1

HD Video Remote Collaboration Application

Beomjoo Seo, Xiaomin Liu, and Roger Zimmermann
School of Computing, National University of Singapore, Singapore

1.1 Introduction

High-quality, interactive collaboration tools increasingly allow remote participants to engage in problem solving scenarios resulting in quicker and improved decision-making processes. With high-resolution displays becoming increasingly common and significant network bandwidth being available, high-quality video streaming has become feasible and innovative applications are possible. Initial work on systems to support high-definition (HD) quality streaming focused on off-line content. Such video-on-demand systems for IPTV (Internet protocol television) applications use elaborate buffering techniques that provide high robustness with commodity IP networks, but introduce long latencies. Recent work has focused on interactive, real-time applications that utilize HD video. A number of technical challenges have to be addressed to make such systems a reality. Ideally, a system would achieve low end-to-end latency, low transmission bandwidth requirements, and high visual quality all at the same time. However, since the pixel stream from an HD camera can reach a raw data rate of 1.4 Gbps, simultaneously achieving low latency while maintaining a low transmission bandwidth – through extensive compression – are conflicting and challenging requirements.

This chapter describes the design, architectural approach, and technical details of the *remote collaboration system* (RCS) prototype developed under the auspices of the Pratt & Whitney, UTC Institute for Collaborative Engineering (PWICE), at the University of Southern California (USC).

The focus of the RCS project was on the acquisition, transmission, and rendering of high-resolution media such as HD quality video for the purpose of building multisite, collaborative applications. The goal of the system is to facilitate and speed up collaborative maintenance procedures between an airline's technical help desk, its personnel

The Handbook of MPEG Applications: Standards in Practice Edited by Marios C. Angelides and Harry Agius

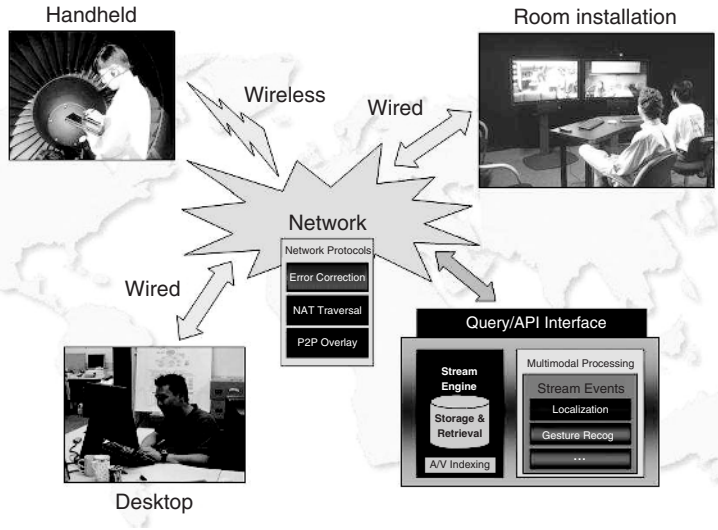

Figure 1.1 RCS collaborative systems architecture.

working on the tarmac on an aircraft engine, and the engine manufacturer. RCS consists of multiple components to achieve its overall functionality and objectives through the following means:

1. Use high fidelity digital audio and high-definition video (HDV) technology (based on MPEG-2 or MPEG-4/AVC compressed video) to deliver a high-presence experience and allow several people in different physical locations to collaborate in a natural way to, for example, discuss a customer request.
2. Provide multipoint connectivity that allows participants to interact with each other from three or more physically distinct locations.
3. Design and investigate acquisition and rendering components in support of the above application to optimize bandwidth usage and provide high-quality service over the existing and future networking infrastructures.

Figure 1.1 illustrates the overall architecture of RCS with different possible end-stations: room installations, desktop and mobile computers.

1.2 Design and Architecture

HD displays have become common in recent years and large network bandwidth is available in many places. As a result, high-quality interactive video streaming has become feasible as an innovative application. One of the challenges is the massive amount of data required for transmitting such streams, and hence simultaneously achieving low latency and keeping the bandwidth low are often contradictory. The RCS project has focused on the design of a system that enables HD quality video and multiple channels of audio to

be streamed across an IP based network with commodity equipment. This has been made possible due to the technological advancements in capturing and encoding HD streams with modern, high-quality codecs such as MPEG-4/AVC and MPEG-2. In addition to wired network environments, RCS extends HD live streaming to the wireless networks, where bandwidth is limited and the packet loss rate can be very high.

The system components for one-way streaming from a source (capture device) to a sink (media player) can be divided into four stages: media acquisition, media transmission, media reception, and media rendering. The media acquisition component specifies how to acquire media data from a capture device such as a camera. Media acquisition generally includes a video compression module (though there are systems that use uncompressed video), which reduces the massive amount of raw data into a more manageable quantity. After the acquisition, the media data is split into a number of small data packets that will then be efficiently transmitted to a receiver node over a network (media transmission). Once a data packet is received, it will be reassembled into the original media data stream (media reception). The reconstructed data is then decompressed and played back (media rendering). The client and server streaming architecture divides the above stages naturally into two parts: a server that performs media acquisition and transmission and a client that executes media reception and rendering.

A more general live streaming architecture that allows multipoint communications may be described as an extension of the one-way streaming architecture. Two-way live streaming between two nodes establishes two separate one-way streaming paths between the two entities. To connect more than two sites together, a number of different network topologies may be used. For example, the *full-mesh* topology for multiway live streaming applies two-way live streaming paths among each pair of nodes. Although full-mesh connectivity results in low end-to-end latencies, it is often not suitable for larger installations and systems where the bandwidth between different sites is heterogeneous.

For RCS, we present several design alternatives and we describe the choices made in the creation of a multiway live streaming application. Below are introductory outlines of the different components of RCS which will subsequently be described in turn.

Acquisition. In RCS, MPEG-2-compressed HD camera streams are acquired via a FireWire interface from HDV consumer cameras, which feature a built-in codec module. MPEG-4/AVC streams are obtained from cameras via an external Hauppauge HD-PVR (high-definition personal video recorder) encoder that provides its output through a USB connection. With MPEG-2, any camera that conforms to the HDV standard[1] can be used as a video input device. We have tested multiple models from JVC, Sony, and Canon. As a benefit, cameras can easily be upgraded whenever better models become available. MPEG-2 camera streams are acquired at a data rate of 20–25 Mbps, whereas MPEG-4/AVC streams require a bandwidth of 6.5–13.5 Mbps.

Multipoint Communication. The system is designed to accommodate the setup of many-to-many scenarios via a convenient configuration file. A graphical user interface is available to more easily define and manipulate the configuration file. Because the software is modular, it can naturally take advantage of multiple processors and multiple cores. Furthermore, the software runs on standard Windows PCs and can therefore take advantage of the latest (and fastest) computers.

[1] http://www.hdv-info.org/

Compressed Domain Transcoding. This functionality is achieved for our RCS implementation on Microsoft Windows via a commercial DirectShow filter module. It allows for an optional and custom reduction of the bandwidth for each acquired stream. This is especially useful when streaming across low bandwidth and wireless links.

Rendering. MPEG-2 and MPEG-4/AVC decoding is performed via modules that take advantage of motion compensation and iDCT (inverse discreet cosine transform) hardware acceleration operation in modern graphics cards. The number of streams that can be rendered concurrently is only limited by the CPU processing power (and in practice by the size of the screens attached to the computer). We have demonstrated three-way HD communication on dual-core machines.

1.2.1 Media Processing Mechanism

We implemented our RCS package in two different operating system environments, namely, Linux and Windows. Under Linux, every task is implemented as a process and data delivery between two processes uses a pipe, one of the typical interprocess communication (IPC) methods, that transmit the data via standard input and output. In the Linux environment, the pipe mechanism is integrated with the virtual memory management, and so it provides effective input/output (I/O) performance. Figure 1.2a illustrates how a prototypical pipe-based media processing chain handles the received media samples. A packet receiver process receives RTP (real-time transport protocol)-similar packets from a network, reconstructs the original transport stream (TS) by stripping the packet headers, and delivers them to an unnamed standard output pipe. A multiplexer, embedded in a video decoder process, waits on the unnamed pipe, parses incoming transport packets, consumes video elementary streams (ES) internally, and forwards audio ES to its unnamed pipe.

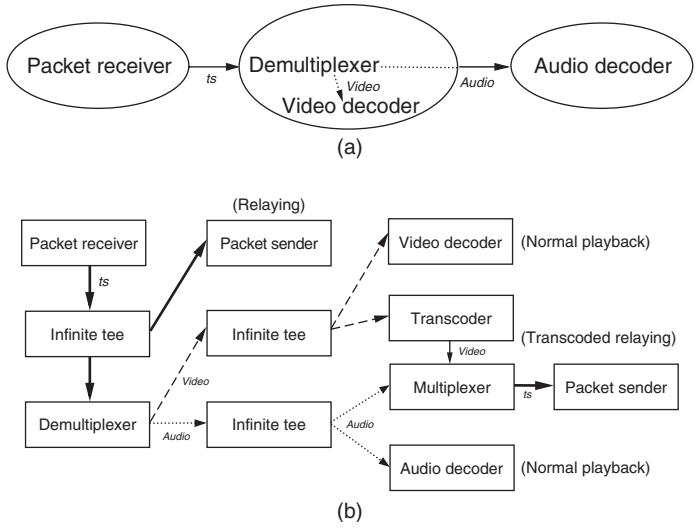

Figure 1.2 Example of delivery paths of received packets, using different media processing mechanisms: (a) pipe-based chaining and (b) DirectShow-based filter chaining.

Lastly, an audio decoder process at the end of the process chain consumes the incoming streams. Alternatively, the demultiplexer may be separated from the video decoder by delivering the video streams to a named pipe, on which the decoder is waiting.

On the Windows platform, our investigative experiments showed that a pipe-based interprocess data delivery mechanism would be very I/O-intensive, causing significant video glitches. As an alternative design to the pipe mechanism, we chose a DirectShow filter pipeline. DirectShow – previously known as ActiveMovie and a part of the DirectX software development kit (SDK) – is a component object model (COM)-based streaming framework for the Microsoft Windows platform. It allows application developers not only to rapidly prototype the control of audio/video data flows through high-level interfaces (APIs, application programming interfaces) but also to customize low-level media processing components (filters).

The DirectShow filters are COM objects that have a custom behavior implemented along filter-specific standard interfaces and then communicate with other filters. User-mode applications are built by connecting such filters. The collection of connected filters is called a *filter graph*, which is managed by a high-level object called the *filter graph manager* (*FGM*). Media data is moved from the source filter to the sink filter (or renderer filter) one by one along the connections defined in the filter graph under the orchestration of the FGM. An application invokes control methods (Play, Pause, Stop, Run, etc.) on an FGM and it may in fact use multiple FGMs. Figure 1.2b depicts one reception filter graph among various filter graphs implemented in our applications. It illustrates how media samples that are delivered from the network are processed along multiple branching paths – that is, a relaying branch, a transcoded relaying branch, and normal playback. The infinite tee in the figure is an SDK provided standard filter, enabling source samples to be transmitted to multiple filters simultaneously.

Unlike the pipe mechanism under Windows, a DirectShow filter chain has several advantages. First, communication between filters is performed in the same address space, meaning that all the filters (which are a set of methods and processing routines) communicate through simple function calls. The data delivery is via passed pointers to data buffers (i.e., a zero-copy mechanism). Compared to IPC, this is much more efficient in terms of I/O overhead. Second, many codecs are available as DirectShow filters, which enables faster prototyping and deployments. During the implementation, however, we observed several problems with the DirectShow filter chaining mechanism. First, the developer has no control over the existing filters other than the methods provided by the vendors, thus leaving little room for any further software optimizations to reduce the acquisition and playback latency. Second, as a rather minor issue, some filter components can cause synchronization problems. We elaborate on this in Section 1.6.1.

1.3 HD Video Acquisition

For HD video acquisition, we relied on solutions that included hardware-implemented MPEG compressors. Such solutions generally generate high-quality output video streams. While hardware-based MPEG encoders that are able to handle HD resolutions used to cost tens of thousands of dollars in the past, they are now affordable due to the proliferation of mass-market consumer products. If video data is desired in the MPEG-2 format, there exist many consumer cameras that can capture and stream HD video in real

time. Specifically, the HDV standard commonly implemented in consumer camcorders includes real-time MPEG-2 encoded output via a FireWire (IEEE 1394) interface. Our system can acquire digital video from several types of camera models, which transmit MPEG-2 TS via FireWire interface in HDV format. The HDV compressed data rate is approximately 20–25 Mbps and a large number of manufacturers are supporting this consumer format. Our earliest experiments used a JVC JY-HD10U camera that produces 720p video (1280 × 720 pixels); however, at only 30 frames per second, not the usual 60. More recently, we have used Sony and Canon cameras that implement the 1080i HD standard.

In contrast, the more recent AVCHD (advanced video coding high definition) standard (which utilizes the MPEG-4/AVC codec) that is now common with HD consumer camcorders does not support a FireWire interface. Therefore, these new cameras cannot stream compressed HD video in real time. To overcome this obstacle, we used the stand-alone Hauppauge HD-PVR model 1212 hardware compressor, which can acquire HD uncompressed component signals (YCrCb) and encode them into an MPEG-4/AVC stream. The HD-PVR is officially supported on the Windows platform; however, a Linux driver also exists. Compressed data is streamed from the HD-PVR via a USB connection. Data rates are software selectable between 1 and 13.5 Mbps. A reasonable quality output is produced at 4 Mbps and above, while good quality output requires 6.5–13.5 Mbps. Figure 1.3 illustrates our prototype setup with the HD-PVR.

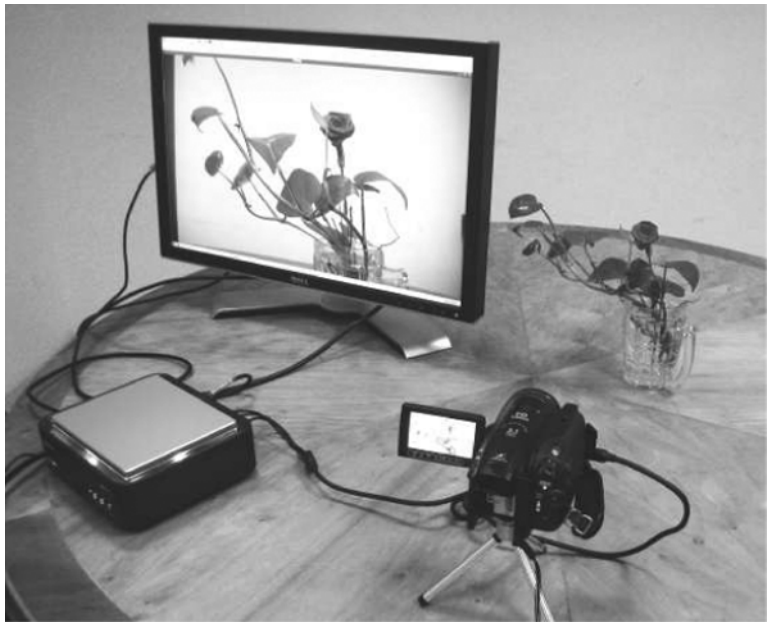

Figure 1.3 Prototype setup that includes a Canon VIXIA HV30 high-definition camcorder and a Hauppauge HD-PVR MPEG-4/AVC encoder.

1.3.1 MPEG-4/AVC HD System Chain

For HD conferencing an end-to-end chain has to be established, including both the acquisition and rendering facilities. At each end, a combination of suitable hardware and software components must be deployed. Since the objective is to achieve good interactivity, the delay across the complete chain is of crucial importance. Furthermore, video and audio quality must also be taken into consideration. Figure 1.3 illustrates our system setup when utilizing MPEG-4/AVC as the video encoding standard. We will describe each component in more detail.

The end-to-end system chain consists of the following components:

- **HD Video Camcorder**. The acquisition device used to capture the real-time video and audio streams. We utilize the uncompressed component signals that are produced with negligible latency.
- **HD MPEG-4/AVC Encoder**. The Hauppauge HD-PVR is a USB device that encodes the component video and audio outputs of the HD video camcorder. It utilizes the colorspace of YUV 420p at a resolution of 1920 × 1080 pixels and encodes the components inputs in real time using the H.264/MPEG-4 (part 10) video and AAC (advanced audio coding) audio codecs. The audio and video streams are then multiplexed into a slightly modified MPEG-2 TS container format. The bitrate is user selectable from 1 to 13.5 Mbps.
- **Receiver Demultiplexing**. A small library called MPSYS, which includes functions for processing MPEG-2 TS, is used. A tool called `ts` allows the extraction of ES from the modified MPEG-2 multiplexed stream.
- **Decoding and Rendering**. Tools based on the ffplay library are utilized to decode the streams, render the audio and video data, and play back the output to the users.

1.3.1.1 End-to-End Delay

Video conferencing is very time sensitive and a designer must make many optimization choices. For example, a different target bitrate of the encoder can affect the processing and transmission latencies. End-to-end delays with our implementation at different encoding rates are presented in Figure 1.4.

The results show that with our specific setup at a bitrate of 6.5 Mbps, the latency is lowest. At the same time, the video quality is very good. When the bitrate is below 4 Mbps, the latency is somewhat higher and the video quality is not as good. There are many blocking artifacts. When the bitrate is above 6.5 Mbps, the latency increases while the video quality does not improve very much. Figure 1.5 illustrates the visual quality of a frame when the video is streamed at different bitrates.

Encoding streams with the MPEG-4/AVC codec has several advantages. It offers the potential for a higher compression ratio and much flexibility for compressing, transmitting, and storing video. On the other hand, it demands greater computational resources since MPEG-4/AVC is more sophisticated than earlier compression methods (Figure 1.6).

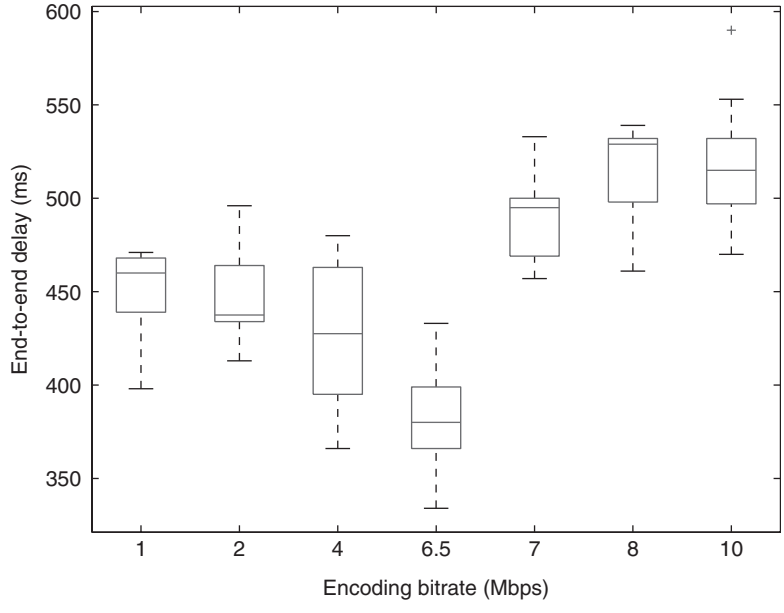

Figure 1.4 End-to-end delay distribution for different encoding bitrates with the hardware and software setup outlined in this chapter. Ten measurements were taken for each bitrate value.

1.4 Network and Topology Considerations

The streams that have been captured from the HD cameras need to be sent via traditional IP networks to one or more receivers. Audio can be transmitted either by connecting microphones to the cameras and multiplexing the data with the same stream as the video or transmitting it as a separate stream. The RCS transmission subsystem uses the RTP on top of the universal datagram protocol (UDP). Since IP networks were not originally designed for isochronous data traffic, packets may sometimes be lost between the sender and the receiver. RCS uses a single-retransmission algorithm (Papadopoulos and Parulkar 1996; Zimmermann *et al*. 2003) to recover lost packets. Buffering in the system is kept to a minimum to maintain a low latency.

To meet flexible requirements, we designed RCS' software architecture to be aware of the underlying network topology. We further reached the design decision that real-time transcoding should be integrated with the architecture to support lower bandwidth links. This requirement becomes especially critical when a system is scaled up to more than a few end user sites. Quite often some of the links may not be able to sustain the high bandwidth required for HD transmissions.

In addition to network bandwidth challenges, we also realized that the rendering quality of the video displayed on today's high-quality LCD and plasma screens suffers when the source camera produces interlaced video. The artifacts were especially noticeable with any fast moving motions. We describe how we addressed this issue in a later section.

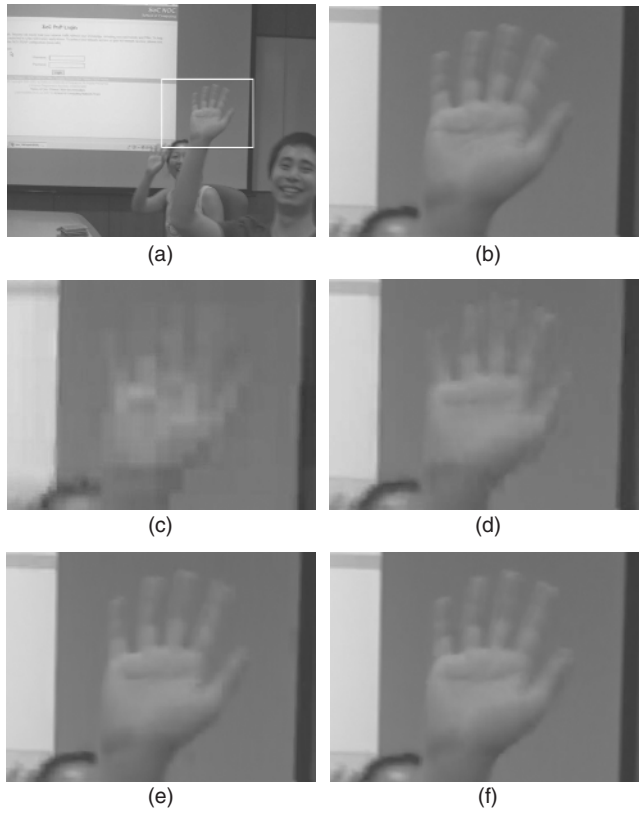

Figure 1.5 Comparison of picture quality at various encoding bitrates: (a) original image; (b) details from the original image; (c) encoded @ 2 Mbps; (d) encoded @ 4 Mbps; (e) encoded @ 6.5 Mbps; (f) encoded @ 10 Mbps.

Figure 1.6 Single end-to-end delay measurement of an MPEG-4/AVC video stream from an HD-PVR encoder at a rate of 6.5 Mbps. The delay is $(887 - 525 = 362)$ ms. The delay is measured by taking snapshot images of both the original display (left) and the transmitted video (right) of a running clock.

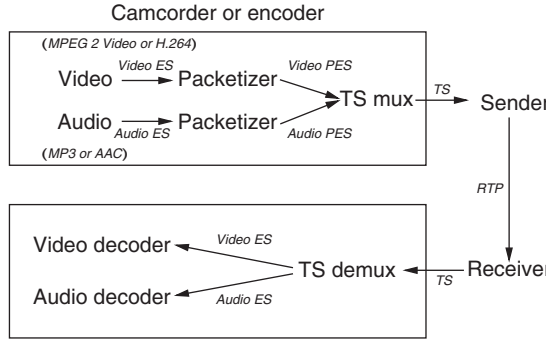

Figure 1.7 Captured media samples (MPEG-2 TS format) are packetized in the RTP format and reconstructed as a sequence of transport stream packets.

1.4.1 Packetization and Depacketization

Figure 1.7 illustrates how RTP packets are generated and delivered in the network. First, camcorders and encoders used for our application generate MPEG-TS packets, whose format is specified in the specification *MPEG-2 Part 1, Systems* (or *ISO/IEC standard 13818-1*) (ISO/IEC 1994). The acquisition process encapsulates a number of TS packets with an RTP header and transmits them over the network. At the receiver side, an RTP reception process recognizes the RTP packets and converts their payload data to a number of TS packets. Next, it separates individual streams by packet identifier (PID) values, and passes them to their corresponding decoders.

A TS packet, whose length is fixed at 188 bytes, has at least a 4-byte header. Each TS header starts with a sync byte (0×47) and contains a 13-bit PID, which enables the TS demultiplexer to efficiently extract individual packetized elementary streams (PES) separately. Every video or audio bitstream or ES cannot be converted to TS packets directly, since the TS format expects PES as input streams. Thus, every ES needs to be converted to a number of PES packets, whose maximum length is limited to 64 KB. Usually, every camcorder vendor assigns a unique PID numbers for each PES. For example, the PID of JVC video ES is 4096, Sony uses 2064, and that of the Hauppauge HD-PVR is 4113. Since identifying the PIDs of individual streams takes a longer time without *a priori* information, we hard-coded such information that is used during the TS demultiplexing in our application.

Once TS packets are acquired via a FireWire or a USB, they need to be aligned at the TS boundary to be transformed into RTP packets. To find the exact offset from the given raw samples, we first attempt to scan the first 188 bytes to locate the position of the sync byte, since the raw data should contain at least one sync byte within the first 188 bytes. Once multiple candidate offsets have been found, the detection continues to check whether their next 188th byte equals to a sync byte. These steps are repeated until only one offset remains. After the aligned offset is detected, the data acquisition software passes 188-byte aligned media samples to the rest of the delivery chain.

A single RTP packet can encapsulate multiple TS packets. To maximally utilize the network bandwidth, we used a maximum transmission unit (MTU) of 1500 bytes; therefore,

the RTP packet could encapsulate up to seven TS packets (≈1500/188). To minimize multiple PES losses from a single RTP packet loss, we separately assign a new RTP packet for each newly arriving PES packet. This condition is detected by examining the payload unit start indicator field in the TS header.

To demultiplex incoming TS packets, we use the MPSYS library[2] by embedding it with the video decoder or running it as a separate process in Linux. The small-footprint library efficiently parses MPEG-TS streams and stores them as either individual PES or ES. In the Windows environment we used an MPEG-2 demultiplexer DirectShow filter when running on the DirectShow platform, or we used the MPSYS library when running the application via the Windows pipe mechanism.

Our packetization scheme, however, has several drawbacks when handling MPEG-4/AVC videos. As specified in RFC 3984 (Wenger *et al*. 2005), the RTP payload scheme for MPEG-4/AVC recommends the use of a network abstraction layer (NAL) unit. The NAL unit that encapsulates a number of slices containing multiple macroblocks is designed for the efficient transmission of the MPEG-4/AVC video over packet networks without any further packetization; therefore, the single loss of an RTP packet does not propagate to adjacent video frames, resulting in better error-resilience. Since the NAL unit works with TS packets, the direct use of the NAL units minimizes the packet overhead. For example, our TS-encapsulated RTP scheme consumes at least the following overhead for headers: 20 (IP) + 8 (UDP) + 12 (RTP) + 7 × 4 (7 TS packet headers) + 8 (PES header, if necessary) = 76 bytes, while the NAL-aware RTP scheme requires the following headers: 20 IP + 8 UDP + 12 RTP = 40 bytes (MacAulay *et al*. 2005). Although we have not implemented this scheme due to its higher parsing complexity to reconstruct the raw MPEG-4/AVC bitstreams, it possesses many undeniable advantages over our TS-aware RTP scheme.

1.4.2 Retransmission-Based Packet Recovery

Our packet recovery algorithm has the following features:

- Reuse of the existing retransmission-based packet recovery solution.
- Reduction of the response time of a retransmission request.

There are many alternative solutions to recover lost packets. One popular solution is to use redundant data such as a forward error correction (FEC)-enabled coding scheme. This approach removes the delay associated with a retransmission request, while somewhat overutilizing the network bandwidth more than the minimally required rate and may require significant on-line processing power.

We validated our single-pass retransmission scheme in a loss-free networking environment by simulating a loss-prone network. For the simulation purposes, we included a probabilistic packet loss model and a deterministic delay model at the receiver side. The packet loss model drops incoming packets probabilistically before delivering them to a receiver application session. The receiver application detects missing packets by examining the sequence numbers in the RTP headers. If the algorithm finds any missing packets,

[2] http://www.nenie.org/misc/mpsys/

it immediately issues a retransmission request to the sender. The delay model postpones the delivery of the retransmission requests by a given amount of time. We used a two-state Markov model, widely known as the *Gilbert model*, to emulate a bursty packet loss behavior in the network.

We used a fixed 10 ms delay, since it represents the maximum round trip delay in our target network infrastructure. We varied the packet loss rates as follows: 1, 5, and 10%. For lost packets, our recovery mechanism sends at most a single-retransmission request.

Figure 1.8 reveals that our software recovered a lot of lost packets and maintained a tolerable picture quality with noticeable, but not overwhelming, glitches even in extremely loss-prone network environments such as with a packet loss rate of 10%. This applies as long as the network can utilize more than the required available bandwidth. As seen in the figure, our retransmission scheme recovered lost packets very successfully in a 1% packet loss environment. The retransmission scheme with a 5% packet loss environment also showed a similar trend as for the 1% packet loss environment. Our real-world experiments also confirmed the effectiveness of the retransmission-based recovery mechanism, even with a video conference between cross-continental multisites.

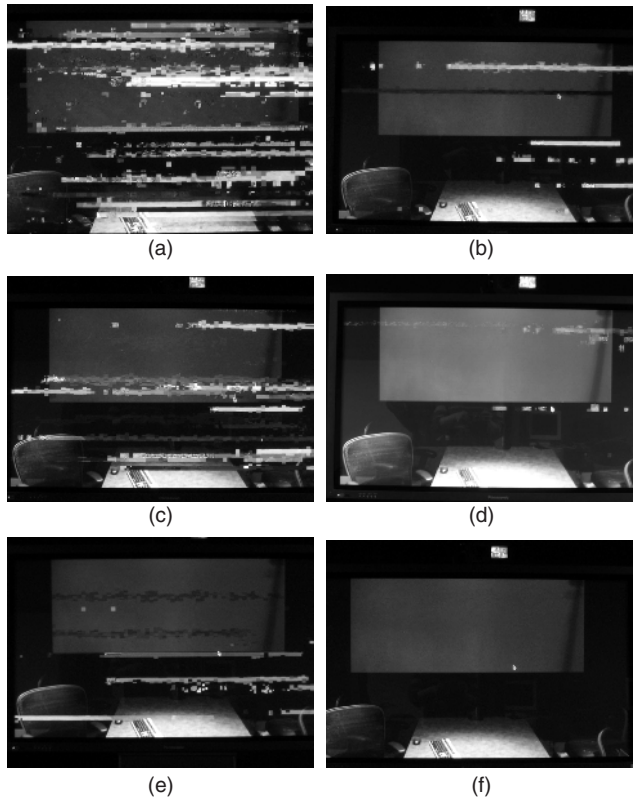

Figure 1.8 Artifacts of retransmission-based packet recovery algorithm: (a, c, and e) show the picture quality without retransmission policy with 10, 5, and 1% loss, respectively. (b, d, and f) show the picture quality with retransmission policy with 10, 5, and 1% loss, respectively.

1.4.3 Network Topology Models

The next step-up in complexity from traditional two-way conferencing is to scale the system to three sites. Unlike in audio conferencing applications where multisite sound data can be mixed together, a three-way video conferencing system requires at least two incoming video channels and one outgoing channel per participating node. This may become a limiting factor in terms of bandwidth and decoding processing resources. Our design also took into consideration real-world factors such as the characteristics of corporate networks which may be asymmetric and heterogeneous and which require optimizations with respect to the underlying available network bandwidth.

Our airline maintenance application involved three sites designated A, B, and C, where A and B are connected via a 1 Gbps dedicated link, while C is connected to other sites via a 25 Mbps public link, thus being limited to one HD stream at a time. In fact, all the video traffic to and from C had to pass through B. Moreover, participants at A are expected to experience all HD quality. This unique situation affected the design of our communication model, and we explored a number of alternative scenarios. To compare these alternatives, we present four possible scenarios, shown in Figure 1.9.

- The *full-mesh model*, illustrated in Figure 1.9a, is a simple three-way communication model, where every site has an individual path with every other site. In its deployment, however, we encountered a fundamental obstacle, as there did not exist enough network bandwidth on the path from C to B. The constraint was largely due to the design of the underlying physical topology. In fact, the path from C to A in the physical network bypasses B, doubling the network utilization of the path from C to A. Without any topology awareness, the logical path would result in intolerable image corruption, resulting from heavy network congestion at the low-bandwidth link.
- The *partial-relay model* in Figure 1.9b tackles the link stress problem of the previous model by relaying the traffic at B. The visual experience at an end user site is the same

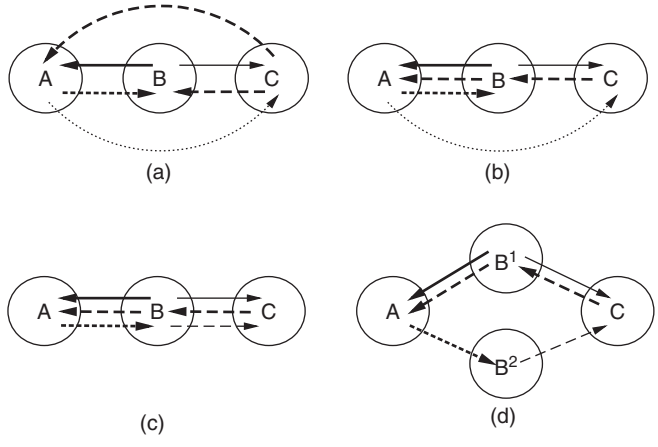

Figure 1.9 Different application-level network topologies for three-way HD conferencing: (a) full-mesh model, (b) partial-relay model, (c) full-relay model, (d) off-loading model. Bold arrows represent an HD path, while normal arrows represent a transcoded SD path.

as that of the conceptual model with a little time shifting due to the newly introduced relay delay. In the meanwhile, the traffic generated from A is still transmitted to B and C, separately. Thus, the outgoing traffic of A will be one HD plus one SD (standard-definition) quality stream.

- The *full-relay model*, shown in Figure 1.9c, additionally minimizes the link stress redundantly imposed on the path from A to C for the logical connection from A to C via relaying at B. This model eventually equals to a centralized model, since B moderates all the traffics. If the required bandwidth for SD video were, however, much smaller than that of HD video and the link capacity of A and B were so high enough to ignore small SD traffics, this optimization would not be benefited any more.

  The two relay models are still exposed to another problem. As shown in Figure 1.9c, B simultaneously captures and delivers one HD video as follows: receives two HD videos from the network, simultaneously renders them in parallel, relays one HD video, and transcodes one captured HD video to SD and delivers the reduced video. These operations are simultaneously executed on a single machine, resulting in significant CPU load. As an improvisational remedy for such a heavy load, we proposed the off-loading solution illustrated in Figure 1.9d.
- The *off-loading model* off-loads the traffic coming from A by redirecting it to B^2, which is geographically located near the B^1 site; thus, a B participant can view two HD videos transmitted from A and C on separate monitors. However, we found that the B^1 machine was still overloaded. Another suggestion to reduce the B^1 load is to move the HD streaming path to B^2.

1.4.4 Relaying

A relay node can play an important role in alleviating bandwidth bottlenecks and in reducing redundant network traffic. However, it may require full knowledge of the underlying physical network topology. In the RCS model, one node may serve as both a regular video participant and as a relay agent.

The relay program is located in the middle of the network, thus being exposed to any occurring network anomalies. To recover from any possible packet losses effectively, the relay host should maintain some small FIFO (first in first out) network buffers that can be used to resequence out-of-order packets and to request lost packets. Packets are then delivered to the destinations after the data cycles through the buffers. It is important to note that a larger buffer size introduces longer delays. Careful selection of the trade-off between the buffer size and the delay is a primary concern of the recovery mechanism. Furthermore, the relay software should be light-weight and not interfere with other programs, because multiple programs may be running on the same machine. In summary, the relay module should satisfy the following requirements:

- recover lost packets (through buffering);
- have an acceptably low relay delay;
- require minimal CPU load.

To implement the relay functionality, we modified the existing network transmission modules. At a traditional receiver, incoming packets sent from another site are temporarily

buffered and then pipelined to the video rendering engine as soon as the small local buffer is full. Our relaying mechanism augmented the existing code by writing the full buffer into the user-specified pipe area (or named pipe). The relay sender simply reads data from the pipe and sends the data continuously to the network. Our augmented relay transmission module supports both delivery policies. The relay receiver also included the retransmission-based error recovery algorithm.

However, our experiments showed that the local pipe mechanism, even though it is simple and light-weight, suffered from irregular load fluctuations, resulting in significant quality degradations. Under Linux, it seemed that the pipe mechanism was closely related with the unbalanced CPU load, which made it less useful in some environments. Such oscillations could potentially be a side effect of uneven load scheduling of two separate programs, the receiver and the sender. Thus, the relay operation would probably benefit from running as a single program.

1.4.5 Extension to Wireless Networks

There are numerous challenges when designing and implementing HD streaming over a wireless network. Some existing technologies, for example, 802.11a/g, provide for a maximum sustained bandwidth of approximately 23 Mbps. This is significantly lower than the theoretical and advertised maximum of 54 Mbps. Furthermore, the channel characteristics in wireless networks are very dynamic and variable. As such, packet losses, bandwidth fluctuations, and other adverse effects are a frequent occurrence and require a careful design of the transmission protocol and rendering algorithms. An early prototype of our RCS implementation for wireless networks is shown operational in a laboratory environment in Figure 1.10. In our real-world application, we were able to demonstrate wireless HD streaming in a large aircraft hangar with high visual quality and minimal interference. Figure 1.11 shows the multisite system during a test scenario with the wireless video transmission shown in the upper right corner.

Figure 1.10 HD transmission over a wireless, *ad hoc* link (802.11a) between two laptops in the laboratory.

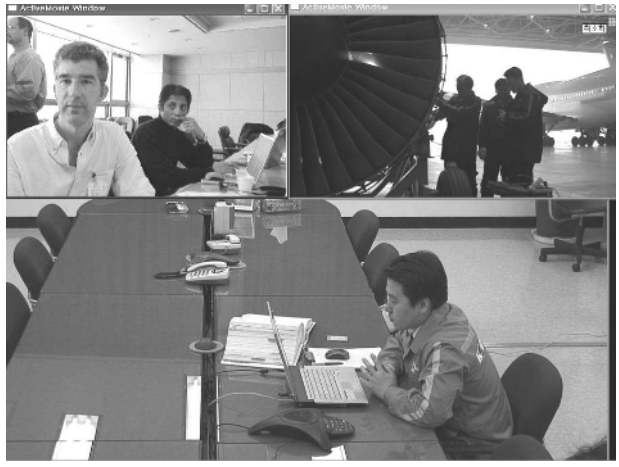

Figure 1.11 HD multiparty conference with two wired (top left and bottom) and one wireless HD transmission (from an aircraft hangar).

1.5 Real-Time Transcoding

Transcoding refers to a process of converting digital content from one encoding format to another. Owing to its broad definition, it can be interpreted in a number of different ways: conversion from a given video format to another (format conversion); lowering of the bitrate without changing the format (bitrate reduction); reduction of the image resolution to fit to a target display (image scaling); or naively performing complete decoding and re-encoding (cascaded pixel-domain transcoding). Since transcoding allows the adaptation of the video bandwidth to the different requirements of various end users, it is a vital component in the toolkit of a multiway video conference solution, and we narrow the focus of our discussion to three types of bitrate reduction architectures: cascaded pixel-domain transcoding, closed-loop transcoding, and open-loop transcoding.

The cascaded pixel-domain architecture fully decodes compressed bitstreams to reconstruct original signals and then re-encodes them to yield the desired bitstream. While achieving the best performance in terms of video quality, it presents significant computational complexity mainly due to the two iDCT and one DCT processes required. The closed-loop method is the approximation of the cascaded architecture. At the expense of accuracy, and only by using a pair of iDCT and DCT stages, it improves the transcoding complexity significantly.

The open-loop architecture modifies only DCT coefficients in the encoded bitstream by increasing the quantization step size (requantization) or by dropping high-frequency coefficients (data partitioning). In particular, the requantization method converts the encoded bitstream into the DCT domain through variable length decoding (VLD) and then applies coarse-grained quantization to the intermittent signals, which eventually results in more DCT coefficients becoming zero and variable length codes becoming shorter. Since the open-loop approach does not use any DCT/iDCT stages, it achieves minimal processing complexity, but it is exposed to a drift problem. A drift error is caused by the loss of high-frequency information, which damages the reconstruction of reference frames and

their successive frames. On the other hand, the cascaded and closed-loop architectures are free from this problem.

In our application, the transcoding component should satisfy the following criteria:

- acceptable video quality;
- acceptable transcoding latency;
- minimal use of local resources.

All three requirements are crucial for a software-driven transcoding component, and it added significant flexibility to our RCS three-way video communication. We started our experiments by customizing an existing transcoding utility, called *mencoder*, which implements the cascaded pixel-domain or the closed-loop system. It is available as one of the utilities for the open-source *MPlayer* video player software package. In the Linux environment, MPlayer is very popular for rendering a multitude of video formats, including the latest video standard such as MPEG-4/AVC. The mencoder was configured to decode incoming MPEG-2 TS packets and then to encode them into a designated video format. We tested two types of transcoded video formats: MPEG-2 program streams (PS) and MPEG-2 TS.

Our earlier experiments in transcoding were a partial success. We were able to successfully transcode MPEG-2 TS into MPEG-2 PS or newly encode an MPEG-2 TS stream. However, two problems were found. First, the transcoding delay was so high that the final end-to-end delay measured about 2 s. Secondly, the machines at one of our sites could not transcode the original HD videos into SD-quality video, due to its underpowered processor. When reproducing the SD-quality video, the transcoder continuously dropped frames, causing frequent video hiccups even without any network retransmissions. Through a series of parameter reconfigurations, we found the optimal video resolution of 300×200 pixels that did not cause any frame drops or video hiccups. Even then, the CPU load was very high, more than 50% on a single core Pentium machine. Thus, we were not able to run two transcoding instances simultaneously on a single machine. The mencoder tended to grab more CPU cycles if any idle time was detected. Such overloads resulted in highly uneven CPU utilization, sometimes causing random program terminations. Transcoding was such an expensive operation that we needed to separate it from other running programs.

Another alternative for software-based transcoding was to use a rather simple, but fast, requantization method, one of the open-loop architectures. Compared to mencoder, this approach does not fully decode and encode the streams, but quantizes pixels at the compressed level. Such a technique would perform much faster and overall be more light-weight.

We experimented with a commercialized DirectShow filter from Solveig Multimedia, called *requantizer*, for our RCS Windows implementation. It was inserted in the middle of several filter chains to convert a 20 Mbps MPEG-2 TS into 10 Mbps MPEG-2 TS in real time. Experimental results showed that the requantizer was able to reduce the bitrate by half while maintaining the same video resolution without any noticeable artifacts caused by drift error. Its CPU utilization was consistently measured to be negligible at less than 1%. It also had no negative effects on any increase in the end-to-end delay from a filter source to a sink. Since the requantization-based transcoding met our criteria, we finally chose the open-loop architecture as our transcoding scheme for the implementation.

One drawback of the requantization scheme was that its bitrate reduction was very limited. Although it met our application needs to some degree, it failed to achieve a bitrate reduction of more than a factor of 2. When reducing the bitrate by more than 50%, we found that the result was a serious deterioration of the picture quality.

1.6 HD Video Rendering

Once a media stream is transmitted over a network, the rendering component requires an MPEG-2 or MPEG-4 HD decoder. While we use specialized hardware assistance for encoding, we considered various hardware and software options for decoding of streams with the goal of achieving the best quality video with minimal latency. With RCS, we tested the following three solutions:

1. *Hardware-Based (MPEG-2)*. When improved quality and picture stability are of paramount importance, we experimented with the CineCast HD decoding board from Vela Research. An interesting technical aspect of this card is that it communicates with the host computer through the SCSI (small computer systems interface) protocol. We have written our own Linux device driver as an extension of the generic Linux SCSI support to communicate with this unit. An advantage of this solution is that it provides a digital HD-SDI (high-definition serial digital interface; uncompressed) output for very high picture quality and a genlock input for external synchronization. Other hardware-based decoder cards also exist.
2. *Software-Based (MPEG-2)*. Utilizing standard PC hardware, we have used the libmpeg2 library – a highly optimized rendering code that provides hardware-assisted MPEG-2 decoding on current-generation graphics adapters. Through the XvMC extensions of Linux X11 graphical user interface, libmpeg2 utilizes the motion compensation and iDCT hardware capabilities on modern graphics GPUs (graphics processing units; e.g., nVidia). This is a very cost-effective solution. In our earliest experiments, we used a graphics card based on an nVidia FX 5200 GPU, which provides low computational capabilities compared to current-generation GPUs. Even with the FX 5200 GPU, our software setup achieved approximately 70 fps @ 1280×720 with a 3 GHz Pentium 4.
3. *Software-Based (MPEG-4/AVC)*. The *ffplay* player is used as the main playback software to decode and render the streams. It is a portable media player based on the *ffmpeg* and the SDL libraries. The player supports many options for users to choose such as to select which kind of video and audio format will be played. For our experiments, the ES extracted by the `ts` tool are input into ffplay while we also specify the input video format using the options.

For MPEG-4/AVC rendering, our prototype system configuration had the following specifications:

- Quad core CPU: Intel(R) Core(TM)2 Extreme CPU X9650 @ 3.00 GHz.
- Video card: nVidia Corporation Quadro FX 1700.

- Sound card: Intel Corporation 82801I (ICH9 Family) HD Audio Controller.
- Operating system: Ubuntu 9.10 with Linux kernel version 2.6.31-17 SMP.
- Main memory: 3.25 GB.

To quantify the image quality of different encoding rates of a Hauppauge HD-PVR box, we use a simple but still widely used performance metric, peak signal-to-noise ratio (PSNR). Especially, the PSNR of the luminance component (Y) for a given image is known to be more suitable for the evaluation of a color image than the normal PSNR. In our experiment, we prerecord a reference video through a Sony HDV camcorder, replay it for the HD-PVR box to re-encode analogous video output with five different encoding rates (1, 2, 4, 8, and 10 Mbps), and obtain the PSNR values of every encoded image from the reference picture. The encoded video resolution was equally configured to that of the reference video – that is, 1920×1080 i.

Figure 1.12 depicts the evaluation results of 300 video frames (corresponding to 10 s) for all encoded videos. As shown in the figure, the encoding rate of 4 Mbps could reproduce a very comparable image quality to those of high bitrate videos. Although not shown in this figure, the encoding rate more than 5 Mbps tends to show better treatments on dynamically changing scene. Additionally, we also observe that higher bitrate more than 8 Mbps does no longer improve the picture quality significantly.

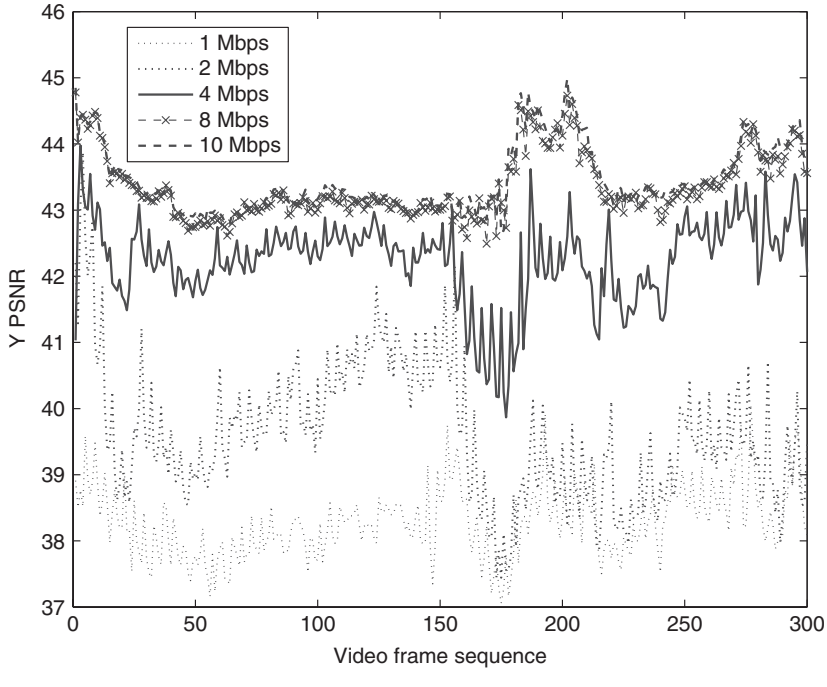

Figure 1.12 The luma PSNR values of different encoding rates by a HD-PVR box are plotted over 300 video frames.

1.6.1 Rendering Multiple Simultaneous HD Video Streams on a Single Machine

In RCS, we performed extensive experiments with a three-way connection topology. Every site was able to watch at least two other participants. Hence, every machine was equipped with the necessary resources to render two HD streams. In our early measurements, one HD decoding process occupied approximately 20% of the CPU load based on our hardware platform. Thus, we naturally expected that every machine could render two simultaneous HD videos locally. Rendering two SD streams was also expected to present a lighter load compared with two HD streams because of the comparatively lower rendering complexity.

We had no problem to display two HD video streams on a single machine. Originally, we were uncertain whether the machines at two sites could support two HD rendering processes simultaneously because of their rather low-end single core CPU architectures. In a slower single core CPU model in our lab, the two HD displays occasionally showed unbalanced CPU loads during tests. We were able to run two HD video renderers and two audio renderers simultaneously on some machines. However, the weaker computers could not run two audio players concurrently while running two video player instances.

In summary, we confirmed that two HD renderings including network transmission modules worked fine at sufficiently powerful sites. However, CPU utilization was a little bit higher than we expected; thus, it was unclear whether the video transcoding utility would be runnable in parallel on a single machine.

1.6.1.1 Display Mode

In order to provide flexibility at each of the end user sites, we implemented a number of display mode presets that a user could easily access in our software. The display mode in RCS specifies how to overlay multiple videos on a single window. The single mode shows only one video at a time (Figure 1.13). The grid mode divides the video screen into multiple equisized rectangular cells and shows one video per grid cell (Figure 1.14). Since we did not plan to support more than eight incoming streams, the maximum number of grid cells was fixed at eight. The last mode, picture-in-picture (PIP) mode, shows two

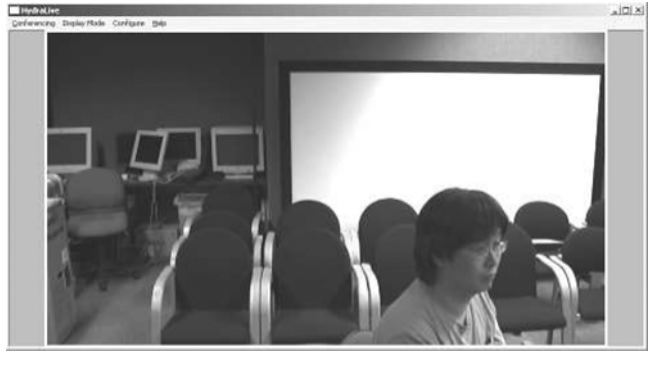

Figure 1.13 Single display mode.

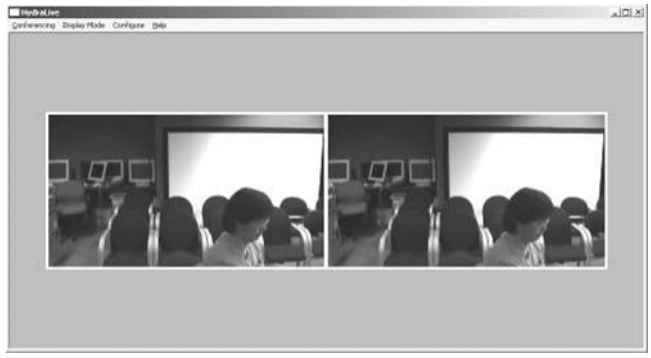

Figure 1.14 Grid display mode (side-by-side).

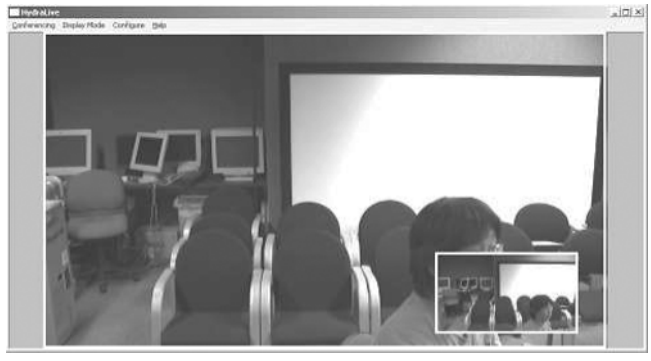

Figure 1.15 Picture-in-picture display mode.

video streams simultaneously: one main video stream in the background and the other small subvideo screen at the right bottom corner in the foreground (Figure 1.15).

We also provided a navigational method that quickly switches from one video stream to another by pressing the arrow keys. Let us assume an example where there is a need to display three video streams (1, 2, and 3). In single mode, the display order upon any right arrow key stroke is $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$. The left key reverses the display order to $1 \rightarrow 3 \rightarrow 2 \rightarrow 1$. In grid mode, the ordering for the right arrow key is $1,2,3 \rightarrow 3,1,2 \rightarrow 2,3,1 \rightarrow 1,2,3$ and for the left key $1,2,3 \rightarrow 2,3,1 \rightarrow 3,1,2 \rightarrow 1,2,3$. In PIP mode, the order for a right arrow key press is $1,2 \rightarrow 3,1 \rightarrow 2,3 \rightarrow 1,2$ and for the left key $1,2 \rightarrow 2,3 \rightarrow 3,1 \rightarrow 1,2$. The up and down arrow keys are assigned to change the display modes. The cycling order of the up key is single $\rightarrow$ grid $\rightarrow$ PIP $\rightarrow$ single. The down key reverses the order: single $\rightarrow$ PIP $\rightarrow$ grid $\rightarrow$ single.

One crucial issue in this display mode is related to the limitation of the DirectShow filter chaining mechanism: synchronized rendering. When all the videos are connected to a single video mixing render (VMR) filter for a unified display on a single video plane, the starting times of individual video renderings are synchronized with the longest start-up latency among all the individual video renderers. This is primarily due to a

VMR implementation policy, where the video mixing operation starts only after all media samples of its input filters are available. On the other hand, as exemplified in Figure 1.12, when video rendering chains are separated and running as different processes, such time synchronization problems do not exist.

1.6.2 Deinterlacing

We tested a number of Sony HD camcorders whose video format is interlaced video output (1080i). As long as the interlaced videos are displayed on an interlaced television and progressive videos are shown on a monitor-like screen, different video modes will not be a problem. However, many new big-screen displays are now progressive in nature and thus they might produce interlacing artifacts during display. Although our test plasma television technically supported interlaced rendering, it turned out to be difficult to enable the computer graphics cards to output interlaced signals, and the autodetection mechanism usually defaulted to a progressive mode. This practical problem may be solvable with further investigations into the compatibility between video drivers and display capabilities. However, even if the interlaced mode can be set successfully, we would be somewhat hesitant to use it because, from our experience, the interlaced display of text output is very unsatisfactory.

In response, we decided to add a deinterlacing routine to the video rendering software. It eliminated the interlacing artifacts produced by alternating odd and even fields of frames. Again, such a module should be light-weight as it will postprocess signals during the last stage of the video rendering process. If its processing load is too heavy, it may result in a failure to display two simultaneous HD renderings.

We implemented the linear blending deinterlacing algorithm at the very end of video rendering pipeline, right before the pixels were displayed on the screen. The approach is to interpolate consecutive even and odd lines. Specifically, the algorithm computes the average values of the pixels of the previous three lines (prioritizing the odd lines or the even lines) and then using them as the final pixel values as follows:

$$\begin{aligned} &i\text{th pixel value of } j\text{th line} = [i\text{th pixel of } (j-3)\text{th line} \\ &\quad +2 \times (i\text{th pixel of } (j-2)\text{th line}) + i\text{th pixel of } (j-1)\text{th line}]/4 \end{aligned}$$

Our blending implementation does not use previously rendered video frames. As a result, the artifacts such as "mouse teeth" and "tearing" are noticeably eliminated after applying the averaging mechanism. However, it does have the side effect of blurring the images. Fast motions tend to show less clear images, resulting in poorer video quality. Moreover, interlacing artifacts are still present for fast motions. Our deinterlacing solution did not cause any noticeable performance degradation and its CPU load still remained consistent and stable, similar to the case without it.

In the Windows environment, we tested a hardware supported deinterlacing method, the PureVideo technology available from nVidia Corporation. It performs the motion estimations and compensations through the hardware accelerator on an nVidia graphics card. Surprisingly, its video rendering occupies just about 10% of the CPU load with excellent deinterlaced video output results. We realized that a number of off-the-shelf deinterlacing software libraries available for the Windows environment produced a very decent deinterlaced quality with acceptable CPU load.

As a result, we reached the conclusion to use such off-the-shelf deinterlacing libraries available freely when we moved our development platform to Windows. The only remaining question was whether the video rendering software would still be able to maintain the same degree of low latency that we achieved on the Linux platform.

1.7 Other Challenges

1.7.1 Audio Handling

Multichannel echo cancellation is a largely open research problem. Researchers are pursuing both near-term- and long-term solutions to address the needs of high-quality audio acquisition challenges in conference type environments. Echo cancellation for a single audio channel has been identified as a needed component. Optimal microphone and speaker placements are other design issues. Finally, the output audio quality requirements need to be contrasted and optimized for meeting type environments (as compared to, for example, theater type production environments).

1.7.2 Video Streaming

Optimization of high-quality video in terms of QoS (quality of service)/usability requirements in conjunction with objective performance metrics such as latency is an ongoing research problem. Video streaming issues must be studied in various configurations and settings with new algorithms as well as through usability testing. It should be noted that RCS focuses on HDV quality. As a consequence, a minimum amount of bandwidth must exist in the network, otherwise it is physically impossible to achieve high-quality transmissions. Furthermore, there are constraints on the hardware, which must provide the capabilities and performance required. For example, the RCS rendering system is carefully designed around MPEG-2 and MPEG-4 software decompression modules. To achieve high performance, it is desirable to utilize the hardware capabilities of modern graphics cards. In our current design, a specific combination of graphics hardware, drivers, and software components is necessary to achieve the best possible performance. Further research is required to investigate these trade-offs and to improve performance. It is also important to understand the operating environment in which a remote conferencing system will operate. Public and corporate networks have different characteristics in different parts of the globe.

1.7.3 Stream Format Selection

The RCS software is designed to capture, transmit, and decode MPEG-2 and MPEG-4 bitstreams in the TS format. Although the video rendering software is capable of playing both MPEG-2 formatted TS and PS videos, the software chain was significantly rewritten to optimize the transmission and rendering of TS video streams effectively. The transcoded video output can be either TS or PS formatted.

RCS has also shown the usefulness of a single-pass retransmission mechanism in a lossy network. Some of the retransmitted packets may arrive late or are dropped in the network. The RCS receiver software, aware of the underlying data format, selectively issues a

retransmission request for each lost packet. Changes in the software design, for example, as a result of new transcoding modules, may produce data formats other than TS. These are design choices that need to be carefully analyzed in the context of the overall architecture.

1.8 Other HD Streaming Systems

There are several commercial systems available that focus on high-quality video conferencing (i.e., with a visual quality beyond SD). Among them, two popular high-end systems are highlighted here. The TelePresence system from Cisco Systems provides a specially engineered room environment per site, integrating cameras, displays, meeting table, and sound systems (Szigeti *et al*. 2009). The video images taken from custom-designed high-resolution 1080p video cameras are encoded as 720p or 1080p H.264 bitstreams. The encoded bitrates range either from 1 to 2.25 Mbps for 720p or from 3 to 4 Mbps for 1080p. Unlike the usual MPEG-based compression algorithms, reference frames are constructed aperiodically to encode more efficiently. Individual sound samples, acquired from microphones that are positioned at special locations, are encoded with AAC-LD (advanced audio coding low delay). The encoded bitrate and coding delay are 64 kbps and 20 ms, respectively. The encoded media data are then packetized and multiplexed, using the RTP. The system does not employ any packet loss recovery mechanism, but a receiver, after detecting the packet losses, requests a sender to send a reference frame to rebuild the video image, while disposing unusable frames quickly. The end-to-end latency between two systems, excluding the transmission delay, is estimated less than 200 ms. The Halo system from HP[3] features similar room installations with fully assembled hardware communicating over a private, dedicated network. While the cameras used in Halo are SD, the video streams are upconverted to HD at the display side. Each stream requires about 6 Mbps and each room generally supports four streams. The Halo system is turnkey and fully proprietary. While the above two high-end systems are extremely expensive because of their professional setup, several companies offer an affordable solution. For example, LifeSize[4] features 720p cameras and displays. Its proprietary compressor provides very low bandwidth (e.g., 1.1 Mbps for 720p video). While the camera and compressor are proprietary, the display is generic.

A number of research prototypes similar to our solution were implemented in different research communities; they can be classified into two groups. The first group uses uncompressed HD video streams, which are especially useful for very time-sensitive applications such as distributed musical collaborations. Among them is the UltraGrid system, which transmits uncompressed HD at a bandwidth requirement close to or above 1 Gbps (Gharai *et al*. 2006). The Ultra-Videoconferencing project at McGill University[5] was designed especially for low-latency video conferencing applications. It delivers uncompressed 720p HD sources using HD-SDI at 1.5 Gbps and 12 channels of 24-bit raw PCM (pulse code modulation) data with 96 kHz sampling rate.

The second group uses compressed HD videos and audios, captured from commodity MPEG-2 HD camcorders. Kondo *et al*. at Hiroshima University in Japan experimented

[3] http://www.hp.com/halo/index.html
[4] http://www.lifesize.com/
[5] http://www.cim.mcgill.ca/sre/projects/rtnm/

with an HD delivery system for multiparty video conferencing applications in Linux environment (Kondo *et al*. 2004). Their prototype system captures MPEG-2 transport bitstreams from hardware encoders such as JVC HD camcorder or Broadcom kfir MPEG-2 encoder card, embeds FEC codes (using the Reed–Solomon method) on the fly, interweaves them, and finally shuffles the transmission order of the packets to minimize the effect of burst packet losses. While requiring 10%–50% more transmission bandwidth, its error resilience showed two orders of magnitude packet loss rate reduction. The one-way delay was reported around 600 ms for hardware decoder and 740 ms for software decoder (VLC Client). Audio streams were separately transmitted through a customized RAT (robust audio tool). Similar software packages that were developed for the Windows environment reported much longer latencies (around 1–2 s one-way delay). Compared with these, our system features much lower end-to-end delay with the same capturing setup (due to our software optimization efforts), software-based real-time video transcoding capability, and bandwidth-saving packet relaying mechanism.

1.9 Conclusions and Future Directions

We have discussed design challenges for multiway HD video communications and have reported on recent experimental results of specific approaches built into a prototype system called RCS. We implemented real-time transcoding, a relay mechanism, and deinterlaced video rendering, and deployed these mechanisms successfully, including two simultaneous HD renderers per computer. In case of the transcoding output format, we could obtain MPEG-2 TS or PS formatted 300×200 video output from the original MPEG-2 HD TS videos. Both formats could be supported easily, but we found the TS format to be more resilient.

References

Gharai, L., Lehman, T., Saurin, A. and Perkins, C. (2006) Experiences with High Definition Interactive Video Conferencing. IEEE International Conference on Multimedia & Expo (ICME), Toronto, Canada.

ISO/IEC 13818–1 (1994). Information Technology – Generic Coding of Moving Pictures and Associated Audio: Systems Recommendation H.222.0, International Standard, National Organization for Standardization, ISO/IEC JTC1/SC29/WG11, NO801, 13 November, 1994.

Kondo, T., Nishimura, K. and Aibara, R. (2004) Implementation and evaluation of the robust high-quality video transfer system on the broadband internet. *IEEE/IPSJ International Symposium on Applications and the Internet*, 135.

MacAulay, A., Felts, B. and Fisher, Y. (2005) Whitepaper – IP streaming of MPEG-4: Native RTP vs MPEG-2 transport stream Technical Report, Envivio, Inc.

Papadopoulos, C. and Parulkar, G.M. (1996) *Retransmission-based Error Control for Continuous Media Applications*. Proceedings of the 6th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 1996), Zushi, Japan.

Szigeti, T., McMenamy, K., Saville, R. and Golwacki, A. (2009) *Cisco TelePresence Fundamentals*, 1st edn, Cisco Press, Indianapolis, Indiana.

Wenger, S., Hannuksela, M., Stockhammer, T., Westerlund, M. and Singer, D. (2005) RTP Payload Format for H.264 Video. RFC 3984.

Zimmermann, R., Fu, K., Nahata, N. and Shahabi, C. (2003) *Retransmission-Based Error Control in a Many-to-Many Client-Server Environment*. SPIE Conference on Multimedia Computing and Networking (MMCN), Santa Clara, CA.