MIT-CSAIL-TR-2010-031 July 5, 2010

ChitChat: Making Video Chat Robust to Packet Loss

Jue Wang and Dina Katabi

ChitChat: Making Video Chat Robust to Packet Loss

Paper 1569335057
Jue Wang and Dina Katabi
MIT CSAIL

Abstract

Video chat is increasingly popular among Internet users. 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.

This paper presents 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.

1 Introduction

Video chat is increasingly used for both personal and business communications [25]. Measurements show a fast growth in video conferencing over Skype, Windows Live Messenger, Google Talk, etc. [10]; and the trend is likely to become stronger with the introduction of Apple FaceTime over mobile phones. 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 freeze and bad quality [13, 28]. Without better mechanisms for handling packet loss, video chat will not reach its full potential.

While much of the literature has studied the problem of reliable video delivery, it often does not distinguish video chat from live video streaming [33, 20, 54, 44, 16]. 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 [31], the ITU G.114 standard recommends a maximum delay of 150 ms for video calls in order to ensure lip synchronization [39]. Meeting this timing constraint is challenging given today's processing [46] and propagation delays. Hence any additional latency to tackle transient packet losses is highly undesirable.
- 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 [12], making the uplink a likely bottleneck for video chat applications.

As a result of these characteristics, solutions for packet loss proposed in the context of streaming applications [33, 20, 54, 44, 16] are mostly ineffective for chatting applications.

(a) Forward error correcting codes (FEC): FEC codes are inefficient for dealing with transient packet loss in video calls. Typical Internet loss rate is less than 1% [53]. 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 [24].[1] 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% [45] to 50% [24].

(b) Path Diversity: Proposals for routing packets over multiple paths using an overlay network [1, 17, 16, 36], 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 [45], this approach too is not effective for these applications.

(c) Retransmissions: Commercial video chat softwares try to avoid packet retransmissions [19] because such a mecha-

[1] 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 [45, 24].

nism requires delaying the received frames and hence interacts badly with the tight timing constraints of video calls.

This paper introduces 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. 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 improves the perceived video quality.

To achieve our objective of having each packet describe the whole frame, we mix the information in a frame before presenting it to the video codec. However, naively mixing the information in each frame destroys the codec's ability to recognize objects as they move across frames, and use this information to compress the video. 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 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 [9]. Our results show:

- In comparison to Windows Live Messenger, ChitChat reduces the number of 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.

2 RELATED WORK

Past work has recognized the negative impact of packet loss on video applications [23, 50, 48, 51] and proposed techniques for dealing with the problem [18, 44, 52, 40]. These proposals can be divided into two categories: solutions for Internet-core losses, and solutions for last-hop losses.

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 [16], the authors use multiple description coding, and send different descriptions over different paths to the receiver. SplitStream [22] employs multiple distribution trees and splits the load among the nodes while taking into account heterogeneous bandwidth resources. PROMISE [30] 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.

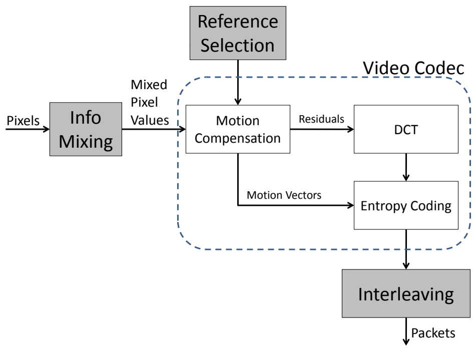


Figure 1—A Block Diagram of ChitChat's Architecture. The grey boxes refer to ChitChat's components. The white boxes are typical components of today's codecs.

Past solutions for losses on the access links are mainly limited to retransmission or forward error correction. Some designs buffer corrupted frames and ask for retransmission of lost packets [33, 20]. Others add forward error correction at the sender to allow the receiver to recover from losses without retransmissions [54, 44]. However, as explained in §1, FEC is inefficient for today's Internet where the loss rate is usually below 1% [26, 53], 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%.

ChitChat's information mixing algorithm is motivated by SoftCast, a recent proposal for wireless video [34]. 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 with existing video codecs and physical layers.

Also related to our work are mechanisms for media rate control, such as TFRC [29], DCCP [38], and others [37, 14, 43]. 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.

3 CHITCHAT AT A HIGH LEVEL

ChitChat is a video-chat centric approach to deal with transient packet loss. Its design ensures that: 1) received frames can be decoded immediately and displayed, 2) no extra bandwidth is required, and 3) the approach works with both edge and core losses.

ChitChat's architecture, illustrated in Figure 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.

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 [35]. Hadamard coding has been used in multiple prior systems to smooth out the impact of channel errors. For example, both MIMO systems [15] and SoftCast [34] 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 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.

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:

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

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

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. 1) by applying Hadamard, we will be able to distribute the channel noise evenly on the pixels we combine together; 2) 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; 3) 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 won't 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.

4.2 Hadamard Mixing in our Design

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 reliable delivery in the face of losses. Removing the DC value before coding images or frames is typical in the literature [27] 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 using the Hadamard matrix. In particular, consider 4 adjacent macroblocks, A, B, C and D. We represent these macroblocks as follows:

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

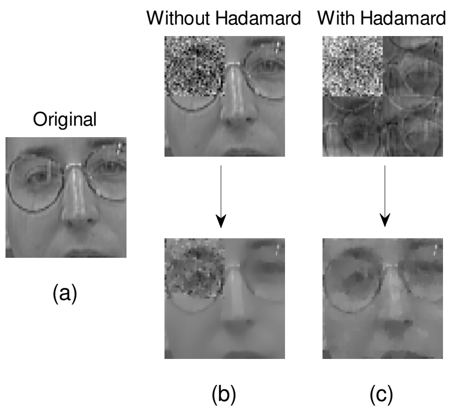


Figure 2—Hadamard mixing spreads the error and produces less jarring frames. (a) shows the original image (b) shows the image after adding noise to a macroblock and denoising the resulting erroneous block, (c) shows a Hadamard-mixed version of the image after adding noise, unmixing, and then denoising. The figures shows that Hadamard mixing spreads the noise, allowing the receiver to easily denoise the image.

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. Thus, when a packet is lost for a frame, we lose the corresponding differences, which leads to some noisy macroblocks. To understand the effect of Hadamard mixing on these losses, let us consider the toy example in Fig 2. Specifically, we take an image, corrupt one of its macroblocks, and observe the impact of corruption, with and without Hadamard mixing. Figure 2(a) shows the original image; whereas the top row in Figures 2(b) and 2(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 [47]. 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 2(b) and 2(c). As can be seen from the figures, the errors with Hadamard-mixing look less jarring, which shows the benefit of applying Hadamard.

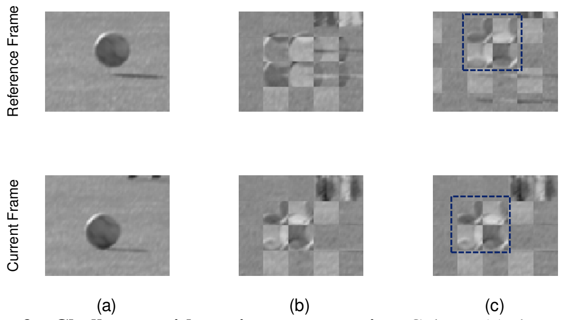


Figure 3—Challenges with motion compensation. Column (a) shows two consecutive frames in the pixel domain. The ball has moved from one frame to the next. Column (b) shows the same two frames after Hadamard transform. One cannot see an object that moved across the two frames though the original frames 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.

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 our evaluation, the general effect of Hadamard continues to spread the noise and deliver a better video quality.

5 Mixing-Aware Reference Selection

Video codecs (e.g., MPEG4/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 [49]. 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.[2]

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 3 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

[2]The typical block size for motion compensation is 16×16 pixels. MPEG-4 however has the option of using different block sizes depending on the level of motion in the video. In a video conferencing applications, the background is 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.

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 3 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 3(c) shows an alternative Hadamard transform of the original frames in Figure 3(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 computations 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.

5.1 Algorithm

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

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

$$A = F(x_0 : x_0 + 15, y_0 : y_0 + 15)$$
$$B = F(x_0 + 16 : x_0 + 31, y_0 : y_0 + 15)$$
$$C = F(x_0 : x_0 + 15, y_0 + 16 : y_0 + 31)$$
$$D = F(x_0 + 16 : x_0 + 31, y_0 + 16 : y_0 + 31)$$

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.

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 macroblock 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 axillary 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.

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 [49].

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 [49]. With ChitChat, these steps proceed without modification.

5.2 Intra-coded Frames

Video codecs occasionally send frames that are coded independently without any reference frames. These frames

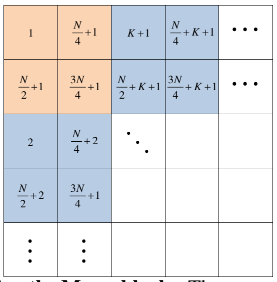


Figure 4—Interleaving the Macroblocks. The macroblocks are interleaved to ensure that adjacent macroblocks will not be transmitted in the same packet.

are typically called I-frame. 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 [42]. 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.

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 4. 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 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 $\frac{N}{4}$. 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.

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.

7 Video Recovery at the Receiver

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 frame to the user.

Third, it's uncommon for all the packets for a frame to be lost, based on our experiment and other literature [5]. Knowing the interleaving matrix, the receiver can map the lost packets into lost macroblocks. 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 packet. Thus, to recover a lost macroblock we need to estimate its motion vector and the residuals.

As in today's codecs, we estimate a lost motion vector using the motion vectors of nearby blocks [49]. 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 $\hat{C}$.

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 three original macroblocks A, B, C, and D.

Note that we use the same 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. Similarly 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.

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 [47]. 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 [47]. However, in our design, to mitigate this phenomenon, we use a simple total variation based denoising method [21], which will originally run multiple iterations to achieve a desired degree of smoothness. Yet to ensure low complexity, we force the denoising to stop 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.

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 [6]. 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 [7]. Among these it is widely common to use the Peak Signal-to-Noise Ratio (*PSNR*) and video outages, 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 [55].

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 [13]. Hence, video outage has been suggested as an alternative metric for video quality, particularly for video conferencing applications [32]. 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 [11]. Thus, we define a video outage to be an event where the video stalls for more than $1/3$ 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 [46]. 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, a capture sample rate of 30 fps is sufficient to record all the frames 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 [46] where we imprint an index number of the frame on a 16×16 area at the bottom-left corner of each frame.

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 at same MIT office above, while the other end is a residential DSL connection provided 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 %.

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 Compared Schemes

We compare the following four schemes.

- Windows Live Messenger 2009. Windows Live Messenger is an instant messaging client created by Microsoft and it has over 330 million active users by June 2009 [8]. 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 [8].
- 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 [4].[3]
- 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 the same H.264 codec above but without any FEC.

[3]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.

8.3.1 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 [45]. 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.

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, then 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 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 [41], which is proprietary. Our scheme, on the other hand, uses the 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 [41].

8.4 Tested Video

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 *Susan*. We chose these videos because, among the available reference test video sequences, they best represent a video conferencing setting: A talking person 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 WLM as a pre-recorded source replacing the webcam input [3].

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 [7], 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 5 plots the average PSNR at different loss rates. The results are computed over all the intervals for all traces. The figure shows the following:

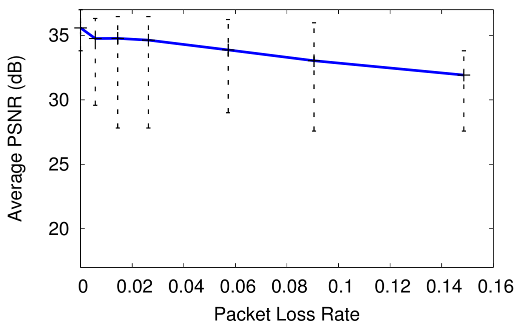
- When no loss occurs, the average PSNR of ChitChat is as good as WLM 2009 and H.264, which proves the mixing of information in ChitChat does not render the standard video compression less efficient. In contrast, at zero losses, the FEC-based approach has a PSNR 3 dB lower than the other schemes. This 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 than 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 6 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 smooth version of the frame with relatively high quality, without a cliff effect.

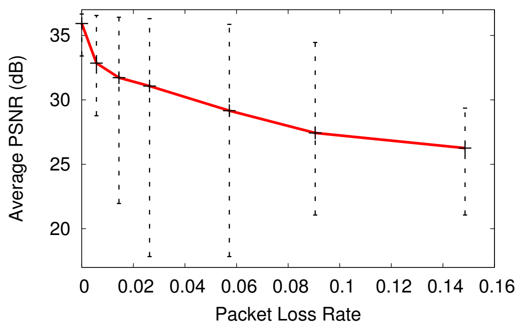
9.2 Video Outage

We would like to 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 [11].

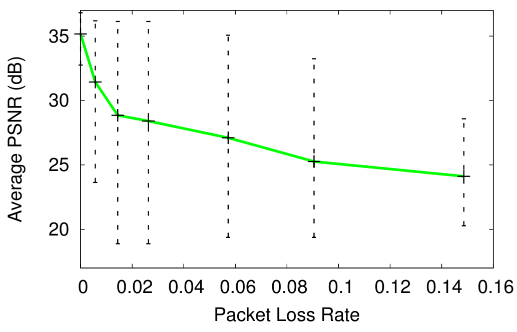
It is important to note that due to rate control, the sender might decide to encode at a lower rate than 3 fps, causing video stalls. Such outages are not due to packet loss; rather they are caused by the rate control algorithm detecting lack of



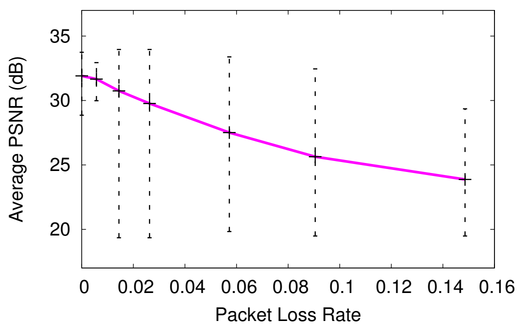
(a) **ChitChat**



(b) **Windows Live Messenger 2009**

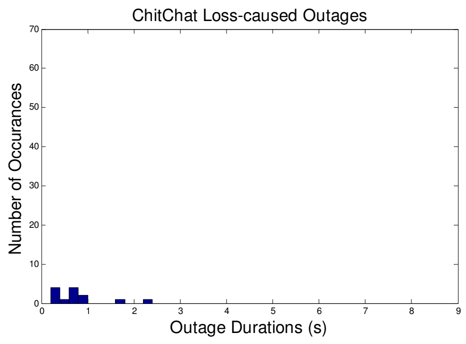


(c) **H.264**

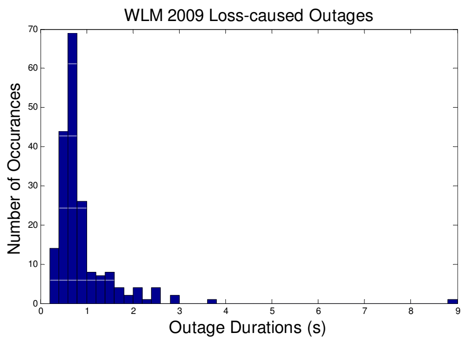


(d) **H.264 with one FEC packet per frame**

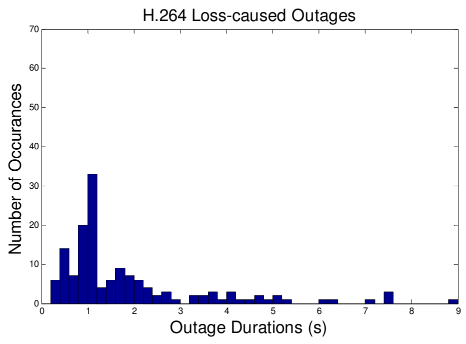
Figure 5—PSNR at different loss rate. The figure shows that ChitChat's video quality is significantly higher than the compared schemes over the whole range of loss rates.



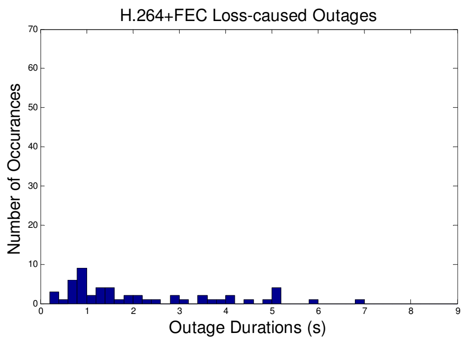
(a) **ChitChat's Loss-Caused Outages**



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



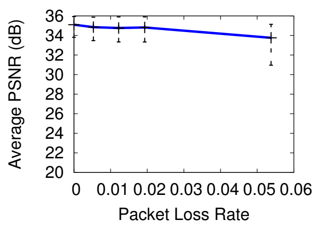
(c) **H.264's Loss-Caused Outages**



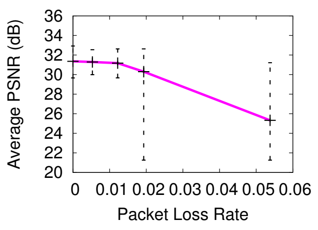
(d) **H.264+FEC's Loss-Caused Outages**

Figure 8—Histograms of Loss-Caused Outage Duration

available bandwidth. Also, they are the same for all compared schemes since by the design of our experiments (see §*8.3.1*) all schemes send the same number of packets at the same time instances, experience the same packet losses, and react



(a) **ChitChat**



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

Figure 6—With increased loss rate, FEC shows a cliff-effect while ChitChat degrades smoothly.

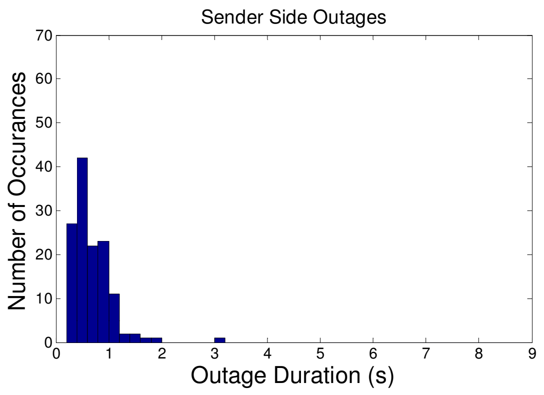


Figure 7—A histogram of sender-side outages. These outages are the result of the rate control algorithm detecting the lack of bandwidth and resorting to a very low frame rate.

using the same rate control algorithm.

Thus, in our study, we distinguish the sender-side outages due to rate control, from the additional outages seen only at the receiver because of packet loss and the resulting frame loss. Figure 7 shows a histogram of the outages at the sender side. Figure 8 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 mechanism that allows the sender to quickly discover an outage and stop using lost frames as reference frames. One also can explain the larger number of short outages as being due to excessive usage of motion compensation and coding later frames with respect to reference frames (i.e., interframe coding) as opposed to frequently sending frame that are independently coded without any reference frame (i.e., I-frames).

10 Conclusion

We present a new approach for improving robust video transmission over the Internet. It 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.

References

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

[2] Camtasia studio. `http://www.techsmith.com/`.

[3] Fake webcam. `http://www.fakewebcam.com/`.

[4] Ffmpeg. `http://www.ffmpeg.org/`.

[5] Indepth: Packet loss burstiness. `http://www.voiptroubleshooter.com/indepth/burstloss.html`.

[6] Itu-r bt.500-11 recommendation: Methodology for the subjective assessment of the quality of television pictures.

[7] Itu-t rec. j.247 (08/08) objective perceptual multimedia video quality measurement in the presence of a full reference.

[8] Messengersays blog. `http://messengersays.spaces.live.com/`.

[9] Windows live messenger. `http://messenger-msn-live.com`.

[10] Skype fast facts q4 2008, 2008.

[11] Quality of experience for mobile video users, 2009.

[12] At&t high speed internet dsl plans, 2010. `http://www.att.com/gen/general?pid=10891`.

[13] Video conferencing is here. Are you ready or falling behind. Technical report, Global IP Solutions, 2010.

[14] T. Ahmed, A. Mehaoua, R. Boutaba, and Y. Iraqi. video streaming with fine-grained tcp-friendly rate adaptation. In *in Lecture Notes in Computer Science*, pages 18–31. Springer-Verlag.

[15] S. Akhlaghi, A. K. Kh, and A. Falahati. Reducing the effect of channel time variations in mimo broadcast systems, 2006.

[16] J. Apostolopoulos. Reliable video communication over lossy packet networks using multiple state encoding and path diversity. In *Visual Communications and Image Processing (VCIP*, 2001.

[17] J. G. Apostolopoulos and M. D. Trott. Path diversity for enhanced media streaming. *IEEE Communications Magazine*, 42:80–87, 2004.

[18] J. G. Apostolopoulos and M. D. Trott. Path diversity for enhanced media streaming. *IEEE Communications Magazine*, 42:80–87, 2004.

[19] S. A. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. 2004.

[20] A. C. Begen and Y. 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.

[21] X. Bresson and T. F. Chan. Fast dual minimization of the vectorial total variation norm and applications to color image processing, 2008.

[22] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth

content distribution in cooperative environments. 2003.

[23] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei. Quantifying skype user satisfaction. *SIGCOMM Comput. Commun. Rev.*, 36(4):399–410, 2006.

[24] L. D. Cicco, S. Mascolo, and V. Palmisano. Skype video responsiveness to bandwidth variations. In *In NOSSDAV*, 2008.

[25] J. Dunagan and M. Liebhold. The future of real-time video communication. Technical report, The Institute for the FutureIFTF, 2009.

[26] M. Ellis and C. 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.

[27] A. Gersho and R. M. Gray. Vector quantization and signal compression, 1991.

[28] L. Gharai and T. Lehman. Experiences with high definition interactive video conferencing. In *in Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 433–436, 2006.

[29] M. Handley, S. Floyd, J. Padhye, and J. Widmer. Tcp friendly rate control (tfrc): Protocol specification, 2003.

[30] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. Promise: Peer-to-peer media streaming using collectcast, 2003.

[31] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. A measurement study of a large-scale p2p iptv system, 2007.

[32] S. S. Hemami. Robust video coding - an overview, 2007.

[33] F. Hou, P. Ho, and X. S. Shen. A novel differentiated retransmission scheme for mpeg video streaming over wireless links. *Int. J. Wire. Mob. Comput.*, 1(3/4):260–267, 2006.

[34] S. Jakubczak, H. Rahul, and D. Katabi. One-size-fits-all wireless video. In *ACM SIGCOMM HotNets*, 2009.

[35] J. 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.

[36] D. Jurca, D. Jurca, P. Frossard, and P. Frossard. Packet selection and scheduling for multipath video streaming. *IEEE Transactions on Multimedia*, 9, 2006.

[37] M. A. Khojastepour and A. Sabharwal. Delay-constrained scheduling: Power efficiency, filter design, and bounds. In *IEEE INFOCOM, Hong Kong*, pages 7–11, 2004.

[38] E. Kohler, M. Handley, and S. Floyd. Designing dccp: Congestion control without reliability, 2003.

[39] C. Lewis and S. Pickavance.

[40] Z. Liu, Y. Shen, S. S. Panwar, K. W. Ross, and Y. 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.

[41] J. Loomis and M. Wasson. Vc-1 technical overview, 2007.

[42] M.Dreese. A quick introduction to divx recording, 2003.

[43] J. M. Monteiro, R. N. Vaz, A. M. Grilo, and M. S. Nunes. Rate adaptation for wireless video streaming based on error statistics. *Int. J. Wire. Mob. Comput.*, 2(2/3):150–158, 2007.

[44] T. Nguyen and A. Zakhor. Distributed video streaming with forward error correction, 2002.

[45] Omer Boyaci, Andrea Forte, and Henning Schulzrinne. Performance of video chat applications under congestion. In *International Symposium on Multimedia*, December 2009.

[46] 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.

[47] G. Raja and M. J. 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).

[48] A. R. Reibman and D. Poole. Characterizing packet loss impairments in compressed video. In *IEEE Int. Conf. Image Proc*, 2007.

[49] I. E. Richardson. H.264 and mpeg-4 video compression: Video coding for next generation multimedia, 2003.

[50] Stephan Wenger. H.264/avc over ip. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 13, July 2003.

[51] T. Stockhammer, M. M. Hannuksela, and T. Wiegand. H.264/avc in wireless environments. *IEEE Transactions on Circuits and Systems for Video Technology*, 13:657–673, 2003.

[52] Y. Wang, S. Panwar, S. Lin, and S. Mao. Wireless video transport using path diversity: Multiple description vs. layered coding. In *Image Processing Proceedings*, pages 21–24, 2002.

[53] Y. A. Wang, C. Huang, J. Li, and K. 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.

[54] H. Wu, M. Claypool, and R. Kinicki. Adjusting forward error correction with temporal scaling for tcp-friendly streaming mpeg. Technical report, ACM TOMCCAP, 2005.

[55] H. Zhao, Y. Q. Shi, and N. Ansari. Hhiding data in multimedia streaming over networks. In *2010 8th Annual Communication Networks and Services Research Conference*, 2010.