- or inter-coded) for that frame. Knowledge of both the current and previous unencoded frames and previous encoded frames is used. For two-pass encoding, knowledge of future frames and the efficiency with which they can be coded are also used in the rate-control process.

Several—sometimes conflicting—requirements drive our rate-control choices. Some of these requirements come from the streaming aspect of real video. One primary streaming requirement is that users have the ability to easily join live streams, and also be able to seek within on-demand content. Another is that the coding should be resilient to loss. The solutions to these requirements partially come from rate control and intelligent choices of frames and macro-blocks to be intra-coded. However, these requirements must balance against nonstreaming requirements such as maximizing the quality of the content, which usually means minimizing the short-term variability of the frame-rate and quality level. For example, encoding three frames in quick succession followed by a long temporal gap and three more frames is much less desirable than encoding four or five or six frames at a more regular pace.

In the most recent versions of our encoding tools, we have started to allow content creators to adjust some of the fundamental parameters of rate control. The most significant of these is the temporal depth of the preroll buffer. As mentioned earlier, this buffer limits the amount of bit averaging that is allowed in encoding process. A larger buffer allows a coder to muscle through short difficult to encode sections, such as a pan or fade, with no degradation in video quality. The downside of a larger buffer is increased startup latency of a streamed presentation. By allowing our content creators to make this choice, we give them the flexibility to tailor encoding to their audiences.

As an overall design feature, it has been our experience that having a core rate-control algorithm that can make all of these trade-off decisions (in terms of explicitly specifying a cost function and then attempting to minimize it) while taking into account the content creators' desires (such as the ability to support a larger preroll buffer) yields a significant improvement in video coding quality.

4) Client-Side Video Postprocessing: In addition to the basic video compression/decompression services, RealVideo 8 also offers several postprocessing filters aimed at improving the subjective quality of the video playback. The structure of these filters is presented in Fig. 11.

After the decompression, video frames and their motion-vectors are sent to the *frame rate upsampler*, which is a special temporal filter that attempts to interpolate intermediate frames. We found this technique especially useful for low-bit rate encoded content, where original frames are regularly skipped.

The next block is a deringing filter used to reduce ringing artifacts common for most of the transform-based codecs.

Finally, RealPlayer offers a variety of additional effects, such as sharpening filter, color controls, etc.

VII. Conclusion

In this paper, we provided an overview of the architecture of today's Internet streaming media delivery networks and various problems they pose for video coding.

We also explained some of the existing mechanisms in RealSystem 8 that support adaptive transmission of pre-encoded information, and described the overall architecture of its RealVideo 8 codec.

We showed that RealSystem 8 provides an open and extensible platform, capable of accommodating various future needs of streaming media infrastructure on the Internet, and in particular, new demands for improved video-coding techniques.

References

[1] RealNetworks website, RealNetworks Facts. (2001), Seattle, WA. [Online]. Available: http://www.realnetworks.com/gcompany/index.html

[2] D. J. LeGall, "MPEG: A video compression standard for multimedia applications," *Commun. ACM*, vol. 34, pp. 46–58, 1991.

[3] Vivo Software web site, VivoActive software and documentation. (1997). [Online]. Available: http://www.vivo.com/help/index.html

[4] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol—HTTP/1.1," RFC: 2616, June 1999.

[5] *TCP: Transmission Control Protocol*, DARPA Internet Program, Protocol Spec., RFC: 793, Sept. 1981.

[6] *Video Coding for Low Bitrate Communications*, ITU-T Recommendation H.263, Nov. 1995.

[7] *Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s*, ITU-T Recommendation G.723.1, Mar. 1996.

[8] Progressive Networks. (1995, Apr.) Progressive Networks Launches the First Commercial Audio-on-Demand System Over the Internet (press release). [Online]. Available: http://www.realnetworks.com/company/pressroom/pr/pr1995

[9] R. Glaser, M. O'Brien, T. Boutell, and R. G. Goldberg, "Audio-on-Demand Communication System," U.S. Patent 5 793 980, Aug. 1998.

[10] *Video Codec for Audiovisual Services at $p \times 64$ kBit/s*, CCITT Recommendation H.261, 1990.

[11] Progressive Networks. (1997) Progressive Networks Announces Real Video, The First Feature-Complete, Cross-Platform Video Broadcast Solution for the Web (press release). [Online]. Available: http://www.realnetworks.com/company/pressroom/pr/pr1997

[12] G. Motta, J. Storer, and B. Carpentieri, "Improving scene cut quality for real-time video decoding," *Proc. IEEE Data Compression Conf. DCC'00*, pp. 470–479, Mar. 26–30, 2000.

[13] S. Sen, J. L. Rexford, J. K. Dey, J. F. Kurose, and D. F. Towsley, "Online smoothing of variable-bit-rate streaming video," *IEEE Trans. Multimedia*, vol. 2, no. 1, pp. 37–48, Mar. 2000.

[14] A. E. Mohr, E. A. Riskin, and R. E. Ladner, Unequal loss protection: Graceful degradation over packet erasure channels through forward error correction, IEEE J. Select. Areas Commun., to be published.

[15] RealNetworks. (1998). RealNetworks announces RealSystem G2, the next generation streaming media delivery system (press release). [Online]. Available: http://www.realnetworks.com/company/pressroom/pr/pr1998

[16] A. Lippman, "Video coding for multiple target audiences," in *Proc. IS and T/SPIE Conf. Visual Communications and Image Processing*, San Jose, CA, Jan. 1999, pp. 780–784.

[17] H. Schulzrinne, A. Rao, and R. Lanphier, "Real-time streaming protocol (RTSP)," IETF, RFC 2326, Apr. 1998.

[18] H. Schulzrinne, S. Casper, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," IETF, Request for Comments 1889, Jan. 1996.

[19] P. Hoschka, "The Application/SMIL media type," (draft-hoschka-smil-media-type-04.txt), Dec. 1999.

[20] M. R. Macedonia and D. P. Brutzman, "MBone provides audio and video across the Internet," *IEEE Comput.*, pp. 30–36, Apr. 1994.

[21] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton Univ. Press, 1962.

[22] B. Bolobas, *Graph Theory, An Introductory Course*. New York: Springer-Verlag, 1979.

[23] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[24] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[25] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inform. Theory*, vol. 46, pp. 1204–1216, 2000.

[26] T. Berger, *The Information Theory Approach to Communications*, G. Lingo, Ed. New York: Springer-Verlag, 1977. Multiterminal source coding.

[27] D. Slepian and J. K. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 471–480, 1973.

[28] A. Wyner and J. Ziv, "The rate distortion function for the source coding with side information at the receiver," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 1–11, 1976.

[29] H. S. Witsenhausen, "On source networks with minimal breakdown degradation," *Bell Syst. Tech. J.*, vol. 59, no. 6, pp. 1083–1087, July–Aug. 1980.

[30] J. K. Wolf, A. D. Wyner, and J. Ziv, "Source coding for multiple descriptions," *Bell Syst. Tech. J.*, vol. 59, no. 8, pp. 1417–1426, Oct. 1980.

[31] L. Ozarow, "On a source coding problem with two channels and three receivers," *Bell Syst. Tech. J.*, vol. 59, no. 10, pp. 1909–1921, Dec. 1980.

[32] S. McCanne, V. Jacobsen, and M. Vetterli, "Receiver-driven layered multicast," in Proc. ACM SIGCOMM, Stanford, CA, Aug. 1996.

[33] D. Sisalem and H. Schulzrinne, "The loss-delay adaptation algorithm: A TCP-friendly adaptation scheme," in *Proc. Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Cambridge, U.K., July 1998.

[34] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in Proc. ACM SIGCOMM, Stockholm, Sweden, Aug. 2000.

[35] P. A. Chou, A. E. Mohr, S. Mehrotra, and A. Wang, "FEC and pseudo-ARQ for receiver-driven layered multicast of audio and video," *Proc. IEEE Data Compression Conf. (DCC)*, Mar. 27–30, 2000.

[36] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic," *IEEE Trans. Networking*, vol. 2, pp. 1–15, 1994.

[37] B. Girod, N. Farber, and U. Horn, "Scalable codec architectures for Internet video on demand," in *Proc. 1997 Asilomar Conf. Signals and Systems*, Pacific Grove, CA, Nov. 1997.

[38] M. Flierl, T. Wiegand, and B. Girod, "A locally optimal design algorithm for block-based multi-hypothesis motion-compensated prediction," in *Proc. Data Compression Conf. DCC'98*, Snowbird, UT, Apr. 1998, pp. 239–248.

[39] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. New York: Academic, Aug. 1990.

[40] V. Koshelev, "Estimation of mean error for a discrete succesive approximation scheme," *Probl. Inform. Transm.*, vol. 17, pp. 20–33, July–Sept. 1981.

[41] W. H. R. Equitz and T. M. Crover, "Succesive refinement of information," *IEEE Trans. Inform. Theory*, vol. 37, pp. 269–274, Mar. 1991.

[42] B. Rimoldi, "Succesive refinement of information: Characterization of the achievable rates," *IEEE Trans. Inform. Theory*, vol. 40, pp. 253–259, Jan. 1994.

[43] M. K. Uz, M. Vetterli, and D. J. LeGall, "Interpolative multiresolution coding of advanced television with compatible subchannels," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 1, pp. 86–99, 1991.

[44] U. Horn and B. Girod, "Scalable video transmission for the Internet," *Computer Networks and ISDN Syst.*, vol. 29, pp. 1833–1842, 1997.

[45] *Video Coding for Low Bit Rate Communication*, ITU-T Recommendation H.263, Version 2, Feb. 1998.

[46] F. C. M. Martins and T. Gardos, "Efficient receiver-driven layered multicast using H.263+ SNR scalability," *Proc. IEEE Int. Conf. Image Processing (ICIP)*, pp. 32–35, Oct. 4–7, 1998.

[47] Y. Wang, M. T. Orchard, and A. R. Reibman, "Multiple description image coding for noisy channels by pairing transform coefficients," in *Proc. Workshop on Multimedia Signal Processing*, Princeton, NJ, June 1997, pp. 419–424.

[48] V. K. Goyal, J. Kovacevic, and M. Vetterli, "Multiple description transform coding: Robustness to erasures using tight frame expansions," in *Proc. Int. Symp. Information Theory*, Cambridge, MA, Aug. 1998, p. 408.

[49] *Video Coding for Low Bit Rate Communication*, ITU-T SG16/Q.15 H.26L Project, Feb. 2000.
[50] G. S. Greenbaum, "Remarks on the H.26L Project: Streaming Video Requirements for Next Generation Video Compression Standards," ITU-T SG16 (Q15), Doc. Q15-G-11, Monterey, CA, Feb. 16–19, 1999.
[51] *Call for Evidence Justifying the Testing of Video Coding Technology*, ISO/IEC JTC1/SC29/WG11 N3318, Mar. 2000.
[52] *A Response for Evidence Justifying the Testing of Video Coding Technology*, ISO/IEC JTC1/SC29/WG11 M6171 (S. Greenbaum, K. 0. Lillevold: RealNetworks, Inc.; J. McVeigh, R. R. Rao: Intel, Corp.), July 2000.
[53] *Call for Proposals for New Tools to Further Improve Video Coding Efficiency*, ISO/IEC JTC1/SC29/WG11 N3671, Oct. 2000.
[54] RealNetworks.. RealSystem G2 SDK and documentation, Seattle, WA. [Online]. Available: http://www.realnetworks.com/devzone/downlds/index.html
[55] K. O. Lillevold, "Improved Direct Mode for B Pictures in TML,", Portland, Oregon, Aug. 22–25, 2000. ITU-T SG16 (Q15), Doc. Q15-K-44.

Gregory J. Conklin (M'98) received the B.S. degree (Hons.) and the M.S. degree from Cornell University, Ithaca, NY, in 1995 and 1999, respectively, both in electrical engineering.

In 1998, he joined RealNetworks, Inc., Seattle, WA, where he is currently a Principal Engineer. His current research interests include low-bit rate video coding, scalable video coding and transmission, video coding for packet-based networks, and image and video postprocessing.

Mr. Conklin is a member of Eta Kappa Nu and Tau Beta Pi.

Gary S. Greenbaum (M'98) received the B.S degrees in mathematics and physics from Pennsylvania State University, State College, PA, in 1988, and the M.S. and Ph.D degrees in high-energy particle physics from the University of California, San Diego, CA, in 1989 and 1995, respectively, while participating as a visiting scholar at Stanford University, Stanford, CA.

In 1996, he joined RealNetworks, Inc, Seattle, WA, as the first Video Codec Engineer, and currently heads the Audio and Video Technologies Group.

Karl O. Lillevold received the M.S. degree from the Norwegian Institute of Technology, Trondheim, Norway, in 1992, in electrical engineering.

In 1993, he joined Telenor Research, Norway, where he developed the public domain H.263 software simulation test model, TMN. In 1996, he joined Intel Corporation, Portland, OR, where he continued work on video-coding research and development. In 1999, he joined RealNetworks, Inc., Seattle, WA, where he is currently a Principal Codec Engineer. His current research interests include low-bit rate video coding, video coding and transmission for packet-based networks, and video pre- and postprocessing.

Alan F. Lippman (M'98) received the Ph.D. degree in applied mathematics from Brown University, Providence, RI, in 1986.

Between 1986 and 1995, he held a number of academic and industry positions, and in 1995, joined RealNetworks, Inc., Seattle, WA, as Chief Engineer. He is the author of more than 20 academic papers, including works on speech recognition and image processing. which

Dr. Lippman was a recipient of the IEEE Signal Processing Society's Best Paper Award in 1995 for his co-authored paper "Non-Parametric Multivariate Density Estimation: A Comparative Study." He is currently a member of the IEEE Signal Processing Society Technical Committee on Multimedia Signal Processing.

Yuriy A. Reznik (M'97) received the engineer degree (Hons.) in electronics engineering in 1992, and the candidate degree in computer science in 1995, both from Kiev Polytechnic Institute, Kiev, Ukraine.

From 1989 to 1993, he was with faculty of Radio Engineering at Kiev Polytechnic Institute, first as a Research Assistant and, since 1991, as an Engineer. In 1993, he become a member of the Research Staff at the Institute of Mathematical Machines and Systems, National Academy of Sciences of Ukraine. Since 1995, he also has been serving as a Consultant to various U.S. and international companies, and in 1998, he joined RealNetworks, Inc., Seattle, WA, where he is currently a member of technical staff. His research interests include theory of computing, combinatorics, analysis of algorithms, information theory, digital signal processing, and their applications.