ChitChat: Making Video Chat Robust to Packet Loss

by

Jue Wang

B.S. in Electrical Engineering
Tsinghua University, 2008

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Electrical Engineering and Computer Science
August 4, 2010

Certified by .....
Dina Katabi
Associate Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

ChitChat: Making Video Chat Robust to Packet Loss

by

Jue Wang

Submitted to the Department of Electrical Engineering and Computer Science
on August 4, 2010, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Video chat is increasingly popular among Internet users (e.g. Skype, Windows Live Messenger, Google Talk). Often, however, chatting sessions suffer from packet loss, which causes video outage and poor quality. Existing solutions however are unsatisfying. Retransmissions increase the delay and hence can interact negatively with the strict timing requirements of interactive video. FEC codes introduce extra overhead and hence reduce the bandwidth available for video data even in the absence of packet loss.

In this thesis, we present ChitChat, a new approach for reliable video chat that neither delays frames nor introduces bandwidth overhead. The key idea is to ensure that the information in each packet describes the whole frame. As a result, even when some packets are lost, the receiver can still use the received packets to decode a smooth version of the original frame. This reduces frame loss and the resulting video freezes and improves the perceived video quality. We have implemented ChitChat and evaluated it over multiple Internet paths. In comparison to Windows Live Messenger 2009, our method reduces the occurrences of video outage events by more than an order of magnitude.

Thesis Supervisor: Dina Katabi
Title: Associate Professor of Computer Science and Engineering

Acknowledgments

I am really grateful and fortunate to have Dina as my adviser. Her enthusiasm and passion intrigues my curiosity, and largely motivated the work in this thesis. More importantly, she makes me realize the phenomenal capacity people have when they are excited about what they are doing. I would especially like to thank Dina for kindly guiding me through the entire project, discussing with me whenever I was lost and offering prompt and helpful comments and feedback. At the same time, she allowed me much freedom and encouraged me to pursue and explore the ideas I found interesting.

I owe many thanks to Szymon for his patience in discussing with me about detailed implementation issues and testbed setup. I am really grateful to Shyam for his comments throughout the final month of this work and his much appreciated help with the last-minute paper revision. I am also really thankful to my wonderful friends who helped me conduct the video chat experiments in this thesis: Ashwin, Bo, Jie, Jingsi, Nabeel, Siliang, William and Yuanxin. Without their kind assistance, completing this thesis would not have been possible. I would also like to thank my group mates for their great company during my first two years at MIT: Georgios, Haitham, Nate, Nabeel, Rahul, Sam, Shyam and Szymon.

I would really like to thank my parents for their extraordinary support. Encouraging their daughter, who had never been away from home for more than one month before, to go pursue her dream at the other side of the world can be a harder task than it seems. Without my parents' support or Dina's great understanding and letting me visit home during Christmas, the busiest time of the group, the first year at MIT could have been much scarier.

Finally, I am really thankful to Ashwin, whom I can lean on any time and talk to about everything. Thanks, Ashwin, for being so patient with me even when I am frustrated.

Contents

List of Figures

Chapter 1

Introduction

1.1 Is the Internet Ready for the Challenge of Consumer Video Communication?

Video chat is increasingly used for both personal and business communications [31]. Measurements show a fast growth in video conferencing over Skype, Windows Live Messenger, Google Talk, etc. [14, 61, 11, 7]; and the trend is likely to become stronger with the recent introduction of Apple FaceTime over mobile phones [4]. This interest in video communication shows that, if video chat is made reliable, video telephony may gradually replace audio telephony. Today however, Internet video calls suffer from transient packet loss, which causes frame freezes and poor visual quality [17, 34]. Without better mechanisms for handling packet loss, video chat will not reach its full potential.

According to a recent landmark study conducted by Nemertes Research [13], unless service providers invest billions of dollars in updating them, Internet access infrastructures, instead of cores of the Internet, will be a bottleneck for supporting user demand in the next few years. The exploding visual richness of online communications and entertainment is the main reason for this, and can potentially lead to more delay and congestion experienced by end users. With the rapid advance in technology, this is unlikely a disaster, yet it shows that the Internet today is far from over-provisioned.

Common reasons for packet loss include congestion (long-term) due to lack of abundant bandwidth, wireless loss, and large variation (sudden drop) in bandwidth. While rate adaptation helps prevent the first category of losses from happening, wireless random loss and loss caused by large variation in bandwidth (common on access links) are inevitable.

Typically a loss rate lower than 5% is considered acceptable for traditional data transmission, however, video applications are particularly extremely intolerant of packet loss, because of the compression done at the source. Packet loss results in jitters, corrupted images and video stalls. Jitters are much less noticeable and much more readily tolerable than the other two. User visual experience is severely worsened by corrupted images and video stalls. Commercial video chat softwares usually discard a frame if it is corrupted and no longer a smooth degradation of the original frame. This decision leads to more and longer video stalls. Disruption of communication due to video stall or audio-video out-of-sync is without any doubt the biggest annoyance for video chat users nowadays [17].

1.2 What Distinguish Video Chats from Real-time Streaming?

While much of the literature has studied the problem of reliable video delivery in presence of packet loss, it often does not distinguish video chat from live video streaming [41, 23, 69, 54, 20]. In fact, both applications suffer from significant quality degradation in the presence of packet loss. They also both have real-time requirements. Nonetheless, video chat has unique characteristics that set it apart from live streaming. Specifically:

- Due to its interactive nature, video chat has much stricter timing constraints than live streaming. While video streaming is insensitive to a start-up buffering delay of tens of seconds [39], the ITU G.114 standard recommends a maximum delay of 150 ms for video calls in order to ensure lip synchronization [49].

Meeting this timing constraint is challenging given today's processing [56][1] and propagation delays[2]. Hence any additional latency introduced in order to tackle transient packet losses is highly undesirable.

- Since access link is likely to be the connection with the least available bandwidth and hence has the highest probability of packet loss, it has always been the main bottleneck for media transmission both for streaming and interactive applications [3, 12]. However, video chat is more likely to suffer from packet drops at access links than live streaming. Specifically, a video chat application sends video in both directions. Technologies like ADSL and cable modem however have significantly lower uplink speeds than downlink speeds [16]. Currently, most cable plans offer two or three times as much downlink bandwidth as uplink. This design choice is made based on the fact that most users upload rarely compared to their downloads. With the trend that people more and more often share their data (files, photos, videos) with each other, the allocation of uplink/downlink capacity needs to be reexamined, yet it is unlikely to be the case that an efficient design will allocate more or similar overall uplink bandwidth as the overall downlink bandwidth. However, when we make video calls, we use as much uplink bandwidth as download bandwidth, making the uplink a much more sensitive bottleneck for video chat applications.

As a result of these characteristics, solutions for packet loss proposed in the context of streaming applications [41, 23, 69, 54, 20] are mostly ineffective for chatting applications.

(a) Forward error correcting codes (FEC): Adding redundant error correction codes at the sender side has been widely used in conventional data transmission applications without stringent timing constraints. This is a common way to enable the receiver to detect and correct errors due to packet loss or channel noise. In recent

[1]Usually longer processing time yields better compression and hence more efficient usage of the bandwidth.

[2]Round Trip Time in the Internet has been reduced tremendously over the last decade. Currently, most links within north America have an RTT of lower than 50ms; cross-country links' RTT is around 100ms; intercontinental links such as U.S.- China has an RTT of 200-300ms.

years, FEC has been gradually adopted in real-time applications such as streaming, given that the streaming applications can usually allow some start-up buffering as well as tolerating some infrequent buffering during the streaming. This tolerance to buffering allows the application to apply FEC in a way as efficient as in the traditional data transmission applications. On the contrary, FEC codes are inefficient for dealing with transient packet loss in video calls. Typical Internet loss rate is less than 1% [66]. In principle, an FEC code that combines every 100 packets together, adding only one or two packets of redundancy, can recover from such a loss rate. In video chat, however, coding across frames is precluded by the need to play each frame as soon as it arrives, with no extra delay. Coding within a frame requires adding at least a single FEC packet per frame. Yet, each frame in consumer video chat applications (e.g., Skype) is typically sent using 2 to 5 packets [29].[3] Hence, even when the loss rate is as low as 1%, a minimal FEC redundancy of one packet per frame increases bandwidth usage by 20% to 50%. Thus, FEC is likely to increase the loss rate in a congested environment. Indeed, past studies of Skype show that the use of FEC increases the bandwidth overhead by 20% [55] to 50% [29]. In fact, even for IPTV solutions, people have found that "increasing the amount of protection can reduce packet loss to some extent, but the protection capability is limited due to the short time interleaving and the bit rate reductions required to increase the protection could become severe." [52]

(b) Path Diversity: Proposals for routing packets over multiple paths using an overlay network [1, 21, 20, 44], while effective at reducing packet drops in the core of the Internet, cannot deal with losses on the access links of the communicating nodes. Since the latter losses are common in video chat [55], this approach too is not sufficient for these applications. [22, 36]. In fact, while Skype has an overlay network which could potentially route traffic away from congested points in the backbone, Skype uses a relay node only to traverse NAT boxes or firewalls, and prefers the direct path whenever the nodes are not NAT-ed [22].

[3]Skype and Windows Live Messengers, transmit at 200–700 Kbps with an average frame rate of 10-15 fps and an average frame size of 2 to 5 packets [55, 29].

(c) **Retransmissions:** Commercial video chat softwares try to avoid packet retransmissions [22] because such a mechanism requires delaying the received frames and hence interacts badly with the tight timing constraints of video calls. The unification of transmission for video data (streaming with CDN) with traditional data is highly desirable for various communities. Specifically, the adoption of HTTP in one-way streaming is more and more common these days. However, this convergence is unlikely to happen with the case of interactive applications such as video chats or VoIP, because of the tight timing constraints. In other words, even if reliable protocols like HTTP are used, the retransmitted packets are very unlikely to be of much use for interactive video applications. [4] [5]

1.3 ChitChat: a Video Chat Centric Solution

This thesis presents ChitChat, a new approach for dealing with packet loss in video calls. ChitChat neither requires the receiver to delay a frame, nor introduces bandwidth overhead. Further it addresses both edge and core losses, especially effective in the cases of short-term transient changes in bandwidth and the resulting packet buffer overflow. Our basic idea is simple. We want to ensure that the information in each packet in a frame describes the whole frame. As a result, even when some packets are lost, the receiver can still use the received packets to decode a smooth version of the original frame. This reduces frame loss and the resulting video freezes and yields a smooth user experience of better perceived video quality. The idea is inspired by intensive work on both multiple description coding and network coding.

To achieve our objective of having each packet describe the whole frame, we mix

[4] When the caller and the callee are behind port-restricted NAT and UDP-restricted firewall, Skype is forced to send its media traffic over TCP; and TCP's retransmission mechanism will apply. However, Skype prefers the use of UDP for transmitting its media data as long as it has this option [22]. Windows Live Messenger uses HTTP for its "Show My Webcam" mode, which is indeed a single-direction streaming mode.

[5] Retransmitted packets containing important information such as header and keyframes which later frames depend on might be helpful. However, predicting whether these packets will arrive in time to be of any use or they are simply extra burden on the link during congestion is a challenging task.

the information in a frame before presenting it to the video codec. However, naively mixing the information in each frame destroys the temporal and spatial correlation in the raw video, which the codec relies on for efficient compression. In other words, this destroys the codec's ability to recognize objects as they move across frames, and use this information to compress the video. This long-lasting problem results from the gap between the video community and the networking community. Given the large variety of usages of video, one main ultimate goal of the video research community has always been high compression efficiency. However, when videos start to be sent over a best-effort IP network, the networking community focuses on how to improve reliable transmission of these highly compressed videos. Sacrificing compression efficiency too much to improve received video quality is in no one's favor, given the low loss rate of the Internet today. Ideally, reliable video transmission should be achieved using a minimum overhead in size. In ChitChat, to deal with this issue, we design our information mixing algorithm to be shift-invariant (i.e., a movement of an object in the pixel domain translates into a corresponding movement in the mixed frame). We show that this approach allows existing codecs to continue to work efficiently in the presence of information mixing.

We have implemented ChitChat and evaluated it both on a testbed using a CLICK router to introduce packet loss and over multiple Internet paths, within the U.S. and between U.S. and China. We also compared it with Windows Live Messenger 2009, a popular video chat software [11], as well as H.264 and H.264 with one FEC packet per frame. Our results show:

- In comparison to Windows Live Messenger, ChitChat reduces the number of outage events by 15x (from a total of 195 outages to only 13 outages).
- Further, at all loss rates, ChitChat's video quality is higher than that of Windows Live Messenger. Particularly, for loss rates of 1% to 10%, ChitChat improves the average video quality (PSNR) by 3 to 6 dB over Windows Live Messenger.
- Finally, in the absence of packet loss ChitChat delivers the same video quality as Windows Live Messenger.

Chapter 2

Related Work

2.1 Dealing with Packet Loss in Video

Past work has recognized the negative impact of packet loss on video applications [52, 28, 62, 59, 63] and proposed techniques for dealing with the problem [21, 54, 68, 50]. These proposals can be divided into two categories: solutions for Internet-core losses, and solutions for last-hop losses.

2.1.1 Packet Loss at the Internet Cores

Many solutions for dealing with losses in the core of the network employs path diversity, i.e., different versions of the video are sent over independent paths to avoid correlated drops. For example, in [20], the authors use multiple description coding, and send different descriptions over different paths to the receiver. Due to the self-reliance nature of the descriptions, this design can leverage path diversity to improve the video quality in face of core losses. SplitStream [27] employs multiple distribution trees and splits the load among the nodes while taking into account heterogeneous bandwidth resources. PROMISE [38] uses peer-to-peer delivery, monitors the path to potential peers and dynamically switches between them to improve performance. Approaches that use path diversity to reduce losses are complementary to our design. They however cannot deal with losses on access links, while ChitChat can address

both core and last-hop losses. Moreover, loss occurring on the access link is a much more prevalent problem in video transmission nowadays, compared to loss occurring at the cores of the Internet. It is even worse for video chat since conferencing applications need to transmit video in both directions, yet most end systems have low uplink bandwidth.

2.1.2 Packet Loss On Access Links

Past solutions for losses on the access links are primarily limited to retransmission or forward error correction. Some designs buffer corrupted frames and ask for retransmission of lost packets [41, 23]. The stringent timing requirement in interactive applications makes retransmissions much less useful than in conventional applications. Also, if retransmitting certain important information is found beneficial, it can always work in parallel with our design and hence will not be the focus of this thesis. Other designs add forward error correction at the sender to allow the receiver to recover from losses without retransmissions [69, 54]. However, as explained in Chapter 1, FEC is inefficient for today's Internet where the loss rate is usually below 1% [32, 66], and could potentially be recovered by adding only 1% packet redundancy; yet the fact that video chat has to code over a short block length of 1 frame, increases the redundancy overhead to 20% to 50%.

2.1.3 Multiple Description Coding

Multiple description coding (MDC) has been suggested and extensively studied in the past decade [67, 35]. It is considered as a promising solution for dealing with both Internet core losses and access link losses. The goal of MDC is to generate independent descriptions of the same video unit (a video unit can be a frame, a group of pictures (GOP), or an entire video, depending on the application's timing constraints), so that a reconstructed version of proportional degree of fidelity can be derived from the received descriptions in the presence of loss. MDC algorithms proposed in recent years can be generally divided into three categories: postprocessing-stage MDC (FEC

based) [57], encoding-stage MDC [67, 64, 46] and preprocessing-stage MDC [65, 70]. FEC-based MDC schemes [1] is not suitable for video chats as explained in Chapter 1. Encoding-stage MDC schemes typically include multiple description quantization, multiple description transform coding and multiple descriptions of the motion vectors. One major disadvantage of encoding-stage MDC is the requirement of making significant changes to the video codec. In addition, the high computation complexity of some of these encoding-stage MDC designs eliminates the possibility of their usage in real-time applications [46]. Past work on preprocessing-stage MDC are limited to subsampling-based MDC in spatial, temporal, or frequency domain. These subsampling methods disrupt the spatial or temporal correlation in the video sequence and hence result in redundancy. Plus, by simply splitting neighboring frames or pixels, these methods' performance entirely depends on the accuracy of estimating the lost descriptions. In contract, ChitChat's infomation mixing offers error concealment gain even when the motion estimation itself is not accurate. Apart from these unique problems, a lot of the MDC techniques proposed in the past, in all these three categories, produce extra samples to code and hence do not follow the same pattern of MPEG data anymore. Unlike the MDC schemes, our design neither requires changing the video codec, nor produces extra samples for coding, and at the same time, by maintaining the correlation in the video sequence, our design does not introduce overhead in size.

2.1.4 SoftCast

ChitChat's information mixing algorithm is motivated by SoftCast, a recent proposal for wireless video [42]. SoftCast however addresses a different problem: it enables a wireless transmitter to broadcast a video stream that each multicast receiver decodes to a video quality commensurate with its wireless channel quality. SoftCast also requires changing both the physical layer and the video codec, while ChitChat works

[1] FEC-based MDC schemes convert the prioritized output of the codec into unprioritized descriptions by adding progressively weaker FEC protection to video information of decreasing importance.

with existing video codecs and physical layers.

2.2 Rate Adaptation

Also related to our work are mechanisms for media rate control, such as TFRC [37], DCCP [48], and others [45, 18, 53]. These protocols control the transmission rate to ensure that the sender does not transmit way above the available bandwidth and hence does not cause excessive packet loss. Rate control is complementary to our design; it addresses long term persistent losses, rather than transient losses, which are unavoidable in today's best effort Internet subject to common short-term transient bandwidth changes.

2.3 Performance Study of Commercial Applications

Additionally, the last few years have seen interest in studying the mechanisms in commercial video applications and their performances over the Internet. In particular, [29, 24, 55] study Skype's rate adaptation and reaction to changes in available bandwidth. The authors of [55] also study Windows Live Messenger and other video chat applications. All of these studies however operate at the packet level, looking at throughput and packet loss rate, but do not directly tie these measurements to the corresponding video quality. This is due to the proprietary nature of video chat applications which prevents direct access to the decoded video frames. These studies offer great insights for us.

Chapter 3

ChitChat at a High Level

ChitChat is a video-chat centric approach to deal with transient packet loss in best-effort IP networks such as the Internet. Its design ensures that: 1) received frames can be decoded immediately and displayed, 2) no extra bandwidth is required, 3) the approach works with both edge and core losses, and 4) it works seamlessly with state-of-the-art video codecs.

ChitChat's architecture, illustrated in Figure 3-1, has three components that together improve resilience to lost packets:

- The *information mixing* module codes a frame content to ensure that the impact of a packet loss is not concentrated in particular corrupted patches, but is rather smoothly distributed over the frame.
- The *mixing-aware reference selection* module ensures that the video codec can effectively compress the content in the presence of information mixing.
- The *interleaving* module distributes the video data into packets in a manner that allows the receiver to reconstruct a smoothly degraded version of a frame from any subset of its received packets.

As the figure shows, ChitChat operates in conjunction with standard codecs (e.g., MPEG-4/H.264), which makes it easy to integrate in existing video chat applications. The following sections describe ChitChat's components in detail.

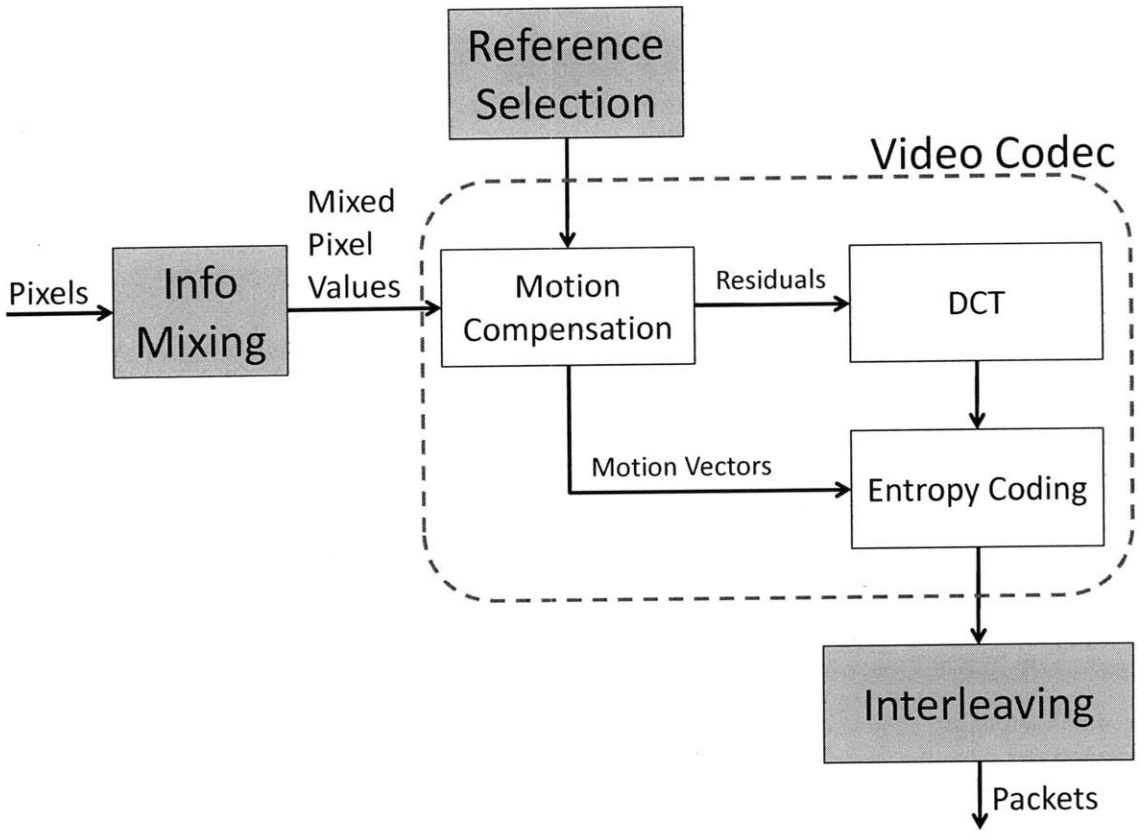


Figure 3-1: **A Block Diagram of ChitChat's Architecture.** The gray boxes refer to ChitChat's components. The white boxes are typical components of today's codecs.

Chapter 4

Information Mixing

ChitChat introduces a preprocessing step that mixes the pixel values in each frame before passing the frames to the video codec. The mixing is done using a Hadamard transform [43]. Hadamard coding has been used in multiple prior systems to smooth out the impact of channel errors. For example, both MIMO systems [19] and SoftCast [42] apply a Hadamard-based code to the symbols transmitted over a wireless channel. In contrast, ChitChat applies a Hadamard transform to the pixels directly. Hence, it can operate without changing the physical layer or the video codec. To the best of our knowledge, ChitChat is the first system to show that applying a Hadamard transform to the pixels themselves improves the video quality in the presence of packet loss. In this chapter, we first explain the Hadamard transform, then describe how we apply it in our context.

4.1 Hadamard Transform

The Hadamard matrix is an orthogonal matrix whose entries are -1 and 1. In particular, the 4-dimensional Hadamard matrix looks as follows:

$$H = \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

The Hadamard transformed version of a, b, c, and d is

$$\begin{bmatrix} \hat{a} \\ \hat{b} \\ \hat{c} \\ \hat{d} \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Hadamard transform has the property of evenly distributing the error. If we receive corrupted versions of the transmitted values: $\hat{a} + e_a$, $\hat{b} + e_b$, $\hat{c} + e_c$, $\hat{d} + e_d$, with errors e_a, e_b, e_c, e_d, we can reconstruct the original values as:

$$\begin{aligned} \begin{bmatrix} \tilde{a} \\ \tilde{b} \\ \tilde{c} \\ \tilde{d} \end{bmatrix} &= \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \hat{a} + e_a \\ \hat{b} + e_b \\ \hat{c} + e_c \\ \hat{d} + e_d \end{bmatrix} \\ &= \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} + \frac{1}{2} \cdot \begin{bmatrix} e_a + e_b + e_c + e_d \\ e_a - e_b + e_c - e_d \\ e_a + e_b - e_c - e_d \\ e_a - e_b - e_c + e_d \end{bmatrix} \end{aligned}$$

From the above equation we can see that any error in the transmitted signal $\hat{a}$ or $\hat{b}$ or $\hat{c}$ or $\hat{d}$ is evenly distributed to the 4 reconstructed numbers.

At the same time, the sum of square error is still $e_a^2 + e_b^2 + e_c^2 + e_d^2$. (This is because

Hadamard is an orthogonal matrix.)

The two properties above work greatly towards our goal of a smooth degradation in face of loss or error.

- By applying Hadamard, we will be able to distribute the channel noise evenly on the pixels we combine together.
- Quantization noise which results from the lossy compression applied by the codec is still evenly distributed on pixels as if did not apply Hadamard to them.
- The total sum of square error does not change, meaning that given the same degree of quantization and same amount of loss, the overall quality (i.e., PSNR) of the part of image we are combining will not change.

Said differently, Hadamard distributes the noise resulting from a packet loss over a frame without adding any extra noise. This creates a smooth frame with no jarring corrupted patches. Also when the noise is distributed over a larger area, it tends to become easier to correct using simple denoising algorithms leveraging only local information.[1]

4.2 Hadamard Mixing in ChitChat

We apply the Hadamard transform directly in the pixel domain, before any compression. Given a frame, we first compute the average luminance in the frame and subtract it from all pixels. We refer to this average as the DC value in the frame. We pass this value directly to the interleaving component which includes it in every packet for this frame to ensure its most reliable delivery in the face of losses. Removing the DC value before coding images or frames is typical in the literature [33] and does not change how the codec works.

As in MPEG, we divide the frame into macroblocks, where each macroblock contains 16×16 pixels. We take every 4 adjacent macroblocks and code them together

[1]Simple denoising filters are usually just low-pass filters. Since low-pass filters consider only the difference between adjacent pixels, reducing the local noise magnitude generally renders the image easier to smooth by low-pass filters.

using the Hadamard matrix. In particular, consider 4 adjacent macroblocks, A, B, C and D. We represent these macroblocks as follows:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \left[\begin{bmatrix} a_1 & a_{17} & \cdots & a_{225} \\ a_2 & a_{18} & \cdots & a_{226} \\ \vdots & \vdots & \ddots & \vdots \\ a_{16} & a_{32} & \cdots & a_{256} \end{bmatrix} \begin{bmatrix} b_1 & b_{17} & \cdots & b_{225} \\ b_2 & b_{18} & \cdots & b_{226} \\ \vdots & \vdots & \ddots & \vdots \\ b_{16} & b_{32} & \cdots & b_{256} \end{bmatrix} \right.$$
$$\left. \begin{bmatrix} c_1 & c_{17} & \cdots & c_{225} \\ c_2 & c_{18} & \cdots & c_{226} \\ \vdots & \vdots & \ddots & \vdots \\ c_{16} & c_{32} & \cdots & c_{256} \end{bmatrix} \begin{bmatrix} d_1 & d_{17} & \cdots & d_{225} \\ d_2 & d_{18} & \cdots & d_{226} \\ \vdots & \vdots & \ddots & \vdots \\ d_{16} & d_{32} & \cdots & d_{256} \end{bmatrix} \right]$$

To apply the Hadamard transform on these 4 macroblocks, we rearrange the values in each macroblock into a vector, e.g., $(a_1, a_2, \cdots, a_{256})$, and use the Hadamard matrix, H, to combine the 4 vectors:

$$\begin{bmatrix} \hat{a}_1 & \hat{a}_2 & \cdots & \hat{a}_{256} \\ \hat{b}_1 & \hat{b}_2 & \cdots & \hat{b}_{256} \\ \hat{c}_1 & \hat{c}_2 & \cdots & \hat{c}_{256} \\ \hat{d}_1 & \hat{d}_2 & \cdots & \hat{d}_{256} \end{bmatrix} = H \cdot \begin{bmatrix} a_1 & a_2 & \cdots & a_{256} \\ b_1 & b_2 & \cdots & b_{256} \\ c_1 & c_2 & \cdots & c_{256} \\ d_1 & d_2 & \cdots & d_{256} \end{bmatrix}$$

We then rearrange the coded vectors into 4 mixed macroblocks $\hat{A}$, $\hat{B}$, $\hat{C}$, and $\hat{D}$. We repeat the process on all non-overlapping sets of 4 adjacent macroblocks in the original frame to produce a mixed frame.

4.3 Effect of Hadamard Mixing on a Toy Example

Video codecs typically code frames as differences with respect to the previous frame (or any chosen reference frame). Thus, when a packet is lost for a frame, after error concealment, most likely we lose the corresponding differences, which leads to some noisy macroblocks. To understand the effect of Hadamard mixing on these losses, let

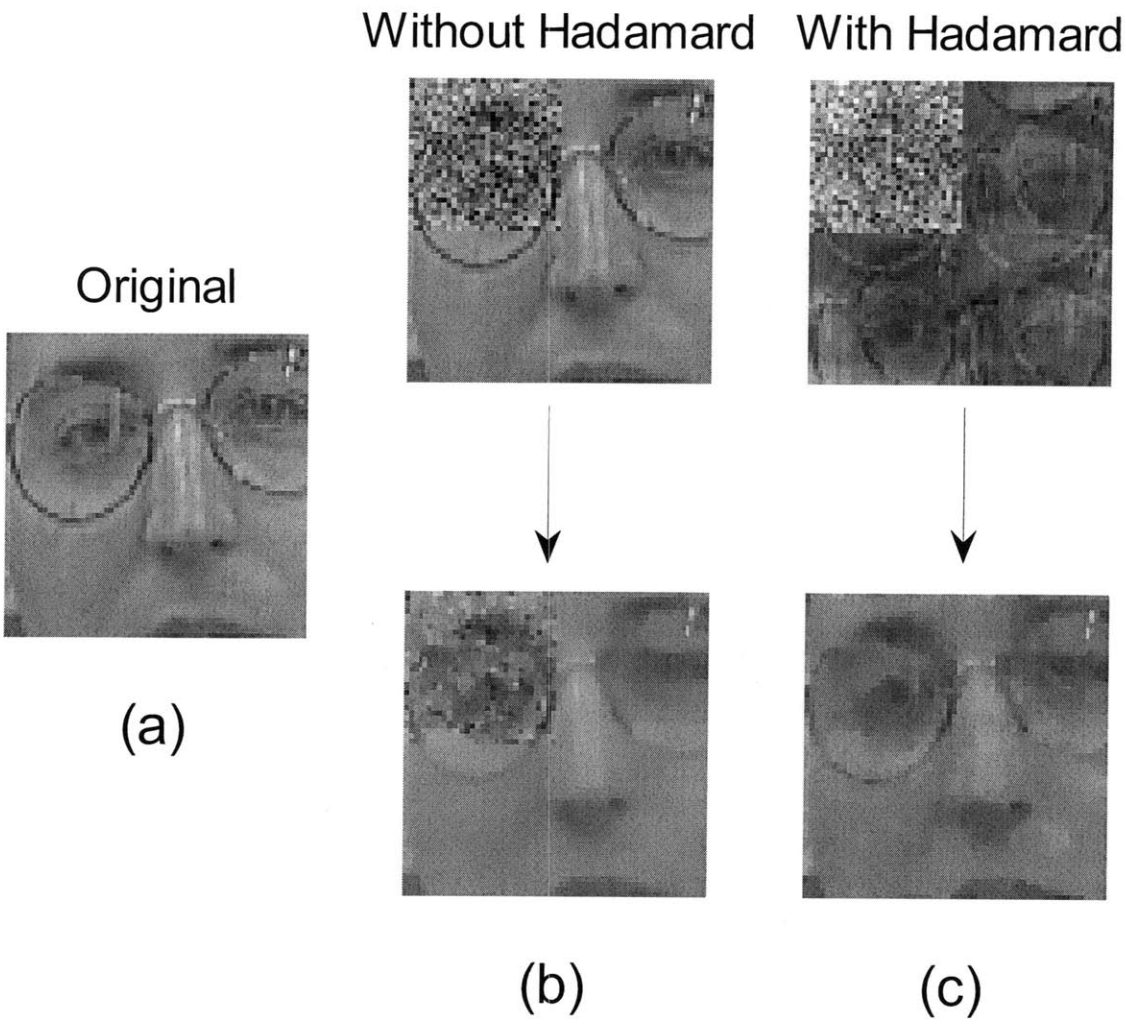


Figure 4-1: **Hadamard mixing spreads the error and produces less jarring frames.** (a) shows the original image (b) shows the image after adding noise to a macroblock and denoising the resulting erroneous block, (c) shows a Hadamard-mixed version of the image after adding noise, unmixing, and then denoising. The figures shows that Hadamard mixing spreads the noise, allowing the receiver to easily denoise the image.

us consider the toy example in Fig 4-1. Specifically, we take an image, corrupt one of its macroblocks, and observe the impact of corruption, with and without Hadamard mixing. Figure 4-1(a) shows the original image; whereas the top row in Figures 4-1(b) and 4-1(c) shows the image after adding random noise for the cases with and without Hadamard mixing. (The noise is Gaussian with a mean of 0 and a standard deviation of 40. These parameters are set to produce noise magnitudes comparable to what we see due to packet losses in our experiments.)

Then, we try to recover the original image by denoising. Denoising filtering is recommended by MPEG-4 in post-processing [58]. Here we use a simple smoothing filter which only operates on adjacent pixels. For the Hadamard-mixed image, however, we first invert the Hadamard transform before applying denoising. The results after denoising are shown in the bottom row in Figures 4-1(b) and 4-1(c). As can be seen from the figures, the errors with Hadamard-mixing look less jarring, which shows the benefit of applying Hadamard.

Note that the error we will see due to packet loss will not, in general, be random Gaussian noise; and will depend on the lost information. But as we show in Chapter 7, the general effect of Hadamard continues to spread the noise and deliver a better video quality.

4.4 Why Not Use a Bigger Hadamard Matrix

One may wonder why we do not combine more macroblocks together, instead of just four. Isn't it even better if we can distribute the noise across a wider range of blocks (a larger area)? The answer is mainly twofold.

- As explained in Chapter 1, in video chat one should code only within a frame to ensure that each received frame can be immediately decoded and displayed. However, the average size of a frame in off-the-shelf video conferencing programs (e.g., Skype, Windows Live Messenger, etc.) is 2-5 packets [55]. Hence, there is no point mixing more macroblocks.
- We will see in Chapter 5, that the number of reference frames that we need

to pass to the video codec corresponds to the number of macroblocks mixed together, and hence mixing too many macroblocks can increase the complexity unnecessarily.[2]

[2]There is another subtle reason, which is the flexibility of motion compensation. One can imagine combining all the macroblocks in a frame. The block motion compensation of the mixed frame will be almost equivalent to global motion compensation of the original image.

Chapter 5

Mixing-Aware Reference Selection

5.1 Motion Compensation in Video Compression

Video codecs (e.g., MPEG-4/H.264) exploit correlation across frames to compress the video. They do so by encoding a frame with respect to a prior frame, called a reference frame. Specifically, for each block in the current frame, the codec finds the closest block in the reference frame, where the distance is typically computed as the sum of square errors [60]. The codec computes a motion vector that represents the vertical and horizontal shifts between the block and its closest reference block. It then encodes the current block using the differences from its closest reference block and the motion vector. This form of compression is called motion compensation.

Motion compensation is the most expensive but also the most valuable step in achieving compression in state-of-the-art video codecs. A lot of work has been dedicated to finding faster, more flexible or more efficient motion search algorithms. Allowing more flexibility in motion compensation in general means expanding the motion search area, increasing the number of reference frames, and reducing the size of the base unit that the codec performs motion compensation on. More often than not, more flexibility results in more bits spent on motion vectors and less bits spent on encoding the residual. Deciding the optimal degree of flexibility has always been a difficult problem in video coding. Yet it is not the focus of this thesis. In our design, our goal is to allow the same degree of flexibility as the base video codec decides to

use. The typical block size for motion compensation is 16×16 pixels, meaning that the codec finds a close match for each non-overlapping 16×16 block in a frame. MPEG-4 however has the option of using different block sizes depending on the amount of motion in the video. In video conferencing applications, the background is often relatively static and we expect slow/medium motion except for the speakers' faces and gestures. So in the prototype we built, we only implemented 16×16 block size, although all of the MPEG-4 sizes can be easily incorporated in our algorithm once we have the auxiliary reference frames as we will describe later in this chapter.

5.2 Obstacles in doing Motion Compensation after Information Mixing

One cannot naively give the codec the sequence of Hadamard-mixed frames and expect motion compensation to continue to work as if the frames were not mixed. Figure 5-1 illustrates this issue. The first column in the figure shows two consecutive frames in the pixel domain. The ball has moved between the two frames. Thus, the codec can compress the current frame simply by representing the macroblock with the ball as a shifted version of some block in the previous frame. The middle column in the figure shows the same two frames after Hadamard-mixing. It is no longer the case that one can see an object that moved between the current frame and the previous frame. Thus, if we simply give the codec the sequence of Hadamard-mixed frames, motion compensation would fail in finding how a block moved across frames.

Luckily however the way we apply the Hadamard transform is *shift invariant* as will be proved. That is if an object shifts position in the original pixel domain, it will also shift by the same amount after the Hadamard transform. One may then wonder how come the two Hadamard-mixed frames in column (b) in Figure 5-1 do not show a moving object? The reason is simple: It is because the boundaries of the blocks over which we perform the Hadamard transform do not shift with the moving object. Figure 5-1(c) shows an alternative Hadamard transform of the original frames in

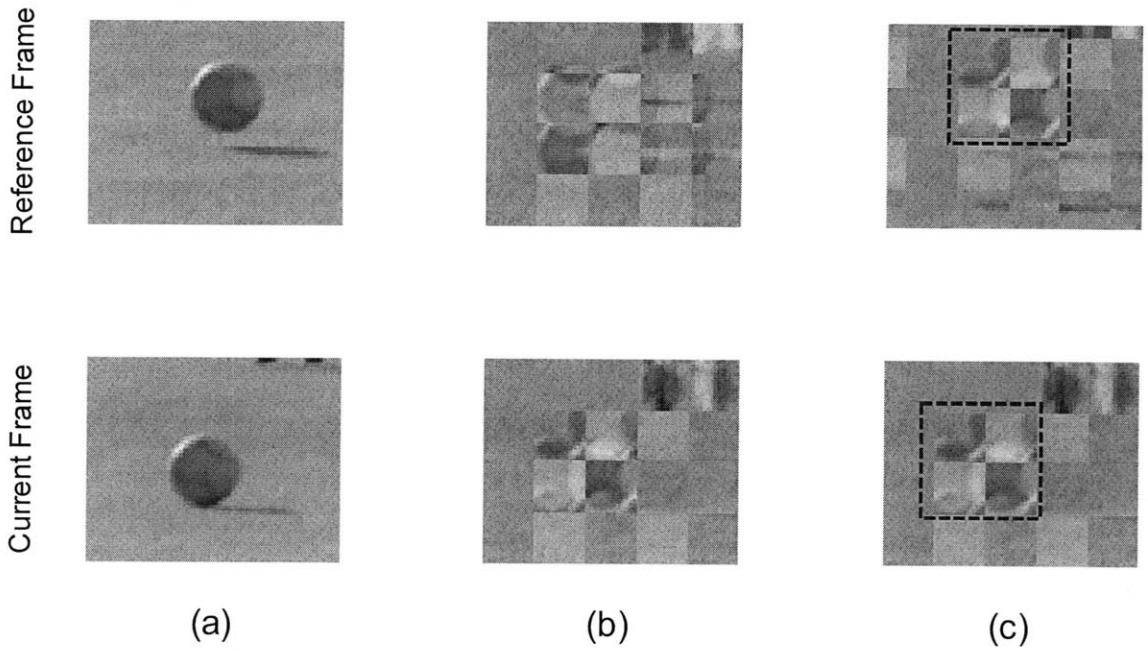


Figure 5-1: **Challenges with motion compensation.** Column (a) shows two consecutive frames in the pixel domain. The ball has moved from one frame to the next. Column (b) shows the same two frames after Hadamard transform. One cannot see an object that moved across the two frames though the original frames represent a moving ball. Column (c) shows the same two frames with an alternative Hadamard transform where the boundaries of the combined blocks move from one frame to the next, with the moving ball. Now, one can see that the area in the dashed square moved from one frame to the next.

Figure 5-1(a) where we move the boundaries of the Hadamard-combined blocks with the moving ball. These frames show a moving block, and hence are easily compressed by today's codecs.

The above argument shows that one can perform effective motion compensation if one computes the Hadamard transform over all possible shifts of the macroblocks. Even if we assume that object movement between frames stays within a limited area, and hence limit our search, similarly to MPEG, to an area of -16 to 15 in both the x and y coordinates, there are 1024 candidate shifts for the Hadamard transform. Given a typical frame size of 240×320 and a block size of 16×16 we need to do Hadamard transform $240 \times 320/16/16 \times 1024 = 307,200$ times for each frame. this

overhead is unacceptable.

This problem, however, can be solved with a negligible computation overhead. The key insight is that there is a huge amount of shared information between these shifted Hadamard-transformed blocks. In particular, since these shifted blocks that we need to calculate Hadamard transform on share a great number of pixels and our mixing is shift invariant, we can perform the computation for each pixel once and reuse it in all these blocks, and thus significantly reduce the overhead. Below we explain our algorithm which exploits this insight.

5.3 Algorithm

Let's define the 4 adjacent macroblocks we are working on as:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$\begin{aligned} A &= F\left(x_0 : x_0 + 15, y_0 : y_0 + 15\right) \\ B &= F\left(x_0 + 16 : x_0 + 31, y_0 : y_0 + 15\right) \\ C &= F\left(x_0 : x_0 + 15, y_0 + 16 : y_0 + 31\right) \\ D &= F\left(x_0 + 16 : x_0 + 31, y_0 + 16 : y_0 + 31\right) \end{aligned}$$

After the Hadamard transform we have:

$$\begin{bmatrix} \hat{A} & \hat{B} \\ \hat{C} & \hat{D} \end{bmatrix} = H\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right)$$

Let frame R be the reference frame for frame F. Many codecs use the previous frame as the reference frame. However, our algorithm can work with any reference chosen by the codec. And in 5.4, we will discuss the impact of different choices in selecting reference frames.

We construct 4 auxiliary reference frames in transform domain $R_{\hat{A}}$, $R_{\hat{B}}$, $R_{\hat{C}}$, $R_{\hat{D}}$

in the following way.

$$R_{\hat{A}}(x,y) = \frac{1}{2}[R(x,y) + R(x+16,y) + R(x,y+16) + R(x+16,y+16)]$$

$$R_{\hat{B}}(x,y) = \frac{1}{2}[R(x-16,y) - R(x,y) + R(x-16,y+16) - R(x,y+16)]$$

$$R_{\hat{C}}(x,y) = \frac{1}{2}[R(x,y-16) + R(x+16,y-16) - R(x,y) - R(x+16,y)]$$

$$R_{\hat{D}}(x,y) = \frac{1}{2}[R(x-16,y-16) - R(x,y-16) - R(x-16,y) + R(x,y)]$$

When the pixel is out of range, e.g., $x - 16 \leq 0$, we use the boundary pixel.

Given the above auxiliary reference frames, motion compensation on Hadamard-mixed frames requires only a minor tweak on motion compensation in the pixel domain. Specifically, without information mixing, given a macroblock in frame F, the codec would search an area of width r_x and height r_y in the reference frame R, looking for the closest block. However, with information mixing, the codec treats each one of the mixed macroblocks slightly differently. When it searches for a closest representation of $\hat{A}$ in the reference, the codec searches in the neighboring region in the auxiliary frame $R_{\hat{A}}$. Similarly, to find the prediction for $\hat{B}$, it searches in $R_{\hat{B}}$; in $R_{\hat{C}}$ to find a prediction for $\hat{C}$; and in $R_{\hat{D}}$ to find a prediction for $\hat{D}$.

This approach effectively solves the boundary issues discussed earlier. In particular, because of fixed boundaries, a pixel which would have been in $\hat{A}$ in the original frame could end up in some $\hat{B}$ in the reference frame, in which case motion compensation would fail to find how a block moves across frames. To address this issue, our algorithm essentially removes these boundaries in the reference frame by computing the four auxiliary frames. Thus, we create auxiliary reference $R_{\hat{A}}$, which has only the $\hat{A}$ transform, auxiliary reference $R_{\hat{B}}$ which has only the $\hat{B}$ transform, and so on. We can then perform motion compensation for each of these blocks by searching in the corresponding reference frame. [1]

[1] By shifting in $R_{\hat{A}}$, we are essentially doing motion search in the original reference frame R, because the $\hat{A}$ in

$$H(R(x : x+31, y : y+31))$$

Note that once the four auxiliary frames are computed, the search for the closest block has the same complexity as in the case without information mixing. This is because, for each Hadamard-mixed macroblock, the codec needs to search only a single auxiliary reference frame ($R_{\hat{A}}$, $R_{\hat{B}}$, $R_{\hat{C}}$ or $R_{\hat{D}}$). The time for computing the auxiliary frames themselves is relatively small in comparison to the motion search, which requires exploring all possible pixel shifts, and is known to be the most computationally expensive part of today's codecs [60].

Finally, after motion compensation, today's codecs encode the residuals using 8×8 two dimensional DCT, order the 64 DCT coefficients in zigzag order, and use entropy coding to encode the motion vectors and DCT coefficients [60]. With ChitChat, these steps proceed without modification.

5.4 Intra-coded Frames and Reference Frame Selection

5.4.1 GOP: Group of Pictures

Video codecs occasionally send frames that are coded independently without any reference frames. These frames are called I-frames. In MPEG, as we have described in previous sections, temporal dependency in the video is explored to reduce the compressed video size. Specifically, a group of successive pictures (GOP) are coded together. The first frame of a GOP is an I-frame (intra-coded frame), followed by P-frames (predictive-coded frames) or B-frames (bidirectional-predictive-coded frames). P-frames contain the motion compensation difference information, i.e. motion vectors and residual coefficients, from the previous I-frame or P-frame. B-frames are both forward and backward predicted. In other words, B-frames are coded based on both

---

is exactly

$$R_{\hat{A}}(x : x + 15, y : y + 15).$$

The function $R(x_1 : x_2, y_1 : y_2)$ returns the pixel values in the area that lies horizontally between x_1 and x_2 and vertically between y_1 and y_2 in frame R, and similarly for $R_{\hat{A}}$. Therefore, searching for $\hat{A}$ in $R_{\hat{A}}$ will identify motion in the original video. The proof for $R_{\hat{B}}$, $R_{\hat{C}}$ and $R_{\hat{D}}$ are identical.

the previous I-frame or P-frame and the succeeding I-frame or P-frame. Clearly, enabling B-frame usage will lead to a delay in the video application, and therefore is not suitable for interactive applications, such as video conferencing.

In a GOP starting with an I-frame followed by N P-frames, if the i^{th} frame is corrupted due to packet erasure or channel noise, later frames are all affected. The encoder will always encode the next frame based on the assumption that the decoder gets all the information it sends before and the decoder always accurately knows what the reference frame is. When loss happens and the decoder fails to get (part of) a frame, the encoder and decoder will have different perceptions of the reference frames used for later frames. This is known as a drifting effect or mismatch between the encoder and decoder. [2]

In general, error in a frame will propagate to all the frames after it in the same GOP, and it only stops at the border of this GOP. We can think of the beginning of every GOP (or the usage of every I-frame) as a resynchronization between the encoder and decoder. A shorter GOP implies a more robust system, but a larger size because I-frames are much larger in size than P-frames. Adaptive GOP size has been suggested and adopted (Skype) in both literature and real systems. In video chat, I-frames are infrequent since there is rarely a change of scene, and hence coding with respect to a prior frame is significantly more efficient than introducing an independent frame. In fact chatting programs tend to introduce an I-frame every 300 frames [30] or whenever loss is detected.

From the perspective of ChitChat, I-frames can be regarded as frames whose reference is an all-zero frame. With this representation, our description applies to I-frames as well. Since the estimation of channel quality and the selection of proper GOP size is not an easy task, in our design, we get rid of the notion of GOP and take a different approach by using acknowledgment.

[2]Intra-coded blocks are used in P-frames when the motion is huge, or periodically with some probability to improve robustness. These blocks will not be affected by errors in previous frames in the same GOP. However in a video conferencing environment with slow/medium motion, usage of a large number of intra-coded blocks will result in a much larger size.

5.4.2 ACK or NACK?

An acknowledgment system is a natural way of synchronization of the encoder and the decoder in a video chat system. Moreover, commercial video chat applications already leverage acknowledgments for rate adaptation. Positive acknowledgment (ACK) and negative acknowledgment (NACK) are two forms commonly used in video chat.

- In a positive acknowledgment (ACK) system, the encoder will encode the current frame using ONLY the acknowledged frames as candidate references, not necessarily the immediately proceeding frames. And it explicitly lets the decoder know which reference frames it used as part of the header information. Whenever the decoder successfully decodes a frame (receives all the packets for the frame), it sends back a positive acknowledgment message (ACK) indicating the most recent frames it decoded. The decoder reference frame buffer can delete any frame earlier than the latest reference frames the encoder uses. And the encoder will discard any frame in its reference frame buffer prior to the latest acknowledged frames. [3] In an ACK system like this, the encoder and decoder are always synchronized completely at any moment. In other words, the decoder always has the reference frames the encoder uses to encode each frame; the encoder always knows what frames the decoder has in its reference frame buffer.
- Instead of using only ACKed frames as reference to ensure 100% synchronization, an alternative is to let the sender always use the immediately preceding frames as reference. When the decoder finds loss has occurred during transmission, it can send back a negative acknowledgment (NACK) to let the sender know what information is missed by the receiver. This negative acknowledgment message can be repeatedly sent many times to ensure the sender gets it. Such a NACK system makes sure that the drifting effect lasts no longer than the time for the sender to get the NACK message and resynchronize the encoder with

[3] In this framework, we eliminate the notion of GOP and I-frames (except for the very first frame of the video). Every frame is coded with respect to some reference frames. Of course intra-coded blocks are still allowed when motion is large.

the decoder then.

In cases of short RTT, a NACK system is effectively the same as an ACK system. In cases of long RTT, there is a trade-off between robustness and video size. The advantage of an ACK system is that the encoder and the decoder are always synchronized and hence the drifting effect is completely eliminated and there is no error propagation across frames. However, this robustness comes at a price of the increase in video size. The gap in an ACK system between a frame and its reference frame is larger than one RTT, because the sender only uses acknowledged frames as reference frames. The larger the gap is, the less temporal correlation there is between a frame and its reference frame. Since the codecs rely heavily on motion compensation across frames to achieve compression, this reduction in temporal correlation translates to increase in the compressed video size. In contrast, always using the previous frame as the reference frame, a NACK system requires less bandwidth than an ACK system, when no loss occurs. This difference in video size depends on the RTT, as will be shown in Chapter 9. When loss occurs, it takes one RTT for a NACK system to resynchronize. Hence the disadvantage of a NACK system is that it is more vulnerable than an ACK system when the RTT is large. In a nutshell, in cases of long RTT, using an ACK system guarantees robustness yet results in a larger size; using a NACK system does not introduce overhead in size but it suffers for one RTT when loss happens.

Considering the trade-off between robustness and video size, one should decide whether to use ACK or NACK based on the RTT and the loss rate of the channel. Different chatting programs use different algorithms to make this decision. ChitChat can leave this decision to the video codec and work with both ACK and NACK designs. However, in our implementation and evaluation, we use a NACK design if not stated otherwise. One thing to note here is that, with ChitChat's information mixing and error concealment, the degradation of the visual experience during the drifting effect in a NACK design is ameliorated to a great extent as shown in Chapter 9.

Chapter 6

Interleaving and Packetization

After the codec is done compressing the video, for each mixed macroblock, it produces a motion vector and quantized DCT coefficients. Next, the macroblocks (with their motion vectors and DCT coefficients) need to be distributed into packets.

In current off-the-shelf video conferencing software, e.g. Skype, Windows Live Messenger, Google Talk, only a few packets are sent for each frame in the video. From our experiment, we see an average packet size of about 700 bytes and 2-5 packets are sent for each frame of size 240×320. This means each packet contains the information for approximately 100 macroblocks. Without interleaving, adjacent macroblocks, $\hat{A}$, $\hat{B}$, $\hat{C}$, and $\hat{D}$, will be transmitted in the same packet, which renders the goal of distributing error in a frame impossible.

Given that our algorithm enables distribution of the error among adjacent blocks, we certainly need interleaving to put the information for $\hat{A}$, $\hat{B}$, $\hat{C}$ and $\hat{D}$ in different packets. Further, given the bursty nature of packet loss in the Internet, we would definitely want to put the information for $\hat{A}$, $\hat{B}$, $\hat{C}$ and $\hat{D}$ in packets as far away from each other as possible.

Thus, we use an interleaving matrix to reorder the macroblocks in one frame, as shown in Figure 6-1. Each cell in the graph represents one mixed macroblock. The number in the cell is the new position of the macroblock in the order after interleaving. N is the total number of macroblocks in the frame. K is the number of 32×32-size blocks in each column in a frame. Adjacent 32×32-size blocks are shaded in different

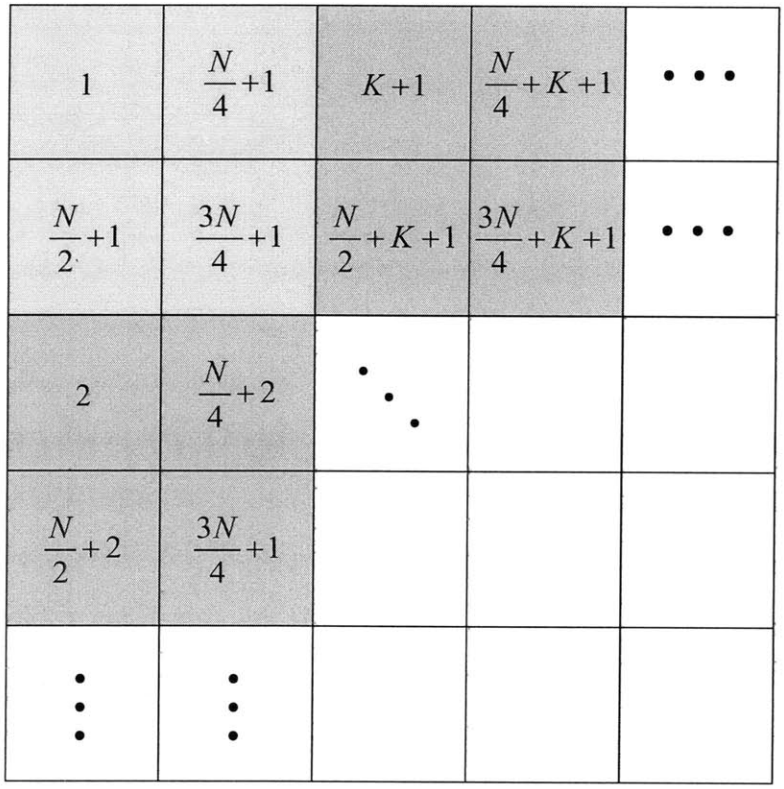


Figure 6-1: **Interleaving the Macroblocks.** The macroblocks are interleaved to ensure that adjacent macroblocks will not be transmitted in the same packet.

colors. The idea is to put $\hat{A}$, $\hat{B}$, $\hat{C}$ and $\hat{D}$ as far away from each other as possible in the new order. Since there are N macroblocks in the frame, we perform interleaving so that $\hat{A}$, $\hat{B}$, $\hat{C}$ and $\hat{D}$ have a minimum gap of $\frac{N}{4}$ in the transmitting order. For example, $\hat{A}$ of the upper-left-most 32×32-size block will be the first in the new order after interleaving; $\hat{B}$ will be the $\frac{N}{4} + 1^{th}$; $\hat{C}$ will be the $\frac{N}{2} + 1^{th}$; $\hat{D}$ will be the $\frac{3N}{4} + 1^{th}$. And as we move on to the next 32×32-size block, we can put the $\hat{A}$ of this block in the position after the $\hat{A}$ of the previous block, which will be the second in the transmitting order.

This interleaving order ensures that information for $\hat{A}$, $\hat{B}$, $\hat{C}$ and $\hat{D}$ is placed as far away from each other as possible in transmission, which means $\hat{A}$, $\hat{B}$, $\hat{C}$ and $\hat{D}$ will end up being transmitted in as many different packets as possible, even if the application is unaware of how the encoded video will be packetized later.

H.264 MPEG-4 Part10 has defined a set of error concealment mechanisms at the

encoder side, notably Flexible Macroblock Ordering and Data Partitioning [60]. The objective of Flexible Macroblock Ordering, similar to ours, is to scatter the errors as evenly across the entire frame as possible, which will make the error concealment of Data Partitioning possible. However, without the information mixing proposed in this thesis, flexible macroblock ordering typically leads to the jarring visual effect in packet erasure environment as we discussed earlier in Chapter 4. In ChitChat, the interleaving not only scatters the error across the entire frame, with the aid of its Hadamard mixing, it also yields a smooth visual experience within a limited region.

Chapter 7

Video Recovery at the Receiver

7.1 Decoding and Error Concealment

At the decoder, we reorder the macroblocks back into the original spatial order before interleaving. Since our interleaving matrix is not random, from the packet loss information we will know exactly which macroblocks are lost.

We distinguish three scenarios. First, if no packet loss occurs, all macroblocks will be received and the frame can be fully recovered by inverting the encoding applied by the video codec, then taking the inverse Hadamard transform. (In fact, the inverse Hadamard transform is the same as the Hadamard transform.)

Second, if all the packets for a frame are lost, then the best any scheme can do is to present the last reconstructed usable frame to the user.

Third, it is uncommon for all the packets for a frame to be lost, based on our experiment and other literature [26]. Knowing the interleaving matrix, the receiver can map the lost packets into lost macroblocks. Figure 7-1 shows an example of the loss map from our experiment. Three packets were sent for this frame and their sizes are 568, 856 and 774 bytes. The second packet was lost during transmission. The white blocks are the macroblocks in the mixed frame whose information the decoder does not know due to this packet loss. The black blocks are the macroblocks which are not affected. Note that when a macroblock is lost, both its motion vector and the residuals from subtracting the reference block are lost, because these two are in the same

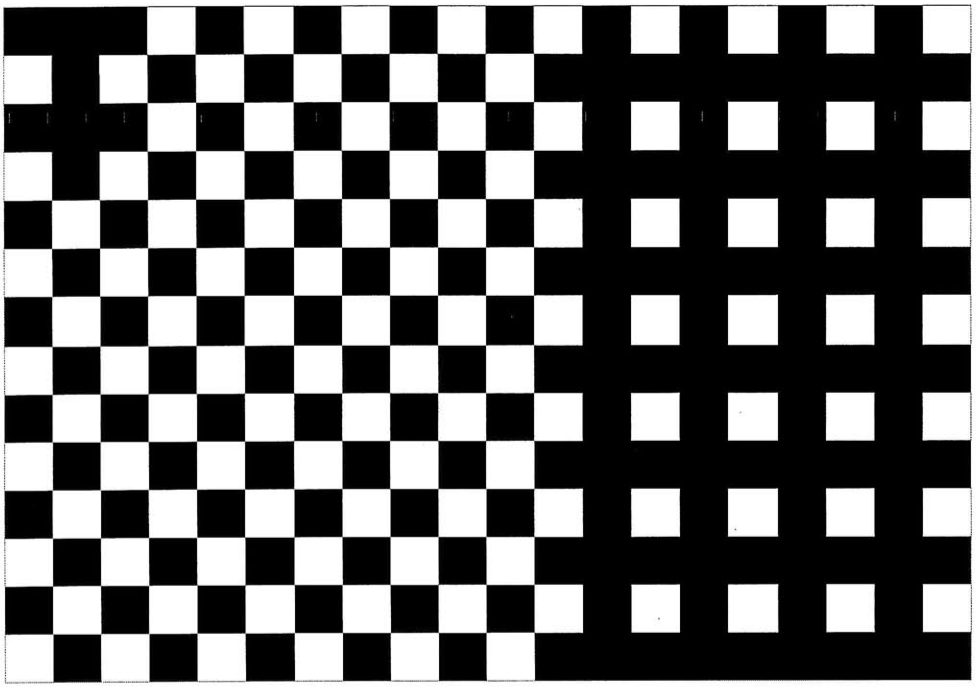


Figure 7-1: **Example of the Loss Map of a Frame.** Three packets of size 568, 856 and 774 bytes are sent for this frame and the second packet is lost. White color indicates the corresponding macroblock after the mixing is lost.

packet. The residuals should almost always be sent in the same packet as the motion vector since they are meaningless without the knowledge of which reference block was subtracted. Thus, to recover a lost macroblock we need to estimate its motion vector and the residuals. In this case, normally MPEG depends on Data Partition and Flexible Macroblock Ordering for error concealment. Apart from estimating the motion vectors in a similar way as MPEG, ChitChat's error concealment also benefits from information mixing.

7.1.1 Estimating Lost Motion Vectors

As in today's codecs, we estimate a lost motion vector using the motion vectors of nearby blocks [60]. For example, if the motion vector for $\hat{C}$ is lost, we will use the motion vector of $\hat{A}$ or $\hat{B}$ or $\hat{D}$ (the first one that the decoder has received) as an estimation for $\hat{C}$.

Figure 7-2: **Error Concealment.** (a) is the original image. (b) is the reconstructed version with ChitChat's error concealment. (c) is the reconstructed version with MPEG's error concealment only.

7.1.2 Error Concealment by Unmixing

Now that we estimated the lost motion vector, how do we estimate the lost residuals? In fact, there is no need to estimate the lost residuals. Since $\hat{C}$ contains mixed information about A, B, C and D, after unmixing, the error will be distributed over all four macroblocks and no single block will experience a dramatic corruption.

Continuing with our example where block $\hat{C}$ is lost, let $\hat{C}'$ be the reference block that the estimated motion vector points to. We then reconstruct a representation of the 4 adjacent macroblocks as follows:

$$\begin{bmatrix} \hat{A} & \hat{B} \\ \hat{C}' & \hat{D} \end{bmatrix}.$$

We then invert the Hadamard transform to obtain an estimate of the four original macroblocks A, B, C, and D.

Note that we use the similar algorithm used by MPEG to recover lost motion vector, then simply inverted the Hadamard mixing. However, this simple algorithm for recovering motion vectors works better in combination with our Hadamard scheme than without it.

- If the motion vector we recover is exactly the same as the lost one, then the

error on this macroblock would only be the residual in the lost macroblocks. The mixing will evenly distribute this error across the four adjacent macroblocks, smoothing it and reducing the jarring effect. In fact, since residuals carry much less information than motion vectors, this noise is normally of small magnitude and hence its effect is fairly mild and similar to the noise in the toy example in Chapter 4.

- If the motion vector we estimated is different from the lost one, the error caused by this estimate will also be equally distributed to all four macroblocks. Yet the impact of this error will be structural. In Figure 7-2 we show an example where the lost motion vectors cannot be accurately estimated. The left image shows the original block we consider from the frame with the loss map shown in Figure 7-1. The right image is the reconstructed block of MPEG by only estimating the motion vectors for the lost macroblocks, without information unmixing. In the middle is the reconstructed block of ChitChat. We can see that by mixing the information, a much smoother decoded block is obtained, even though the motion estimation error concealment scheme is not accurate. In contrast, without mixing the information, merely estimating lost motion vectors leads to jarring visual effect which lacks consistency.

7.2 Out-of-Loop Smoothing

The last step at our decoder is an out-of-loop smoothing block in the pixel domain, which is commonly used by commercial softwares and highly recommended by MPEG-4 [58]. This is because certain artifacts might be seen on the decoded frames due to block-based transforms and quantization. H.264 has an intricate in-loop deblocking filter to deal with this problem [58]. However, in our design, to mitigate this phenomenon, we use a simple total variation based denoising method [25], which will originally run multiple iterations to achieve a desired degree of smoothness. Yet to ensure low complexity, we force the denoising to terminate after a maximum of 10 iterations. Moreover, since this smoothing block is out of loop (only exists at the

decoder), the only timeline we have to meet is the display time to the user. Thus, another option is to let the decoder keep iteratively smooth the frame until the frame's play time is due.

Chapter 8

Experiment Setup

8.1 Video Quality Metrics

Ideally, video quality is measured as the mean of the scores given by human judges who rate video call samples [8]. Human studies however need controlled environments and are quite expensive to run. Instead, tests of video quality typically use objective measures that can be computed automatically [9]. Among these, it is widely common to use the Peak Signal-to-Noise Ratio (PSNR) and video outages [40], which are the two metrics we employ in this study.

8.1.1 Peak Signal-to-Noise Ratio (PSNR)

The PSNR of a single frame is a function of the mean square error (MSE) between the reconstructed frame and its original version. Overall PSNR of a sequence of frames is the a function of the MSE between the reconstructed sequence and the original sequence.

$$MSE = \frac{1}{m \cdot n} \cdot \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

$$PSNR = 10 \cdot log_{10} \frac{(2^L - 1)^2}{MSE} [dB]$$

L is the number of bits used to encode each pixel luminance, typically 8 bits. m

is the number of columns and n is the number of rows in a frame. I is the reconstructed frame or sequence and K is the original version. A typically good PSNR is around 35 dB and it is generally agreed upon that PSNR values below 20 dB are unacceptable [71].

8.1.2 Video Outage

Surveys show that the vast majority of users identify video freezing and lag as the biggest annoyance in video conferencing, instead of less clear images [17]. Hence, video outage has been suggested as an alternative metric for video quality, particularly for video conferencing applications [40]. Further, it has been found that a frame rate of below 5 fps in a slow/medium video is noticeable by a normal user, and people will find video of a frame rate lower than 3 fps to be jerky [15]. Thus, we define a video outage to be an event where the video stalls for more than 1/3 of a second. The duration of an outage is the period of time that the video stalls.

8.1.3 Measuring Outages and PSNR

Measuring video outage and PSNR is complicated by the fact that we want to compare against Windows Live Messenger (WLM), which is a proprietary closed software. In particular, to measure video outage, we need to know which frames are sent by the sender and which frames are received by the receiver; to measure PSNR, we need to capture all the raw frames both at the sender and at the receiver and compare them for loss of quality. For WLM, however, we can only see the coded packets; but we do not have access to the pixel values. To work around this issue, we use the approach proposed in [56]. Specifically, we screen-capture [2] all the frames that are displayed on the screens both on the sender and the receiver's sides by WLM. We perform the screen capture 30 times per second. Since the highest frame rate of the original video is 15 fps [1], a capture sample rate of 30 fps is sufficient to record all the frames

[1] We feed a pre-recorded video of 15 fps to WLM to replace the webcam input. The encoder might choose to lower the frame rate to below 15 fps to match the available bandwidth. Yet the highest frame rate will not exceed 15 fps.

displayed on the screen. To make sure this screen capture process does not introduce any artifacts or noise, we tested it by playing multiple raw videos on the machines we used in the experiment and screen-capturing them. The screen-captured video output proved to be always the same as the displayed raw video input, pixel by pixel. Apart from this, to be completely fair to WLM, we perform the same screen capture on all our decoded videos using the other compared schemes, and use the screen-capture output for measurement.

Once we have the original encoded sent video and received decoded video, we need to accurately match the decoded frames to the encoded frames and the original frames, so that we can calculate the frame rate, video outage and PSNR of the decoded video. Again we use a trick proposed in [56] where we imprint an index number of the frame on a 16×16 area at the bottom-left corner of each frame [2].

8.2 Measured Internet Paths

The focus of our algorithm is to cope with packet loss in video chat. Packet loss generally happens in the following three situations. 1) Congestion in limited bandwidth environment. 2) Wireless loss. 3) Large variation in bandwidth. To evaluate our design under these circumstances, we conduct experiment over the following Internet paths:

- MIT office building and Residential area in *Cambridge, MA*: On this trace, the two locations are 0.4 miles from each other. One end is the high speed connection of an MIT office. The other end is a residential DSL wireless connection provided by Verizon, usually shared with two roommates. The average loss rate observed on this path was 1%.
- MIT office building and Residential area in *College Station, TX*: The two locations are approximately 1900 miles from each other. The first end is the same MIT office above, while the other end is a residential DSL connection provided

[2]Here we use 2 digit numbers, while in [56] the author used bar codes.

by Suddenlink, used exclusively by the experiment laptop. The average loss rate observed on this path was 4%.

- MIT graduate dorm and Tsinghua graduate dorm in *Beijing, China*: This is a cross-Pacific link with a distance of about 12000 miles. At the MIT side, high speed Ethernet connection with abundant bandwidth was used. At the Tsinghua graduate dorm in Beijing, wired connection was used. One thing to notice is that *Tsinghua University* is where you can find almost the highest Internet connection speed in China. The average loss rate observed on this path was 0.4%. [3]

The experiments are conducted over a period of 3 weeks, during which we collect a total of 5 hours of video data over each path. To limit the computational overhead, we sample the traces at intervals of 10 minutes and process each time a window of 1 minute.

8.3 Lab Testbed Setup

We also conducted similar experiment on a testbed we set up in the lab, in order to evaluate ChitChat in a wider range of controlled loss rates. In Figure 8-1 we show our lab testbed setup. We run video calls between **A** and **B**. Since **A** and **B** are both in the same building and both have Gigabit bandwidth, we use a **CLICK** router [47] to create a bandwidth bottleneck. Specifically, **A** and the **CLICK** router are in the same subnet. We set **A**'s gateway to be the **CLICK** router and force all the traffic going from/to **A** to travel through the **CLICK** router. Therefore, all the Internet traffic from/to **A** is the natural competing traffic for the video calls.

We set the bandwidth of the **CLICK** router to be 1Mbps and the queue size to be 40. This means with an average packet size of 700 bytes, when the queue is full,

[3]Typically consumer video calls between U.S. and China are of very low quality, due to the fact that the available bandwidth in China is normally very limited. 400kbps is considered a good downlink speed in Beijing, China. With congestion, video calls will be very often dropped. Therefore, we conducted this experiment at times when there was little competing traffic at the China side. As a result, we expect the loss we see here to be primarily Internet core losses.

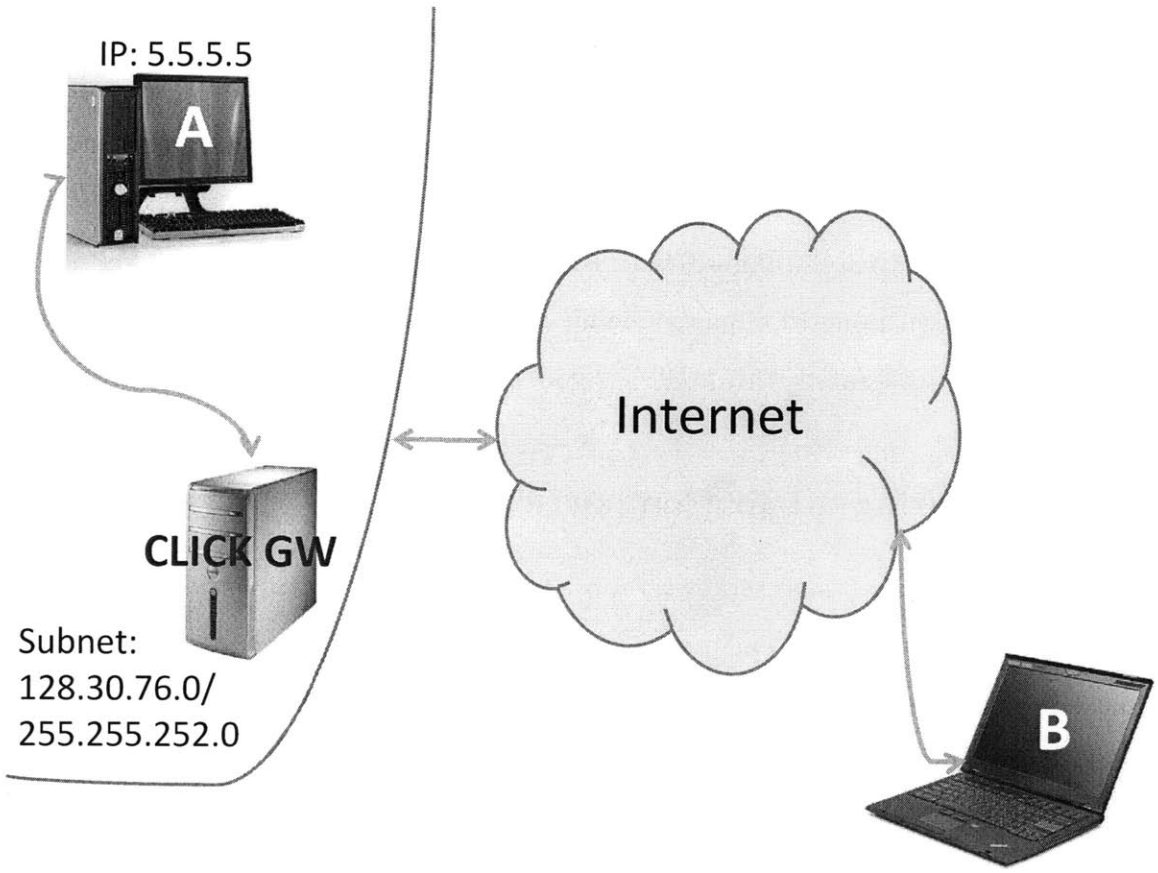


Figure 8-1: **Testbed Setup in the Lab.** We hide **A** behind a **CLICK** router and run Windows Live Messenger video calls between **A** and **B**. The competing traffic is all the Internet traffic going from/to **A**.

the router will discard any packets that arrived more than 200ms ago.

8.4 Compared Schemes

We compare the following four schemes.

- Windows Live Messenger 2009 [11]. Windows Live Messenger is an instant messaging client created by Microsoft and it has over 330 million active users by June 2009 [10]. Its free video call service sees more than 230 million video conversations each month, with an average length of 13.3 minutes, making it one of the largest consumer video telephony providers in the world [10].
- H.264 AVC/MPEG-4 Part 10. H.264 AVC/MPEG-4 Part 10 is a state-of-the-art

video codec standardized by ITU-T Video Coding Experts Group together with the ISO/IEC Moving Picture Experts Group (MPEG). It is widely adopted by applications like Blu-ray, Youtube and video conferencing softwares. We use the reference implementation available at [6].[4]

- H.264 AVC/MPEG-4 Part 10 with FEC. This scheme uses the same video codec as above, with one FEC packet being added to every encoded frame.
- ChitChat integrated with MPEG-2 codec [60] [5] but without any FEC.

8.4.1 Ensuring a Fair Comparison

In this work, our goal is to tackle the problem of robustness to packet loss without changing the details of the underlying video codec or other components of the video chat program. Therefore, to compare different schemes, we need to keep certain factors the same for all of them. This issue however is complicated by the fact Windows Live Messenger is a proprietary software, and hence we cannot control its parameters. Thus, to ensure fairness we force the other three schemes, as much as possible, to use the same parameters and setup as Windows Live Messenger (WLM).

Specifically, commercial video conferencing applications like Windows Live Messenger all have their own built-in rate control mechanisms [55]. The rate control algorithm estimates the available bandwidth and adapts the video frame rate accordingly (i.e., reduces the frame rate in time of congestion and increases it when spare bandwidth becomes available). Using a different rate control algorithm across the compared scheme will result in unfair comparison as the differences in video quality may be due to differences in the bandwidth usage and the frame rate enforced by rate control. However since Windows Live Messenger is not open source, we can neither change its rate control mechanism nor learn its detail to implement it for the other compared schemes.

[4]The exact command that we run is: time ffmpeg re pix_fmt gray force_fps g 12 i input.avi psnr vcodec libx264 vpre fast vpre baseline strict 1 r 15/1 force_fps qscale Q b B output.mp4.

[5]MPEG-2's compression is not as efficient as H.264 with videos of high motion. Yet the basic mechanisms are the same. Since in video calls, usually we observe only slow/medium motion, we expect the performance of MPEG-2 to be relatively similar to more advanced codecs.

Thus, to ensure that all schemes make the same decisions about their bandwidth usage and frame rate we run our experiments as follows. We run WLM video calls between various pairs of locations to collect trace data. We capture all the sent packets and received packets. In addition to muting the sound during the video call, we also classify the packets based on the payload. Since the payload sizes of audio packets and video packets are very distinct, we make sure all the packets we consider are dedicated to encoded video information.

To make sure that all schemes react to the rate adaptation decisions in the same way, we force the other three schemes to send the same number of packets at the same points in time using the same packet size, which ensures an identical usage of bandwidth. We also make all compared schemes encode and send the exact same frames during the experiment, which ensures the same frame rate for all schemes at any moment. With these two constraints, we guarantee all four compared schemes conform to the same rate control decisions, which are those taken by the Windows Live Messenger built-in rate adaptation mechanism.

Ideally we would like also to make all compared schemes use the same video codec. This is however not doable, as Windows Live Messenger employs the VC-1 codec [51], which is proprietary. Our scheme, on the other hand, uses the MPEG/H.264 codec. This, however, should not put Windows Messenger at a disadvantage. This is because, the VC-1 codec is known to generally have a better quality than H.264. Specifically, the viewers in a test conducted by DVD Forum rated the quality of VC-1 the best among VC-1, H.264, MPEG-4 Advanced Simple Profile and MPEG-2 [51].

8.5 Tested Videos

Instead of having our own video conference, which might depend on our environment and the humans conducting the call, we use standard reference videos available at Xiph.org. We experiment with the videos: *Deadline*, *Salesman*, and *Students*. We chose these videos because, among the available reference test video sequences, they best represent a video conferencing setting: A talking person/two talking people with

a relatively static background. We use a frame rate of 15 fps and crop the frames to 320×240, which are the default values for Skype and Windows Messenger. We take 20 seconds of the each video and repeat it to form a long sequence, which we then feed into Windows Live Messenger as a pre-recorded source replacing the webcam input [5].

Chapter 9

Evaluation

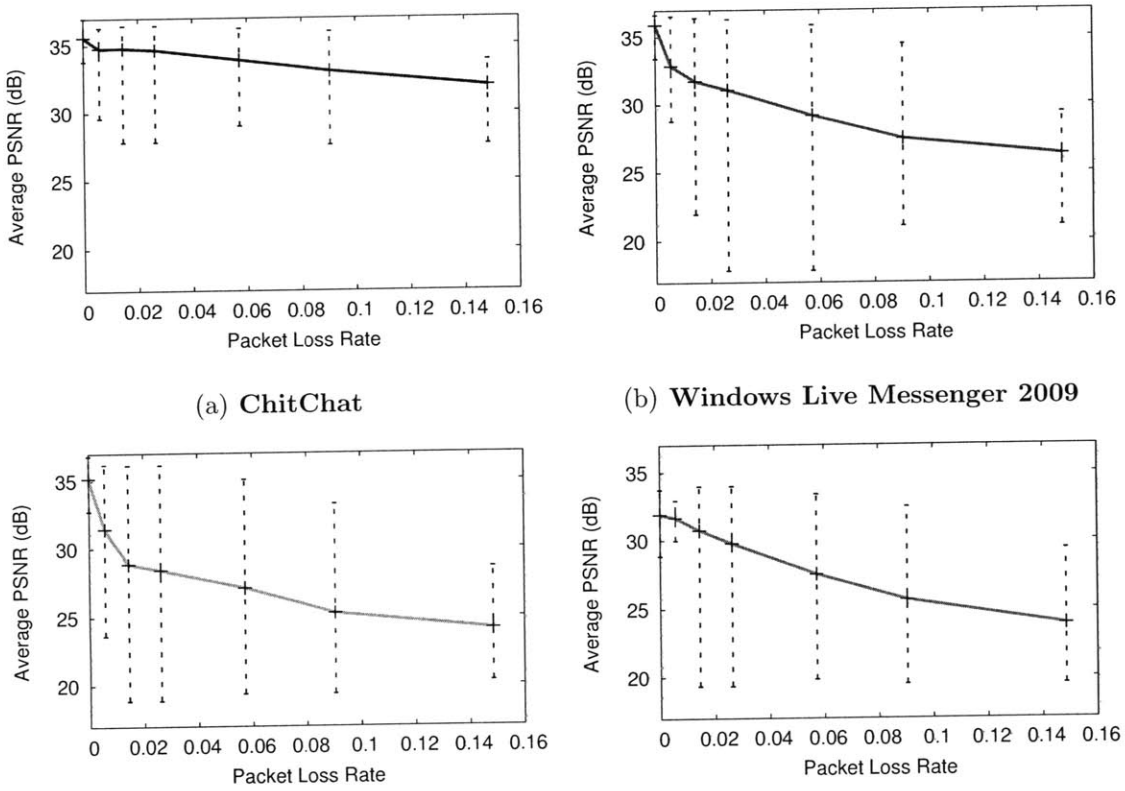
We compare ChitChat with Windows Live Messenger 2009, H.264 and H.264 with FEC.

9.1 Impact of Packet Loss on Video Quality

We would like to examine the PSNR of the decoded videos in presence of packet loss, for the four schemes. Thus, as recommended by ITU-T Video Coding Experts Group [9], we calculate the average PSNR over each 5-second interval, between all frames of the source sequence and the corresponding reconstructed frames. We look at the relationship between packet loss rate and average PSNR for these intervals. When a source frame is lost in the decoded sequence, video conferencing applications generally display the most recent usable frame. Therefore, the PSNR of this frame will be calculated between the source frame and the most recent usable frame. Average PSNR calculated this way reflects both the compressed video quality and the robustness of the system against packet loss since it introduces penalty for lost pictures.

(a) How Do the Four Schemes Compare?

Figure 9-1 plots the average PSNR at different loss rates for all experiments over the Internet paths described in 8.2. The results are computed over all the intervals for all traces. The figure shows the following:



(a) **ChitChat**

(b) **Windows Live Messenger 2009**

(c) **H.264**

(d) **H.264 with one FEC packet per frame**

Figure 9-1: **PSNR at different loss rate.** The figure shows that ChitChat's video quality is significantly higher than the compared schemes over the whole range of loss rates.

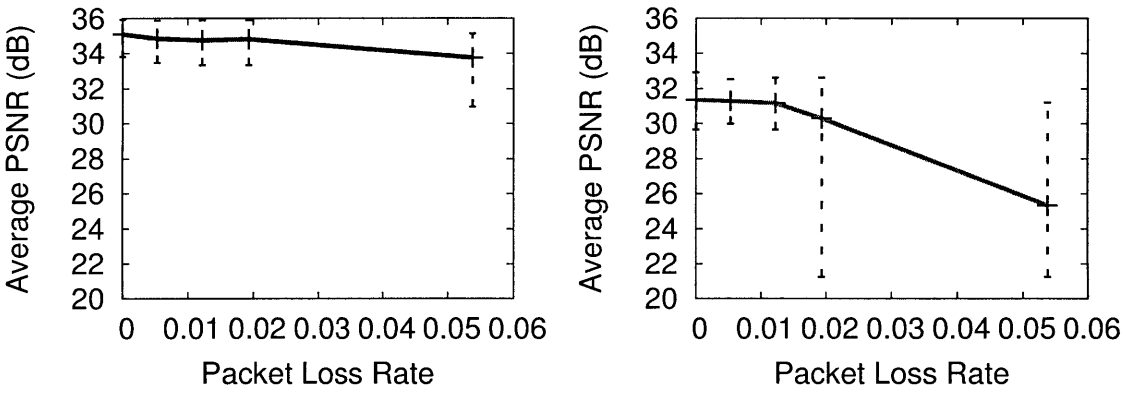
- When no loss occurs, the average PSNR of ChitChat is as good as WLM 2009 and H.264, which proves the mixing of information in ChitChat does not render the standard video compression less efficient. In contrast, at zero loss, the FEC-based approach has a PSNR 3 dB lower than the other schemes. This is because rate control forces all schemes to use exactly the same rate. Hence to add an FEC packet per frame, the codec needs to compress the video more (i.e., quantize more), resulting in lower quality even without loss.
- As the loss rate increases, ChitChat's average PSNR drops much slower than the other schemes. At a typical Internet loss rate of 1%, ChitChat has an average PSNR of 35 dB, 0.8 dB lower than the PSNR without loss. In contrast, at 1% loss rate, the PSNR of Windows Live Messenger and H.264 drops by about 3 dB and 5 dB respectively. The PSNR of the FEC scheme drops by 1.2 dB at this

loss rate; however this comes at the cost of reducing the PSNR in times of no losses. At a high loss rate of 15%, ChitChat still achieves an average PSNR of 32 dB while the other schemes' PSNR all drops to below 26 dB, which indicates a significant amount of dropped frames.

- On the same figure, the vertical bars show the maximum and minimum PSNR within each loss rate bucket for all four schemes. ChitChat has a much more consistent performance (smaller standard deviation) than the others.
- Finally, we note that the video quality of ChitChat, Windows Messenger and H.264 is the same at zero loss rate, and differs only in the presence of packet loss. This shows that ChitChat's gains are due to differences in reacting to packet loss rather than the differences in the codec. The FEC scheme uses H.264 and hence its reduced PSNR at zero loss rate is not due to the codec but rather to the overhead of FEC.

(b) Detailed Comparison with FEC

Aside from its excessive overhead when used in interactive applications, FEC also suffers from the well-known cliff effect, which is especially amplified in video applications. In particular, a scheme that adds one packet of redundancy to each frame can completely recover from one packet loss per frame. However, frames that have two or more lost packets will be as corrupted as if no FEC was used. In fact, even with an average loss rate as low as 1%, it is not uncommon for multiple packets of a frame to get lost from time to time. In particular, Figure 9-2 shows the PSNR degradation of ChitChat and H.264+FEC for the MIT-Cambridge DSL trace. This trace has an overall packet loss rate of 1%. We can see that the FEC scheme can maintain a relatively constant PSNR with small standard deviation at loss rates below 1%, yet as soon as the loss rate nears 2%, a wide range of PSNR is observed. In other words, even a 25% FEC redundancy does not prevent frame quality from worsening severely even with just 2% of packet loss. On the contrary, ChitChat's degradation is both much slower and much smoother. By mixing the information and letting each packet carry equally important information for the entire frame, we allow the receiver to recover



(a) **ChitChat** (b) **H.264+FEC**

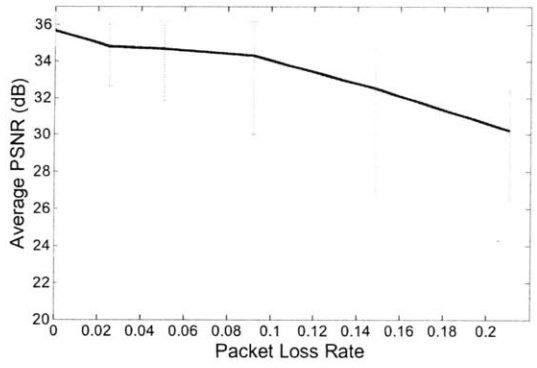
Figure 9-2: **With increased loss rate, FEC shows a cliff effect while ChitChat degrades smoothly.**

a smooth version of the frame with relatively high quality, without a cliff effect.

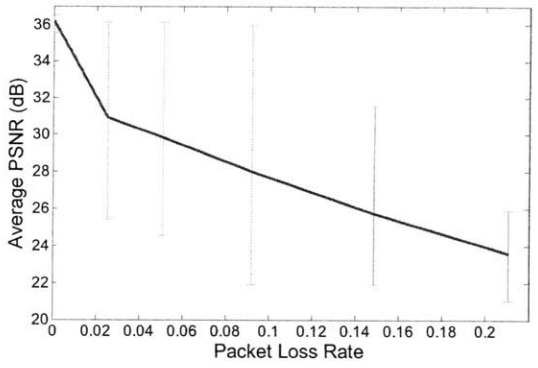
(c) Differences Across Tested Videos

Next we want to check that the behavior of ChitChat is consistent for any conference-style videos. Thus, in this section, we show the video quality of ChitChat and Windows Live Messenger for two different videos: *Salesman* and *Students*. In contrast to the previous figures which reported results from experiments over the Internet paths as described in 8.2, here we conduct the experiment on our testbed to ensure we can span a large range of loss rates and fix all parameters other than the tested video.

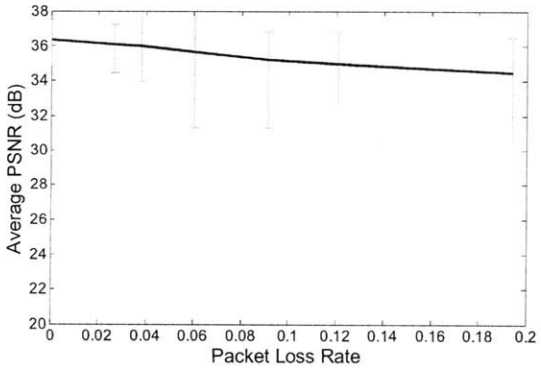
The *Students* video is easier to compress than *Deadline* and *Salesman*, because there is less motion in it than the other two. In Figure 9-3, the top row shows the average PSNR of the *Salesman* video, and the bottom row shows the average PSNR of the *Students* video. As the loss rate increases, the drops of PSNR for both ChitChat and WLM for *Students* are slower than those for *Salesman*. The explanation for this is that, the less motion in the video, the less the penalty in PSNR when frames are dropped due to packet loss. With both videos, ChitChat continues to outperform WLM at all loss rates. At a loss rate of 2%, the average PSNR of ChitChat is about 4 dB higher than WLM; at a loss rate of approximately 20%, the average PSNR of ChitChat is more than 7 dB higher than WLM.



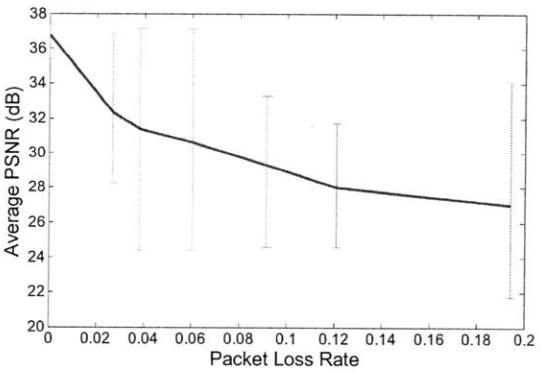
(a) **ChitChat** *Salesman*



(b) **Windows Live Messenger 2009** *Salesman*



(c) **ChitChat** *Students*



(d) **Windows Live Messenger 2009** *Students*

Figure 9-3: **PSNR at different loss rates of the testbed experiment.** The figure shows that ChitChat's video quality is significantly higher than that of WLM for the whole range of loss rates.

9.2 Video Outage

Here we would like to closely examine the frequency and duration of outages experienced by the four schemes. Recall that an outage is as a video stalls for more than 1/3 of a second, which is the threshold at which humans perceive the video to be jerky [15].

It is important to note that due to rate control, the sender might decide to encode at a lower rate than 3 fps, causing video stalls. Such outages are not due to packet loss; rather they are caused by the rate control algorithm detecting lack of available bandwidth. Also, they are the same for all compared schemes since by the design

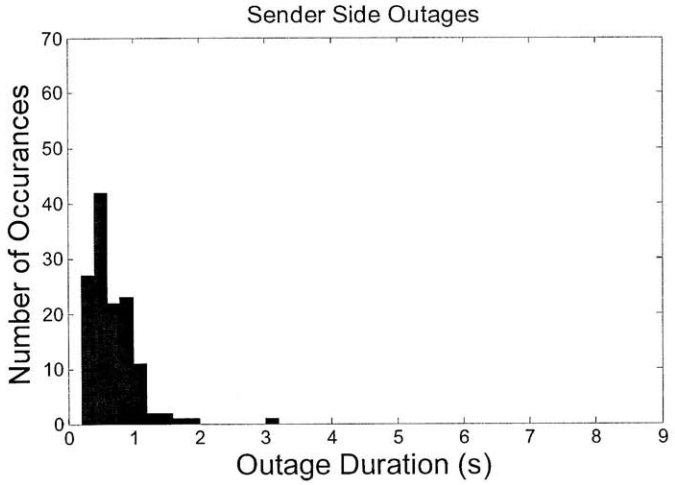
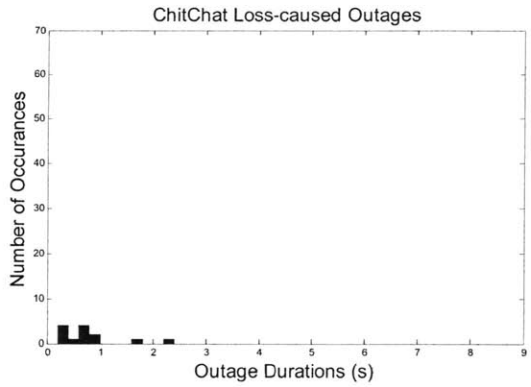


Figure 9-4: **A histogram of sender-side outages.** These outages are the result of the rate control algorithm detecting the lack of bandwidth and resorting to a very low frame rate.

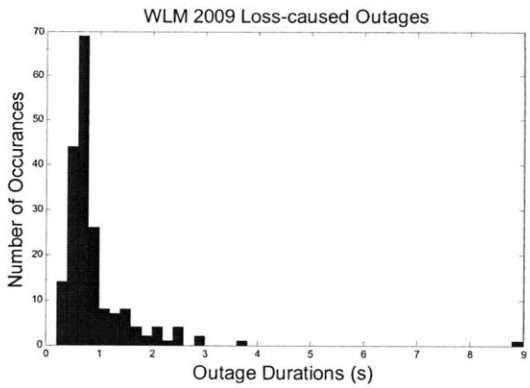
of our experiments (see §8.4.1) all schemes send the same number of packets at the same time instances, experience the same packet losses, and react using the same rate control algorithm. Thus, in our study, we distinguish the **sender side outages** due to rate control, from the **loss-caused** outages seen only at the receiver because of packet loss and the resulting frame loss.

Figure 9-4 shows a histogram of the outages at the sender side from all experiments done over Internet paths. Figure 9-5 shows the additional outages at the receiver, for all four compared schemes.The figures show that ChitChat dramatically reduces the occurrence of loss-caused outage events, and eliminates all long lasting outages. In comparison to Windows Live Messenger, ChitChat reduces the occurrence of outage events by 15x (from a total of 195 outages to only 13 outages). In comparison to H.264, ChitChat reduces the occurrence of outages by 11x (from 148 to 13 outages). In comparison to the FEC scheme, ChitChat reduces the occurrence of outages by 4x (from 53 to 13 outages).

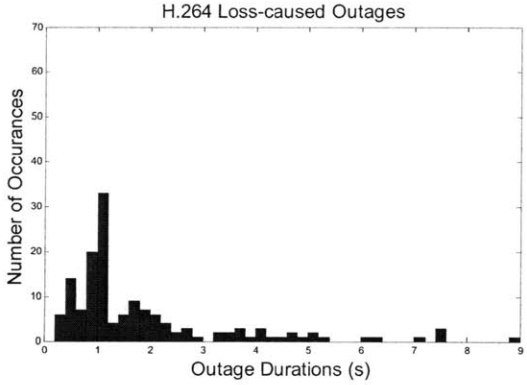
Interestingly, Windows Live Messenger has significantly more short outages than H.264 and the FEC scheme, but is more robust to long-lasting outages. One explanation for the smaller number of long outages is that WLM uses some acknowledgment



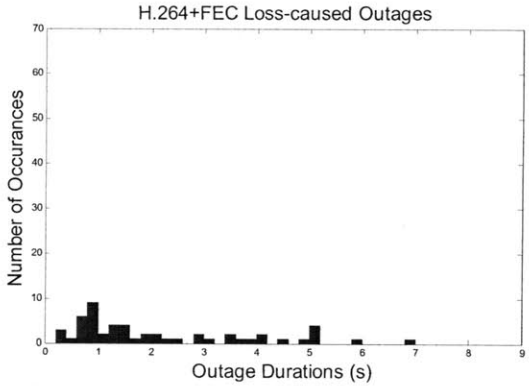
(a) **ChitChat's Loss-Caused Outages**



(b) **Windows Live Messenger's Loss-Caused Outages**



(c) **H.264's Loss-Caused Outages**



(d) **H.264+FEC's Loss-Caused Outages**

Figure 9-5: **Histograms of Loss-Caused Outage Duration**

mechanism that allows the sender to quickly discover an outage and stop using lost frames as reference frames.

9.3 ChitChat with ACK or NACK

Recall that in cases of long RTT, there is a trade-off between robustness and video size as discussed in 5.4.2. Using an ACK system guarantees robustness yet results in a larger size; using a NACK system does not introduce overhead in size but it suffers for one RTT when loss happens. Different chatting programs use different algorithms to choose between ACK and NACK, considering this trade-off based on the RTT and loss rate of the channel. ChitChat, however, can work with both ACK and NACK

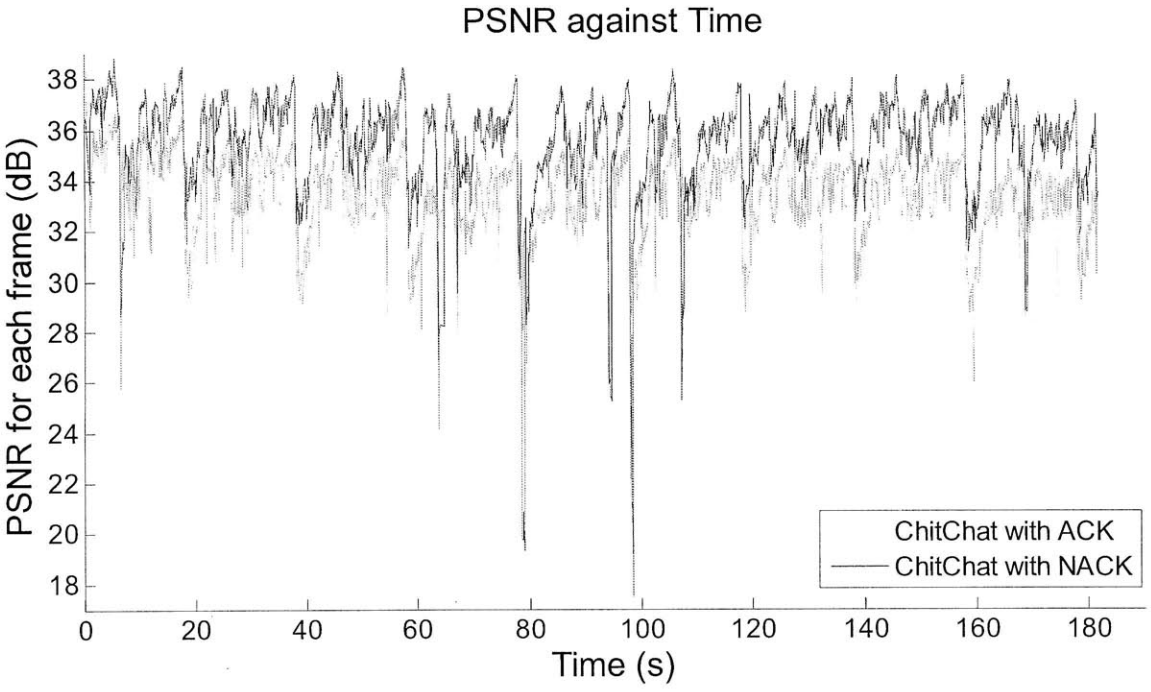
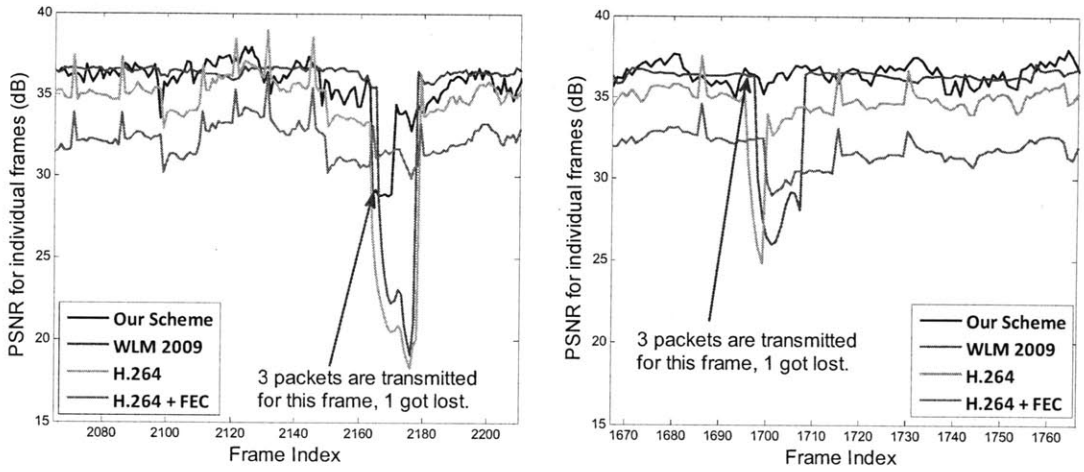


Figure 9-6: **PSNR of ChitChat with ACK and NACK on the U.S.-China Link.** The RTT on this link is about 300 ms. The blue data is ChitChat with NACK. The orange data is ChitChat with ACK.

systems and leave this decision to the codec.

For a U.S-Asia link, the RTT is normally around 300 ms, which is the duration of approximately 5 frames at a frame rate of 15 fps. Figure 9-6 plots the PSNR of each frame against time for part of the U.S.-China trace. The orange data is ChitChat with ACK and the blue data is ChitChat with NACK.

- In an ACK system, the loss-caused error will only affect the current frame and will not propagate to following frames. In the figure, any sudden drop of the orange line because of loss always immediately recovers on the next frame. The packet loss happening at 94.1s is a clear example. The PSNR for the ACK system goes back up immediately after the loss while the PSNR of the NACK system remains lower than normal for about 5 frames. This shows the advantage of robustness in an ACK mechanism. Note that a lot of the low PSNR for the ACK system is due to the inefficiency in compression caused by the long RTT,



(a) High Motion

(b) Low Motion

Figure 9-7: **ChitChat's Contribution to an NACK mechanism on traces with long RTT.** ChitChat reduces the impact of the drifting effect to a great extent and yields usable decoded frames of PSNR higher than 28 dB.

instead of packet loss.

- However, this robustness of an ACK system comes at a price of the increase in video size. We can see from the figure that this gap of 300 ms from the reference frame causes the PSNR of the ACK system to be 2-3 dB lower than the NACK system when the size of the encoded video remains the same.

Therefore with a relatively low loss rate, we consider a NACK mechanism to be much more reasonable for long RTT. Moreover, ChitChat's information mixing design can make a NACK system work better in terms of improving video quality and reducing video outages. When a NACK mechanism is adopted, the drifting between the encoder and the decoder will not last for longer than one RTT before they are resynchronized. Even in the extreme case of U.S.- Asia, this only affects 5 frames. The quality of these 5 consecutive frames all depend on how well the system can recover the first frame suffering from loss. With ChitChat, if at least one packet for the frame is received, we can very often reconstruct a usable copy of the frame of satisfying quality. Therefore the later 4 frames will not be dropped either.

Figure 9-7 zooms in to two examples on the U.S-China trace where one out of the three packets transmitted for a frame is lost. The first loss happens at a part of

the video where the motion is high and the second loss happens at a part where the motion is low. We plot PSNR for individual frames against time. In the first example (left), PSNR of both WLM 2009 and H.264 dropped quickly to below 20dB when the loss happened, indicating the motion in that part of the video is medium/high. In this case, ChitChat successfully reconstructed a version of the first frame affected of 29 dB PSNR, a quality degradation readily accepted by an average user. In the second example (right), from the PSNR drop of WLM 2009 and H.264, we can see this part of the video has low motion, in which case ChitChat was able to reconstruct a very good version of the original frame. Please note that the drop of PSNR for H.264+FEC is not due to the packet loss; it is due to the redundancy FEC introduces. In conclusion, as long as not all packets for a frame are lost, ChitChat can lessen the degradation of video quality during the period of the drifting effect in a NACK system, in cases of both high motion and low motion. In the case where all the packets for one frame are lost, ChitChat with a NACK mechanism will see an outage of one RTT.

9.4 ChitChat's Additional Error Concealment Gain

In 7.1.2, we explained how ChitChat's error concealment works more than simply estimating the lost motion vectors. Here we want to restress the gain in error concealment of ChitChat. If the motion vector we recover is exactly the same as the lost one, the error on this macroblock would only be the residual in the lost macroblocks. The mixing will evenly distribute this error across the four adjacent macroblocks, smoothing it and reducing the jarring effect. In fact, since residuals carry much less information than motion vectors, this noise is normally of small magnitude and hence its effect is fairly mild and similar to the noise in the toy example in Chapter 4.

The more interesting case is where the motion vector we estimated is different from the lost one. In this case, the error caused by this estimate in ChitChat will also be equally distributed to all four macroblocks, while jarring structural noise will be observed without mixing. Figure 9-8 shows an example where the lost motion vectors cannot be accurately estimated. The left image shows the original block we

Figure 9-8: **Error Concealment when the Estimated Lost Motion Vectors are not Accurate.** (a) is the original image. (b) is the reconstructed version with ChitChat's error concealment. (c) is the reconstructed version with MPEG's error concealment only.

consider from the frame with the loss map shown in Figure 7-1. The right image is the reconstructed block of MPEG by only estimating the motion vectors for the lost macroblocks, without information unmixing. In the middle is the reconstructed block of ChitChat. By mixing the information, a much smoother decoded block is obtained, even though the motion estimation error concealment scheme is not accurate. In contrast, without mixing the information, merely estimating lost motion vectors could lead to very disturbing jarring visual effects.

Chapter 10

Conclusion

We present a new approach for improving robust video transmission over the Internet. We design a practical system, ChitChat, based on the idea of information mixing and ensuring the information in each packet describes the whole frame. Experiments are conducted on both intercontinental Internet paths and links within the U.S. as well as on a lab testbed. Our evaluation shows that ChitChat significantly reduces the occurrences of video stalls due to packet loss and improves the overall video quality. By providing a satisfying user experience even in presence of packet loss, this approach can contribute to making video chat reach its full potential.

Bibliography

[1] Akamai. http://www.akamai.com.

[2] Camtasia Studio. http://www.techsmith.com/.

[3] Delivering Video Quality in Your IPTV Deployment.

[4] FaceTime. http://www.apple.com/iphone/features/facetime.html.

[5] Fake Webcam. http://www.fakewebcam.com/.

[6] FFMPEG. http://www.ffmpeg.org/.

[7] Google Talk. http://www.google.com/talk/.

[8] ITU-R BT.500-11 Recommendation: Methodology for the Subjective Assessment of the Quality of Television Pictures.

[9] ITU-T Rec. J.247 (08/08) Objective Perceptual Multimedia Video Quality Measurement in the Presence of a Full Reference.

[10] MessengerSays Blog. http://messengersays.spaces.live.com/.

[11] Windows Live Messenger. http://messenger-msn-live.com.

[12] Supporting Real-time Traffic: Preparing Your IP Network for Video Conferencing, 2006.

[13] The Internet Singularity, Delayed: Why Limits in Internet Capacity Will Stifle Innovation on the Web, 2007. http://www.nemertes.com/press_releases/user_demand_internet_could_outpace_network_capacity_2010.

[14] Skype Fast Facts Q4 2008, 2008.

[15] Quality of Experience for Mobile Video Users, 2009.

[16] AT&T High Speed Internet DSL Plans, 2010. `http://www.att.com/gen/general?pid=10891`.

[17] Video Conferencing Is Here. Are You Ready or Falling Behind. Technical report, Global IP Solutions, 2010.

[18] Toufik Ahmed, Ahmed Mehaoua, Raouf Boutaba, and Youssef Iraqi. Video Streaming with Fine-grained TCP-friendly Rate Adaptation. In *in Lecture Notes in Computer Science*, pages 18–31. Springer-Verlag.

[19] Soroush Akhlaghi, Amir K. Kh, and Abolfazl Falahati. Reducing The Effect of Channel Time Variations in MIMO Broadcast Systems, 2006.

[20] John Apostolopoulos. Reliable Video Communication over Lossy Packet Networks Using Multiple State Encoding and Path Diversity. In *Visual Communications and Image Processing (VCIP*, 2001.

[21] John G. Apostolopoulos and Mitchell D. Trott. Path Diversity for Enhanced Media Streaming. *IEEE Communications Magazine*, 42:80–87, 2004.

[22] Salman A. Baset and Henning Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. 2004.

[23] Ali C. Begen and Yucel Altunbasak. Redundancy-Controllable Adaptive Retransmission Timeout Estimation for Packet Video. In *NOSSDAV '06: Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*, pages 1–7, New York, NY, USA, 2006. ACM.

[24] Dario Bonfiglio, Marco Mellia, Michela Meo, and Dario Rossi. Detailed Analysis of Skype Traffic. In *IEEE Transaction on Multimedia*, 2009.

[25] X. Bresson and T. F. Chan. Fast Dual Minimization of the Vectorial Total Variation Norm and Applications to Color Image Processing, 2008.

[26] Indepth: Packet Loss Burstiness. http://www.voiptroubleshooter.com/indepth/burstloss.html.

[27] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: High-bandwidth Content Distribution in Cooperative Environments. 2003.

[28] Kuan-Ta Chen, Chun-Ying Huang, Polly Huang, and Chin-Laung Lei. Quantifying Skype User Satisfaction. *SIGCOMM Comput. Commun. Rev.*, 36(4):399–410, 2006.

[29] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. Skype Video Responsiveness to Bandwidth Variations. In *NOSSDAV*, 2008.

[30] M. Dreese. A Quick Introduction to DivX Recording, 2003.

[31] Jake Dunagan and Mike Liebhold. The Future of Real-Time Video Communication. Technical report, The Institute for the FutureIFTF, 2009.

[32] Martin Ellis and Colin Perkins. Packet Loss Characteristics of IPTV-like Traffic on Residential Links. In *7th Annual IEEE Consumer Communications & Networking Conference, Workshop on Emerging Internet Video Technologies*, 2010.

[33] Allen Gersho and Robert M. Gray. Vector Quantization and Signal Compression, 1991.

[34] Ladan Gharai and Tom Lehman. Experiences with High Definition Interactive Video Conferencing. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 433–436, 2006.

[35] Vivek K Goyal. Multiple Description Coding: Compression Meets the Network, 2001.

[36] Saikat Guha, Neil Daswani, and Ravi Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System, 2006.

[37] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification, 2003.

[38] Mohamed Hefeeda, Ahsan Habib, Boyan Botev, Dongyan Xu, and Bharat Bhargava. PROMISE: Peer-to-Peer Media Streaming Using Collectcast, 2003.

[39] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, and Keith W. Ross. A Measurement Study of a Large-Scale P2P IPTV System, 2007.

[40] Sheila S. Hemami. Robust Video Coding - An Overview, 2007.

[41] Fen Hou, Pin-Han Ho, and Xuemin Shen. A Novel Differentiated Retransmission Scheme for MPEG Video Streaming Over Wireless Links. *Int. J. Wire. Mob. Comput.*, 1(3/4):260–267, 2006.

[42] Szymon Jakubczak, Hariharan Rahul, and Dina Katabi. One-Size-Fits-All Wireless Video. In *ACM SIGCOMM HotNets*, 2009.

[43] Jeremy Johnson. In Search of the Optimal Walsh-Hadamard Transform. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 3347–3350, 2000.

[44] Dan Jurca, Dan Jurca, Pascal Frossard, and Pascal Frossard. Packet Selection and Scheduling for Multipath Video Streaming. *IEEE Transactions on Multimedia*, 9, 2006.

[45] Mohammad Ali Khojastepour and Ashutosh Sabharwal. Delay-Constrained Scheduling: Power Efficiency, Filter Design, and Bounds. In *IEEE INFOCOM, Hong Kong*, pages 7–11, 2004.

[46] Chang-Su Kim and Sang-Uk Lee. Multiple Description Coding of Motion Fields for Robust Video Transmission, 2001.

[47] Eddie Kohler. The Click Modular Router, 2001.

[48] Eddie Kohler, Mark Handley, and Sally Floyd. Designing DCCP: Congestion Control Without Reliability, 2003.

[49] Chris Lewis and Steve Pickavance. Implementing Quality of Service Over Cisco MPLS VPNs, 2006.

[50] Zhengye Liu, Yanming Shen, Shivendra S. Panwar, Keith W. Ross, and Yao Wang. Using Layered Video to Provide Incentives in P2P Live Streaming. In *P2P-TV '07: Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV*, pages 311–316, New York, NY, USA, 2007. ACM.

[51] Jay Loomis and Mike Wasson. VC-1 Technical Overview, 2007.

[52] Michael G. Luby and Jerry W. Miller. The Impact of Packet Loss on an IPTV Network, 2007.

[53] Janio M. Monteiro, Ricardo N. Vaz, Antonio M. Grilo, and Mario S. Nunes. Rate Adaptation for Wireless Video Streaming Based on Error Statistics. *Int. J. Wire. Mob. Comput.*, 2(2/3):150–158, 2007.

[54] Thinh Nguyen and Avideh Zakhor. Distributed Video Streaming with Forward Error Correction, 2002.

[55] Omer Boyaci, Andrea Forte, and Henning Schulzrinne. Performance of Video Chat Applications Under Congestion. In *International Symposium on Multimedia*, December 2009.

[56] Omer Boyaci, Andrea Forte, Salman Abdul Baset, and Henning Schulzrinne. vDelay: A Tool to Measure Capture-to-Display Latency and Frame-rate. In *International Symposium on Multimedia*, December 2009.

[57] Rohit Puri and Kannan Ramchandram. Multiple Description Source Coding Using Forward Error Correction Codes, 1999.

[58] Gulistan Raja and Muhammad Javed Mirza. In-loop Deblocking Filter for JVT H.264/AVC. In *ISPRA'06: Proceedings of the 5th WSEAS International Conference on Signal Processing, Robotics and Automation*, pages 235–240, Stevens

Point, Wisconsin, USA, 2006. World Scientific and Engineering Academy and Society (WSEAS).

[59] Amy R. Reibman and David Poole. Characterizing Packet Loss Impairments in Compressed Video. In *IEEE Int. Conf. Image Proc*, 2007.

[60] Iain E Richardson. H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia, 2003.

[61] Skype. `http://www.skype.com`.

[62] Stephan Wenger. H.264/AVC Over IP. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 13, July 2003.

[63] Thomas Stockhammer, Miska M. Hannuksela, and Thomas Wiegand. H.264/AVC in Wireless Environments. *IEEE Transactions on Circuits and Systems for Video Technology*, 13:657–673, 2003.

[64] Vinay A. Vaishampayan and N. J. A. Sloane. Multiple Description Vector Quantization with Lattice Codebooks: Design and Analysis. *IEEE Trans. Inform. Theory*, 47:1718–1734, 2001.

[65] D. Wang, N. Canagarajah, D. Redmill, and D. Bull. Multiple Description Video Coding Based on Zero Padding, 2004.

[66] Y. Angela Wang, Cheng Huang, Jin Li, and Keith W. Ross. Queen: Estimating Packet Loss Rate Between Arbitrary Internet Hosts. In *PAM '09: Proceedings of the 10th International Conference on Passive and Active Network Measurement*, pages 57–66, Berlin, Heidelberg, 2009. Springer-Verlag.

[67] Yao Wang, Michael T. Orchard, Vinay Vaishampayan, and Amy R. Reibman. Multiple Description Coding Using Pairwise Correlating Transforms. *IEEE Trans. Image Processing*, 10:351–366, 1999.

[68] Yao Wang, Shivendra Panwar, Shunan Lin, and Shiwen Mao. Wireless Video Transport Using Path Diversity: Multiple Description vs. Layered Coding. In *Image Processing Proceedings*, pages 21–24, 2002.

[69] Huahui Wu, Mark Claypool, and Robert Kinicki. Adjusting Forward Error Correction with Temporal Scaling for TCP-friendly Streaming MPEG. Technical report, ACM TOMCCAP, 2005.

[70] Guanjun Zhang and Robert L. Stevenson. Efficient Error Recovery for Multiple Description Video Coding, 2004.

[71] Hong Zhao, Yun Q. Shi, and Nirwan Ansari. Hiding Data in Multimedia Streaming over Networks. In *the 8th Annual Communication Networks and Services Research Conference*, 2010.