FlexCast: Graceful Wireless Video Streaming

Siripuram T Aditya, Sachin Katti
Stanford University
{staditya,skatti}@stanford.edu

Abstract

Video streaming performance on wireless networks is choppy. The culprit is the unpredictable wireless medium, whose fluctuations results in fluctuating throughput and bit errors. Current video codecs are not equipped to handle such variations since they exhibit an all or nothing behavior. If the channel is strong and above a threshold, the video stream gets decoded perfectly. If not, typically nothing gets decoded. Thus, there is no graceful degradation with wireless conditions.

In this paper, we present a new technique FlexCast, that delivers a video reconstruction whose quality automatically varies with the channel conditions. The key idea is a novel joint-source channel code, that allows the sender to continuously transmit video bits, and the receiver to decode a video quality corresponding to the number of bits it receives and the instantaneous wireless channel quality. We show via experimental evaluation that FlexCast performs almost as well as the optimal scheme, and outperforms the state of the art graceful video delivery systems by nearly 6dB PSNR.

Categories and Subject Descriptors

C.2 [**Computer Systems Organization**]: Computer-Communication Networks

General Terms

Algorithms, Performance, Design

Keywords

Wireless, Video, Coding

1. INTRODUCTION

Wireless video is becoming the major driver for mobile wireless traffic [1]. However, the actual user perceived mobile video performance leaves a lot to be desired. The culprit is the combination of current video codecs and unreliable wireless networks. Video codecs are designed to work at a particular bitrate (adjustable over long time scales by the video server), however if the instantaneous wireless channel quality cannot support that bitrate (i.e. it can deliver that bitrate only with a higher BER), video performance suffers drastically. The reason is that video codecs exhibit an all or nothing behavior, below a particular error free bitrate, they typically cannot decode the raw video stream, leading to fluctuating video quality.

The main reason for this choppy performance is the implicit assumption that senders of the video stream can measure the wireless link quality, and adjust the bitrate accordingly. However, this assumption is untenable considering the actual behavior of wireless links, and the current architecture of video streaming systems. First, wireless links vary continuously and unpredictably due to a variety of factors (interference, multipath fading etc). Second, the majority of wireless video is consumed over the Internet, with video streams residing in data centers connected over long latency paths (on the order of tens to hundreds of milliseconds) to the wireless link. The typical coherence time (period over which wireless channel conditions are stable) of a wireless link is on the order of a few milliseconds [25], hence even if one could measure the channel and send it back to the video sender, the information would be stale by the time the streamer acts on it. Due to this reality, senders are forced to pick conservative bitrates, that are wasteful when wireless channel conditions are good, and harmful when conditions are bad.

In this paper, we present the design and implementation of FlexCast, a novel technique that achieves a video reconstruction that degrades gracefully with instantaneous wireless link quality even in fast varying mobile channels. FlexCast works end-to-end, requiring changes to only how video is encoded and decoded. Further, the video sender in FlexCast is very simple, it transmits a continuous stream of encoded video bits and does not require any feedback about the wireless network conditions. The receiver decodes from any received packet, even those with errors, and achieves a video reconstruction commensurate with the link quality at that point. Thus, FlexCast preserves the simplicity of the current end-to-end design and yet provides graceful performance.

FlexCast's design is motivated by the observation that even if a receiver cannot correctly decode a transmitted bitstream, each decoded bit comes with a confidence estimate known as the soft information. Conceptually, soft information can be thought of as the PHY's estimate of the probability that a transmitted bit was "1" or "0". Soft information can be generated for any constellation and channel coding scheme (e.g. convolutional coding schemes in WiFi can compute a soft output of their decoding decisions). The key insight behind FlexCast is that the quality of the soft information (i.e. the receiver's confidence in its decisions) increases with channel strength, and hence if we reconstruct the video stream based on soft information (called ***soft reconstruction*** henceforth), the quality of the video will be proportional to the channel strength.

MOBICOM'11, September 19–23, 2011, Las Vegas, Nevada, USA.

However, soft reconstruction alone is not sufficient. The reason is that in video not all bits are created equal, the most significant bits (MSBs) corresponding to the lower frequency components are much more important than others. Decoding them with high confidence significantly reduces distortion, while decoding the least significant bit (LSBs) of a high frequency component is not nearly as important. Motivated by this observation, a second key component in FlexCast's design is a rateless video codec that enables ***proportional representation***, i.e. the ability to automatically provide different bits in a video frame a level of protection that is proportional to their importance. Note that the protection is rateless, i.e. we do not need to know the wireless channel conditions in advance to decide on the redundancy to add. At the receiver the confidence in decoding a bit is proportional to the amount of protection added at the sender, and the instantaneous channel quality, thus achieving the desired proportional representation.

FlexCast does require access to soft information from the wireless PHY at the receiver. However, the good news is that soft information is already computed at the PHY for decoding channel codes, FlexCast only requires the PHY to expose this already computed information. Thus, architecturally FlexCast is similar to the recently proposed SoftPHY abstraction [13] and uses a modular network stack at the receiver where the PHY interface is augmented to expose soft information along with decoded bits.

We have implemented FlexCast by modifying the popular ffmpeg [2] MPEG4 encoder/decoder and evaluated it on top of a 802.11a/g style wireless PHY in the GNURadio codebase. We evaluate FlexCast in a testbed with USRP2s and compare it to the omniscient scheme (one which knows the network conditions exactly in advance and picks the best video bitrate and channel FEC accordingly), as well as SoftCast [12] and Apex [23], the two state of the art graceful video delivery systems. Our experimental evaluation shows that FlexCast provides graceful wireless video delivery across a wide variety of varying and unknown network conditions. Specifically, we show the following:

- FlexCast achieves a video reconstruction that is within 1 dB of the omniscient scheme across a wide range of SNRs, without having any knowledge of the channel SNR.
- FlexCast outperforms SoftCast which also operates without knowing the channel SNR by nearly 4dB.
- In walking speed mobility scenarios, FlexCast outperforms Apex by nearly 3dB PSNR and SoftCast by 4dB.
- In fast mobility scenarios where network conditions fluctuate rapidly, FlexCast outperforms Apex by nearly 6dB PSNR and Softcast by 4dB.

FlexCast also provides architectural gains over Apex and SoftCast in addition to the PSNR gains discussed above. FlexCast's design is layered and end-to-end, i.e. unlike Apex and SoftCast it does not require the wireless PHY to be aware of video semantics and appropriately change its own coding and modulation choices. The layered design preserves modularity and allows the PHY and video layers to independently evolve and innovate. Finally, the encoding and decoding complexity of FlexCast is linear, and we reuse most of the components found in MPEG4. Hence, we believe FlexCast can be integrated into existing video codecs via small modifications.

2. RELATED WORK

FlexCast is related to a large body of work on video encoding, including MPEG and its variants, layered encoding and multiple resolution coding [6, 15, 16, 22, 27, 10] (the reference list is representative and not meant to be complete). These schemes typically encode the video into multiple layers, with each successive layer needing the previous layer for successful decoding. All these schemes assume the source knows the wireless link conditions and can carefully choose the best bitrate and codec for each layer, given the channel SNR. Similar arguments apply to techniques such as superposition coding [8], which is a PHY layer technique for providing unequal error protection. To implement superposition coding, the sender needs accurate knowledge of the channel SNR. Further superposition coding requires that receivers have channel strengths that are significantly different, and finally, it is limited in the video resolutions it can actually provide. FlexCast makes no such assumptions, and provides graceful performance without requiring any link state feedback.

FlexCast is directly related to two state-of-the-art techniques, Softcast [12] and approximate communication [23] that also aim to provide graceful wireless video performance. However, these techniques both need the PHY at the wireless AP to be aware of video semantics, and appropriately encode their own transmissions. Specifically, Softcast needs a completely new wireless PHY that shuns all conventional modulation and coding techniques. Further, SoftCast is essentially an analog transmission technique, and cannot obtain the gains we get from digital channel coding, a point we discuss in depth in Sec. 7. Approximate communication (or Apex) requires the wireless PHY to be aware of different video frames, and maps data from these two frames to carefully chosen constellation points to provide unequal error protection. However, to realize Apex's benefits the channel SNR should be sufficiently high ($>$ 10dB) so that we can use a dense constellation (at least 16 QAM). Such high SNRs are uncommon in mobile scenarios. Further, it requires knowledge of the channel SNR to adjust the symbol mapping on the constellation. FlexCast has none of these onerous requirements, it works with existing wireless PHYs without requiring any link feedback, and yet outperforms both SoftCast and Apex as we will see in our evaluation.

FlexCast is also related to digital fountain codes [14] in the sense that it does not matter if a packet is lost, and it can recover from the other coded packets. However, there are two critical differences. First, digital fountain codes are all or nothing codes. Hence if n packets are encoded using a digital fountain code, we need at least n coded packets before we can recover anything. In FlexCast, we can recover a reconstruction even if we get less than n packets with a quality proportional to the number of packets received. Second, digital fountain codes only work with packets with no bit errors. FlexCast can take advantage of even packets with bit errors, that are common in wireless networks.

Finally, FlexCast is related to prior work on joint source-channel coding [20, 18, 11] for video. This line of work shares the same high level principle as FlexCast, i.e., since entropy coding and error prone wireless channels do not play well with each other, a video encoding scheme should consider the source compression and the channel coding problem separately. Similar to FlexCast, many of these schemes consider dividing video bits according to their importance and providing unequal error protection. FlexCast differs in the design of the protection mechanism itself, since the same code compresses as well as protects, and provides a semi-rateless property that allows FlexCast to operate well even when link conditions are varying and the transmitter has no channel state information.

wireless channels are error prone,

3. OVERVIEW

We begin with a short description of how to visualize video sig-

nals, and what achieving graceful performance means. The discussion informs many of FlexCast's key ideas that are described subsequently. We will first discuss a *single frame*, and then generalize the discussion to a sequence of frames or video. The frame discussed below corresponds to the I-frame in traditional video codecs. A video server encodes the video stream and sends it to the wireless AP, which in turn encodes it for transmission on the wireless link to the mobile client.

A video frame is a matrix of pixels, with each pixel typically represented by a 8-bit value. However, since natural images do not show large variations, there is a lot of redundancy in the pixel representation. To compress, typically a linear frequency transform is applied to the raw pixel matrix (e.g. Discrete Cosine Transform in MPEG4). The lack of large variations in natural images implies that higher frequency values will be close to zero, which can be discarded (or represented using a small number of bits by quantization) without causing human perceivable signal loss.

The quantized DCT components are then compressed using an entropy code such as Huffman code, packetized and sent to the wireless AP. The wireless AP adds FEC via channel coding to protect against channel distortions and transmits it on the wireless link. The mobile client after receiving the packets, first decodes the FEC from the AP, and then decodes the video stream to recover the DCT components. It then applies an inverse DCT to recover the pixels representing the I-frame.

Since wireless link qualities fluctuate (especially in mobile networks), the AP has to constantly monitor and pick the best link bitrate to transmit at. If the AP incorrectly picks a bitrate that the channel cannot support, then packets will be received in error. To recover from such errors, the AP retransmits at a lower bitrate and attempts to ensure that the packet is delivered correctly. The net effect is fluctuation in goodput to the client, as well as packets being received with errors.

If the goodput drops below the encoded video bitrate, then the video stream cannot be decoded, causing stutter in video rendering and buffering delays. Hence if the goodput fluctuates as it often does in mobile networks, the client will experience intermittent stuttering and buffering delays. To avoid such cases, video senders pick conservative and low video encoding bitrates to ensure that they are likely to deliver a minimum quality. Hence clients are forced to accept a relatively low quality video playback, and cannot take advantage of the times when the channel quality is good. To sum it up, *wireless link variability implies video playback has to be designed for the worst case scenario, and cannot gracefully adapt to the instantaneous link quality.*

Decoding Video from Erroneous Frames: A possible approach to improve video performance is to use the erroneous packets that are thrown away by the PHY and see if their partial information can be used to improve video decoding. In this case, the video decoder would take packets with bit errors from the PHY and attempt to decode the video stream. However this immediately fails, since bit errors are numerous and randomly distributed. If the bit errors are in the most significant bits of the DCT components, the resulting distortion in video will be quite high.

3.1 FlexCast

FlexCast's goal is to enable graceful video delivery, i.e., deliver a video reconstruction quality that is proportional to the instantaneous wireless link quality. Given that wireless link fluctuations are an unavoidable reality, FlexCast assumes that bit errors and packet loss will occur. But instead of throwing these erroneous packets away and requesting retransmissions, FlexCast opportunistically exploits these erroneous frames to decode the video, albeit not fully but with a reconstruction error that is proportional to the difference between what link quality the packet was encoded for and what the actual quality was. FlexCast's design is based on two key components:

1) Proportional Representation: FlexCast exploits packets with bit errors to decode video. However, since bit errors in the MSBs of DCT components cause a large amount of distortion, it is harmful to treat all bits equally. In FlexCast, bits should receive proportional representation, i.e. *the protection a bit gets should be proportional to the overall error a decoding error on that bit would produce*. To implement proportional representation, FlexCast first groups bits of equal importance together by a technique called *distortion grouping*. Intuitively, distortion grouping clusters bits (from the bitstream representing the DCT components) according to the amount of error they would cause if they were decoded in error. For example, all the MSBs in a 8 bit binary representation would form one distortion group (since they would cause an error of 128 for a decoding error each), and so on for the other bit positions. FlexCast then *separately encodes* these distortion groups according to their relative importance.

To add protection, FlexCast designs a rateless video codec based on raptor codes [24] that allows the sender to protect bits in different distortion groups according to their relative importance. In other words, it has the property that if a particular group of bits is encoded with more bits, then its reconstruction error is correspondingly lower after going through a noisy wireless channel. However note that the codec is rateless, it operates without any knowledge of the network conditions. Since network conditions are unknown at the video sender, the video codec cannot guarantee that any distortion group is perfectly decoded. However, it guarantees that the reconstruction error for the group is inversely correlated to instantaneous channel quality.

2) Soft Reconstruction: The second key component of FlexCast is to exploit physical layer soft information at the client to provide a better video reconstruction. FlexCast uses the SoftPHY interface from prior work [13] to expose soft information from the PHY layer to the upper video layer for video decoding. Soft information can be thought of as a probability distribution $p_i(1), p_i(0)$ on the actual bit decisions for bit i, i.e., an estimate of the receiver's confidence that the transmitted bit was "1" or "0". Soft information for each decoded bit is calculated by the decoding algorithm for the rateless video codec as explained in Sec. 5. FlexCast exploits this soft information to perform ***soft reconstruction*** of the DCT component represented by the bit. Specifically, FlexCast computes the DCT coefficient as

$$x = \sum_{i=1}^{i=8} (p_i(1) * 2^{8-i} + p_i(0) * 0) \quad (1)$$

where $b_i, i = 1, \ldots, 8$ represents the 8 bits for that DCT component. Thus it computes the expected value of the DCT coefficient given soft information on the bit decisions. The key idea is that the quality of the soft information depends on the channel strength at that point. For example, if the channel is bad, the soft information will be very indecisive, i.e. $p_i(1) = p_i(0) = 0.5$. In this case we avoid making large errors in estimating the DCT component, and thus the error for the recovered DCT component scales inversely with the channel strength.

4. VIDEO SENDER

MPEG4 takes a sequence of raw video frames, and encodes it to produce combinations of three different frames, First, the I-frame is the reference frame and is essentially a still image that is encoded independently. Next, there are two predicted framed, P-frames and B-frames. A P-frame holds only the changes in the image from the previous frame, while the B-frame incorporates the future frame too.

Figure. 1 shows the high level procedure for MPEG encoding. In MPEG, the raw I-frame is divided into a set of macroblocks. To create the P and B frames, the macroblocks in consecutive raw frames are compared to each other, and the closest macroblocks are identified and the corresponding motion vectors are computed. The differences between the matching macroblocks are then computed into a *differential macroblock* and stored in the P and B frames. After the macroblocks are produced, the rest of the encoding process is the same for all frames.

1) Discrete Cosine Transform: Each macroblock is first passed through a *DCT transform* to compute their frequency contents. Since natural images do not typically exhibit sharp variations, the high frequency components will be close to zero.

2) Quantization: In the second step, each macroblock DCT component is quantized. Quantization at a high level, compresses a range of continuous values into a single discrete value. In MPEG, the important low frequency components are quantized into a larger number of discrete levels, and hence need more bits to represent. The less important components are quantized into a smaller number of discrete levels, and require fewer bits to represent. Depending on the desired resolution, different quantization strategies are employed.

3) Compression: The quantized bits in a macroblock are then compressed using a combination of Run-Length encoding (RLE) and Huffman encoding. The compression step naturally produces a random bitstream of reduced size [21]. These bits for each macroblock are then packaged into the appropriate frame to create the compressed I, P and B frames.

FlexCast only *changes the last step*, it replaces the compression stage (RLE and Huffman) with its own rateless video codec. The rest of the section will therefore focus on describing our rateless video code, we refer the reader to [21] for a detailed description of the earlier stages in the MPEG encoding process. Further, note that FlexCast also simplifies the quantization step for MPEG, since it only uses the quantization strategy that produces the highest video resolution. FlexCast thus retains the same I, P and B frame structure of traditional MPEG, and only changes the last step of how these frames are populated. We hope that due to this relatively small change, it will be easier to integrate FlexCast into MPEG standards.

4.1 FlexCast

FlexCast takes the output bits from the quantizer and encodes them in two stages: distortion grouping and rateless video coding.

4.1.1 Distortion Grouping

First, FlexCast groups bits from the quantized DCT coefficients which if decoded incorrectly, contribute equal amounts to the overall mean square error. For example, if DCT components are represented using 8 bits each, the most significant bits would form one distortion group and so on for the other bit positions. MPEG typically uses variable length quantization [21]), and hence the value

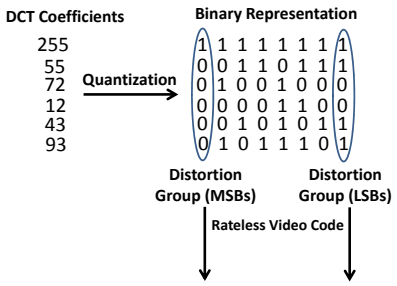

Figure 2: FlexCast's distortion grouping algorithm. Bits which if decoded incorrectly cause the same amount of distortion are grouped into one group.

of each bit is not just dependent on its position. Therefore, instead of grouping bits based on their position, we compute for each bit position in each quantized DCT value their contribution to the overall reconstruction error if it is decoded incorrectly. These estimated distortion contributions are then clustered, and each cluster is grouped into a separate group. Note that this assignment is independent of the video being encoded, and needs to be computed only once offline. The distortion groups are arranged in descending order of their importance (i.e., the group which causes a higher error if it is decoded incorrectly is more important). Figure 2 demonstrates the grouping algorithm.

4.1.2 Rateless Video Code

FlexCast next encodes the bits in each distortion group using it's rateless video code separately. The rateless video encoder is based on Raptor codes [24], which are an efficient class of rateless channel codes. Raptor codes can produce an infinite stream of coded bits without requiring any knowledge of network conditions. However, traditional Raptor codes have an "all or nothing" property, depending on the channel quality they require a minimum number of bits after which all the input bits are decoded, and below which nothing is. Instead, in FlexCast we want graceful behavior, given any number of bits received we want a reconstruction proportional to the number received and the instantaneous channel quality.

To achieve this graceful property, FlexCast modifies the Raptor code design. Lets say there are n bits in a distortion group $D(i)$. Typically, Raptor codes take all the n bits and pass them through a combination of an LDPC (Low Density Parity Check) codes [9] and a standard digital fountain code (the Luby LT codes [14]) to produce coded bits. However in FlexCast we modify the flow:

1. First, the n bits are not coded in any way, they are transmitted directly. This is unlike Raptor codes, which directly start encoding the n bits.

2. **LDPC code:** Next, pick 3 bits at random from the n bits in the distortion group, and XOR them to create an intermediate coded bit. Repeat this process n times to create n intermediate coded bits.

3. **LT code:** The LT code in FlexCast uses a robust Soliton distribution with a mean 4.6 [14]. The encoder first samples this distribution to produce an integer m. Next it picks m bits at random from the intermediate coded bits from the LDPC stage, and XORs them to create a parity bit. This step can be repeated as many times as needed to create an infinite stream of parity bits. The parity bits are appended to the n un-encoded bits from the first step to produce the final rateless encoded video bits.

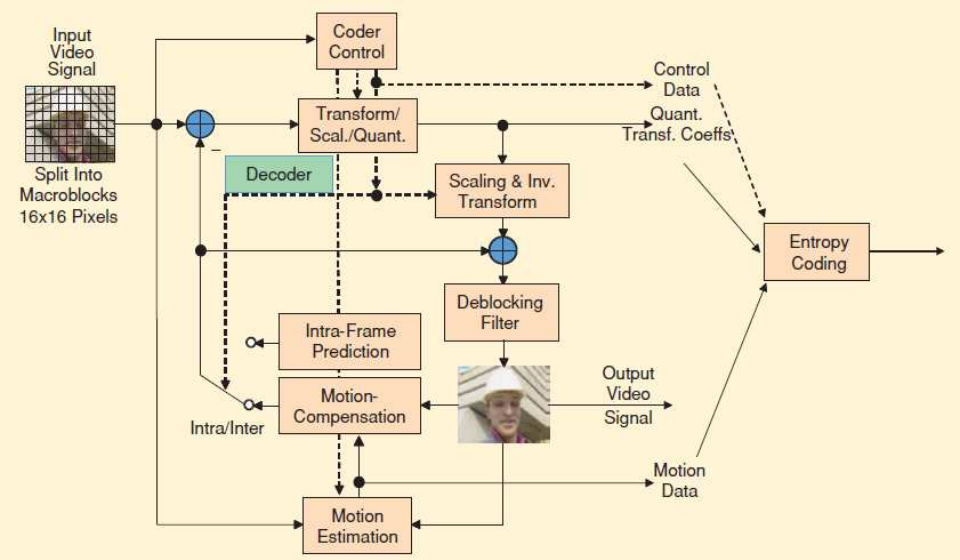

Figure 1: Standard MPEG4 encoding. FlexCast only replaces the entropy coding stage.

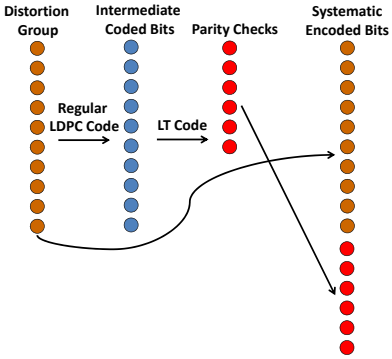

Figure 3: FlexCast's rateless video code based on Raptor codes. FlexCast produces "systematically" encoded bits, i.e the initial input bits appended with parity checks.

To see why this modification to Raptor codes provides the desired graceful performance, note that the new code is a "systematic" code. In systematic codes, the n input bits are kept as is, and parity bits are appended to the end of the n input bits to produce the coded bits. If any of the initial n bits are received in error, the extra parity bits are there to help correct them. This automatically produces graceful performance, since the soft reconstruction error in the initial n bits is directly correlated to the raw channel quality. Next as the receiver gets parity bits, as we will see in Sec. 5, we successively reduce the reconstruction error in the first n bits.

FlexCast repeats the above steps for each distortion group. Figure 3 depicts the process.

4.1.3 *Proportional Representation*

Packetization in FlexCast exploits the fact that the rateless video code is graceful and proportional. Specifically, lets say the video sender has the opportunity to send M packets of size B (where B is typically 1500 bytes) bytes each per second, and there are G distortion groups $D(1), D(2), \ldots, D(G)$. Then the MB bytes are allocated as follows among the distortion groups in the following ratio

$$\text{Allocation for distortion group i} = MB * \frac{N(D(i)) * E_{D(i)}}{\sum_k N(D(k)) * E_{D(k)}} \quad (2)$$

where $E_{D(i)}$ is defined as the average MSE if a bit in the distortion group $D(i)$ is decoded incorrectly, and $N(D(i))$ is the number of bits in distortion group $D(i)$.

In other words, the fraction of space allocated to a distortion group is proportional to its importance, as well as the number of bits in the distortion group. As discussed in the previous section, the rateless video code in FlexCast has the property that if a distortion group receives more bits, then its reconstruction error correspondingly drops. Hence the above allocation automatically ensures that more important distortion groups get relatively more bits allocated to them from the budget and are decoded with an error proportional to their importance.

The video server sends these encoded video packets to the wireless AP. The wireless AP takes the packets and transmits them on the wireless link.

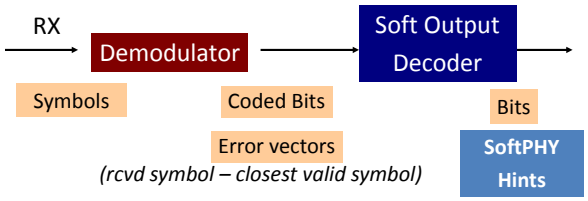

Figure 4: FlexCast augments the PHY layer to pass on SoftPHY hints.

5. MOBILE CLIENT

The decoding at the mobile client proceeds in three steps: SoftPHY, decoding the rateless video code and soft reconstruction.

As discussed earlier in the paper, FlexCast assumes a SoftPHY interface at the physical layer [13] of the mobile client. Intuitively, a SoftPHY interface exports per-bit confidences along with the decoded bits for both correct and incorrect packets. SoftPHY hints

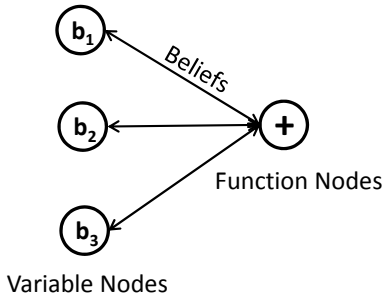

Figure 5: Example of Belief Propagation Factor Graph where $b_3 = b_1 \oplus b_2$.

can be computed for any PHY using a linear convolutional or block code, examples of which include WiFi, WiMax and Zigbee. We omit the details here for brevity, but refer the reader to [13, 26] for a detailed description of SoftPHY computation for a number of standard wireless PHYs. In the rest of the paper we will assume that we have LLRs (log likelihood ratios) or the probabilities on each bit decision, i.e. the probability that the decoded bit is 1 or 0.

5.1 Decoding the Rateless Video Code

FlexCast exploits the SoftPHY hints to decode the rateless video code using an algorithm called belief propagation [19]. We will describe how the decoding algorithm decodes the bits belonging to a single distortion group $\mathbf{D}$, i.e. $b_k, k = 1 \ldots, n$ from the LLRs it receives from the SoftPHY. Remember that the n bits from the distortion group are encoded to produce coded bits $c_1, \ldots, c_M$ as follows by the video server: the first n bits are the same as the raw bits themselves, and following them are parity bits created using a combination of LDPC and LT codes. We will assume that we have received $m > n$ such encoded bits at the mobile client, with the corresponding LLRs. From the LLRs we can compute the probabilities $p(c_i = 1), p(c_i = 0)$.

The high level goal of the decoding algorithm is to compute the probabilities of the bits in the distortion group b_k being "1" (or "0"). Since the first n bits in the received bits correspond directly to the bits in the distortion group $b_k, k = 1, \ldots, n$, the initial probabilities can be computed,

$$p(b_i = 1) = p(c_i = 1) \qquad i = 1, \ldots n \tag{3}$$

The accuracy of the initial probability estimates above depends on the wireless link strength, if the wireless link is noisy, then the probability estimates can be indecisive i.e. $p(b_k = 1) \approx p(b_k = 0) \approx 0.5$, and hence we cannot confidently decode. The extra parity bits that are transmitted by the video server enable us to improve these probability estimates further and improve decoding. Remember that these extra parity bits are generated by passing the first n distortion group bits through a combination of linear LDPC and LT codes. Thus, these parity bits are a linear combination of a subset of the first n bits. To understand how these linear combinations help us improve the estimate of the probabilities, we will use a simplified example where there are only two bits in the distortion group b_1, b_2 , and we receive one extra parity bit, which is just the XOR of the two distortion group bits $b_3 = b_1 \oplus b_2$. These three bits b_1, b_2, b_3 are transmitted by the wireless AP after channel coding, and on the receiver, the SoftPHY passes them up with the computed LLRs.

Lets assume the LLRs for these three bits are such that the corresponding probabilities for the decoded bits being "1" are $p(b_1 = 1) = p(b_2 = 1) = p(b_3 = 0) = 0.7$. From the first two SoftPHY hints, we get the raw initial probabilities for the bits $p(b_1 = 1) = p(b_2 = 1) = 0.7$. How can we use the third SoftPHY hint to improve these probability estimates?

Intuitively the high probability that the third bit is "0" (the XOR of the first two bits), coupled with the high initial probabilities that the first two bits are "1" too suggests that the first and second bits are very likely "1". To quantify how likely, FlexCast uses an algorithm called belief propagation [19]. The algorithm works at a high level by setting up a *factor graph* that encodes the linear constraints among all the bits. In the example above, the linear constraint is that the XOR of the three bits is 0 (since $b_1 \oplus b_2 \oplus (b_1 \oplus b_2) = 0$). Figure 5 shows the factor graph for this example, the three edges going to the XOR function node imply that the input three bits are supposed to add up to zero. The nodes on the right are called the *variable nodes*, and the node corresponding to the constraint is called the *function node*.

As the name of the algorithm suggests, *beliefs* about bit values are propagated along the edges of this graph. The three nodes representing the bits are initialized with beliefs using the LLRs from the physical layer. Hence the beliefs for the first two nodes are $p(b_1 = 1) = p(b_2 = 1) = 0.7$ and for the third node $p(b_3 = 1) = 0.3$. Next, these nodes pass their beliefs to the function node. The function node updates its beliefs about bit b_1 using the following equation

$$p(b_1 = 1) = p(b_2 = 1)p(b_3 = 0) + p(b_2 = 0)p(b_3 = 1) \tag{4}$$

which is essentially computing the probability that $b_1 = 1$ given the probabilities of the other two bits. A similar equation can be written for the other two bits. Plugging the values into the equation we find that the belief at the function node for the first two nodes is $p(b_1) = p(b_2) = 0.58$. These beliefs are sent back to the nodes, which then have to update their own beliefs about their respective values.

At this point the first two nodes receive conflicting beliefs about their bits, the one computed from the LLRs says $p'(b_1 = 1) = 0.7$, and the belief received from the function node says $p^{''}(b_1 = 1) = 0.58$. To resolve this conflict, the node computes the new belief using the following equation

$$p(b_1 = 1) = \frac{p'(b_1 = 1)p^{''}(b_1 = 1)}{p'(b_1 = 1)p^{''}(b_1 = 1) + p'(b_1 = 0)p^{''}(b_1 = 0)} \tag{5}$$

The equation implies that the new belief is the probability that the two beliefs agree on the value of bit $b_1 = 1$, divided by the probability that the two beliefs agree on the value being "1" or "0". This computation can be intuitively thought of as the probability that the decoder would have decoded that bit to be "1" at that point, given that it decodes anything at all. To see why, consider the decoding rule: the decoder has to make a decision on the value of bit b_1 by tossing two weighted coins corresponding to the two beliefs. If the coins both showed "1", then the bit would be decoded to "1", and "0" if they both showed "0". In this experiment, the probability that we would decode to "1" is given by Eq. 5. In the above two bit example, the new beliefs at the first and second nodes can now be computed as $p(b_1 = 1) = p(b_2 = 1) = 0.76$, which is a marked improvement from 0.7. Thus the extra linear constraint from the third node helps improve the probabilities of the first two bits, and pushes it closer and closer towards the correct decoding decision.

5.1.1 Algorithm

The above description is a high level sketch of the belief propagation algorithm, we encourage the reader to refer to[19] for a detailed algorithm as well as mathematical analysis. BP provably converges and computes the correct final probabilities when the factor graph

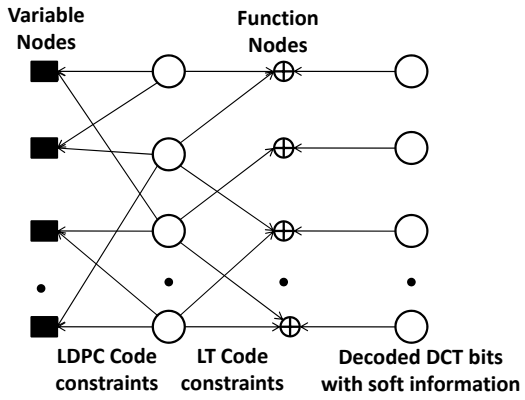

Figure 6: Belief Propagation decoding of FlexCast's code

has no loops. For graphs with loops, BP has been shown to provide very good performance empirically. Finally, we summarize the decoding algorithm based on belief propagation for each distortion group below in a pseudocode. The algorithm is repeated for all the distortion groups.

1. Set up the factor graph in three stages as show in Figure. 6. The first stage consists of variable nodes corresponding to the received bits with SoftPHY hints, the second stage is function nodes that combine the linear constraints of the LT code and LDPC codes, and the final stage corresponds to the variable nodes corresponding to the n bits in the distortion group. The edges connect bits that are part of a linear constraint in the LT/LDPC code.

2. The variable nodes for the first stage are initially seeded with probabilities that they are "1" or "0" via the corresponding LLRs from the SoftPHY. Next, the belief propagation algorithm passes messages through the factor graph edges and updates the probabilities for all the other nodes. The probability update equations are generalized versions [24] of Eqns. 4 and 5. This counts as one iteration.

3. The message passing iterations are repeated until the probabilities of the nodes on the right-most stage, i.e. the bits in the distortion group are all above a threshold, or we have exceeded a pre-specified maximum number of iterations. In our current implementation, we found empirically that a threshold LLR of 15 and a maximum iteration limit of 20 suffices.

At the end of this stage, we have probabilities that the bits in each distortion group are "1" (or "0").

5.1.2 How to realize the compression gains?

FlexCast replaces the entropy coding stage in traditional MPEG4 with its own rateless video code to achieve graceful video delivery. However, entropy coding does provide compression gains in traditional MPEG4 encoding. How do we achieve the same compression gain in FlexCast?

Our approach to achieve compression is based on a well known theoretical result in information theory. The result is based on the insight that when we compress using entropy coding, we are exploiting the data statistics (e.g. 70% of the bits being 0 in a file) to compress. However, if we dont perform this compression directly but instead encode the raw data using a channel code (like the rateless video code FlexCast uses), we can achieve the same gains as compressing at the sender if we send the data statistics to the receiver and it uses these statistics in performing its decoding. In FlexCast that would mean that the BP decoding algorithm described above would initialize the variable nodes at the right in the factor graph using the data statistics to bootstrap the BP decoding algorithm. Caire et al [7] show that this is equivalent to performing entropy coding at the source and then separately channel coding it, i.e. both approaches would require the same number of bits to be transmitted on the wireless channel.

To exploit this insight in FlexCast, we send metadata about the data statistics for each distortion group in the header of the video frame. Similar to other information in the packet headers, this metadata is important and should be correctly received at the receiver for correct decoding. Hence we protect this using extra FEC. However, note that the metadata overhead is negligible, in our experiments we found that it adds 2% extra overhead including the FEC.

5.2 Soft Reconstruction

Soft reconstruction uses the probability estimates to compute the expected value for the DCT components. Specifically, the receiver collects all the bits representing a single quantized DCT component along with their corresponding probabilities, and computes its estimate of the DCT component as follows:

$$y = \sum_{i=0}^{L-1} 2^i * p_1(i) \tag{6}$$

where L is the number of bits used to represent the quantized DCT component, and $p_1(i)$ represents the probability that the i'th bit in the representation of the DCT component is "1". Intuitively, the above equation is computing the expected value of the DCT component, given the probability distributions on each bit. Clearly if all bits have been correctly decoded (i.e. $p_1 = 1.0$ if bit "1" was transmitted), then the computed y will match the value of the actual DCT component. However, when there is still some uncertainty, the soft reconstruction error will only be proportional to the probabilistic uncertainty in the soft information. Since this probabilistic uncertainty is correlated inversely with the channel quality, FlexCast achieves the graceful reconstruction we hoped for.

The final step is the same as traditional MPEG, converting the DCT components to raw pixels for all frames: I, P or B. Specifically, the recovered DCT components above are passed through an IDCT to recover the raw macroblocks for the I-frames, and the differential macroblocks for the P and B frames. These differential macroblocks are now recovered using the motion compensation vectors to compute the raw frames, and the video is rendered.

6. IMPLEMENTATION

Our current implementation of FlexCast is built using the widely available ffmpeg [2] video encoding/decoding library. As we discussed before, FlexCast retains the motion vector computation, quantization, and transform steps of traditional MPEG4. We only replace the entropy encoding/decoding stage with our rateless video code. However, our design does affect some of the parameters of the earlier stages, which we describe below.

First, traditional encoders adjust the Quantization Parameters (QP) based on what network throughput is available. QP adjusts the resolution of the video and is thus a form of lossy compression, and is controlled to meet a user specified bitrate budget (e.g. 1Mbps video bitrate). In FlexCast, there is no concept of a set encoding bitrate, hence we set QP to as low as possible. In other words we encode video at the highest resolution possible.

At the receiver, we augment the the wireless PHY to expose SoftPHY hints. Our current implementation is based on the MIT

12dB	10dB	8dB	6dB
0.3	0.37	0.44	0.51

Table 1: Microbenchmarks - Decoding CPU time normalized wrt actual video stream time.

GNURadio WiFi-like implementation that was used in prior work on SoftRate [26]. It is an OFDM PHY with all the modulations and convolutional coding rates used in WiFi, but with the narrower bandwidth the USRP2 can support. It includes the soft decoding algorithms needed to expose SoftPHY hints [13]. We omit the details for brevity, but the implementation is the same as the one used in SoftRate [26].

For the video decoding component, we retain all the traditional components of MPEG4 except entropy decoding, which is replaced with the belief propagation (BP) algorithm and soft reconstruction as discussed in Section 5. Our current implementation of BP is based on a widely used message passing algorithm implementation [3].

6.1 Complexity

At the encoder, the extra complexity is from the distortion grouping and rateless encoding components. Both operations have linear complexity in the number of input bits. However, we note that encoding is typically not done on the fly and hence is not on the critical path. Senders can encode their video and store it in advance for transmission. Further, unlike current systems which have to encode their video for multiple bitrates and qualities (e.g. 300kbps, 600kbps, 1Mbps for Youtube) into separate files, FlexCast keeps only one high resolution encoded file.

At the decoder, the major source of extra complexity comes from the belief propagation algorithm. The complexity of the algorithm is proportional to the number of edges in the factor graph since that determines the number of messages passed around every iteration. For FlexCast's rateless video code, the number of edges is equal to a constant (≈ 7.6) times the number of bits in the distortion group. Hence the decoding complexity is linear in the number of encoded bits.

Microbenchmark: FlexCast is implemented on a PC with an Intel Core i7 980x processor and 8GB of RAM. We benchmark the performance of our implementation by evaluating how long it takes to decode a 30 second video encoded at a 720p resolution. We normalize the decoding time with the actual video playing time, since at the very least we have to ensure that decoding works faster than the rate at which video needs to be displayed. Table 1 shows the results at different channel SNRs. As we can see, the normalized time is typically less than 0.6 and hence can easily keep up with the video playback. The decoding time grows as the channel gets weaker which we intuitively expect. With noisy channels, the decoder takes more iterations to converge on the LLRs, leading to the higher decoding time.

6.2 Experimental Setup

We evaluate name using both simulations and experiments on a USRP2 testbed. we describe the setup below.

Hardware and Testbed: Our experiments use nodes connected to the USRP2 radio platform. The USRP2 platform, is a radio frontend which is connected to a PC that is running the GNURadio signal processing framework. We deploy an indoor testbed of 14 USRP2s in our lab. The node locations are shown in Fig. 7. To simulate different channel SNRs, we move around the nodes until

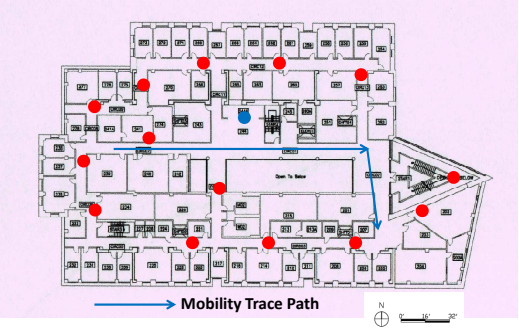

Figure 7: FlexCast Indoor Testbed Layout

we can obtain the SNR for which we want to run the experiment. The blue arrow represents the path the node takes for our mobility experiments.

Compared Schemes: We compare the following four schemes:

- **FlexCast:** This is our implementation of FlexCast which follows the description in Sections 4 and 5.
- **Omniscient MPEG (Omn-MPEG)**: This uses traditional MPEG4, but assumes omniscient knowledge of network conditions. Specifically, here the sender is assumed to instantaneously know the current channel SNR and network capacity, and empirically picks the optimal combination of MPEG encoding and wireless PHY bitrates that maximizes the PSNR for that channel SNR. Hence, this scheme would be the *best* possible performance by a conventional scheme.
- **SoftCast:** This is a recent state-of-the-art scheme that redesigns the wireless PHY at the AP to transmit a signal that is directly proportional to the pixel value [12] to achieve graceful performance.
- **Apex** is based on a technique called approximate communication [23], and is backward compatible with existing MPEG4 but modifies the wireless AP's PHY to provide differential protection and provide some graceful performance. Apex also assumes that a traditional wireless rate adaptation and MAC protocol is operating underneath. To be fair to Apex, since FlexCast uses SoftPHY hints, we implement SoftRate [26], a state of the art rate adaptation protocol that also uses SoftPHY hints and outperforms the standard rate adaptation algorithms.

Tested Video and Video Quality Metric: We use a standard reference video available at the Xiph.org foundation [4]. In particular, we use the football video in the SIF format which has 300 frames in a 4:3 screen format. The football video has large temporal variation, and hence stresses the video encoding and decoding system and is not as compressible as a more static video.

We compare the four schemes using the Peak Signal-to-Noise Ratio (PSNR). PSNR is a standard measure of video/image quality. We refer the reader to [12, 23] for a description of the computation. The key thing to note is that a PSNR below 16dB is effectively noise, whereas a PSNR above 40dB is excellent quality. A PSNR increase of 3dB means that your video quality is relatively twice as better.

7. EVALUATION

We evaluate FlexCast and the other compared approaches in a variety of settings using experiments and trace driven simulations. We summarize the key findings below

- FlexCast performs as well as the Omn-MPEG scheme in both low mobility and high mobility scenarios and across all practical channel SNRs, achieving almost the same PSNR without requiring any of the network state feedback or adapting in any way.
- FlexCast outperforms both SoftCast and Apex. For high mobility with medium to low SNRs (typical cellular scenario), FlexCast provides an average PSNR improvement of 6 dB over Apex, and 4 dB over SoftCast.
- FlexCast's gains come from both soft reconstruction and proportional representation. Without soft reconstruction, FlexCast loses 6dB from its PSNR performance. Without proportional representation FlexCast loses nearly 8dB PSNR.

Symbol Budget: If there are no real-time constraints then every video can be perfectly decoded and rendered with zero error. Video quality varies because frames have to be rendered within a certain time period beyond which they are useless. Hence to fairly compare the different schemes it is important to set a uniform channel airtime budget which is independent of the scheme, what the channel SNR is and so on.

Our approach here is to assume that the client is going to experience the best video experience when the channel is very good. So if we pick a very high video PSNR (specifically 45dB, beyond which the human eye cant tell the difference), we calculate what channel airtime the Omn-MPEG scheme needs at a relatively high channel SNR (20dB) to achieve that PSNR. We take this to be the channel airtime budget for all other schemes as well as channel conditions. Naturally, for the same channel airtime at a lower channel SNR, the achieved goodput will be lower. The goal of our evaluation is to show who gracefully the PSNR drops with varying channel SNRs and other network parameters.

7.1 Testbed Evaluation

We evaluate FlexCast using experiments in our indoor testbed with USRP2s. We compare to Omn-MPEG and SoftCast in these experiments, and are not able to compare with Apex. The reason is that Apex requires a dynamic bit-rate adaptation and MAC protocol operating at the PHY layer, however USRP2s do not meet the timing requirements needed to implement rate adaptation and MAC. However, we do provide a comparison with Apex using trace driven emulation experiments in Sec. 7.2.

FlexCast is layered and works end-to-end, and is compatible with whatever rate adaptation protocol the wireless AP uses. However, as we said above we cannot implement a full dynamic rate adaptation protocol in USRP2s. Hence, to make sure FlexCast is not benefiting from very accurate rate adaptation, we constrain the wireless AP to use only a very crude static rate adaptation protocol when we are running experiments with FlexCast. Specifically, the wireless AP is only allowed to change the constellation among three choices (QPSK, 16-QAM and 64-QAM) and use none of the convolutional channel codes that traditional rate adaptation schemes can use. When the channel SNR is below 8dB, we use QPSK, between $8-15$dB we use 16-QAM and above 15dB we use 64-QAM. The adaptation is done by the receiver notifying the sender using a control packet whenever the SNR crosses any of the above three SNR transition points. Note that the adaptation is very coarse and inaccurate, and happens quite infrequently. Therefore we do not run into the timing related issues for implementing standard rate adaptation protocols on USRP2s.

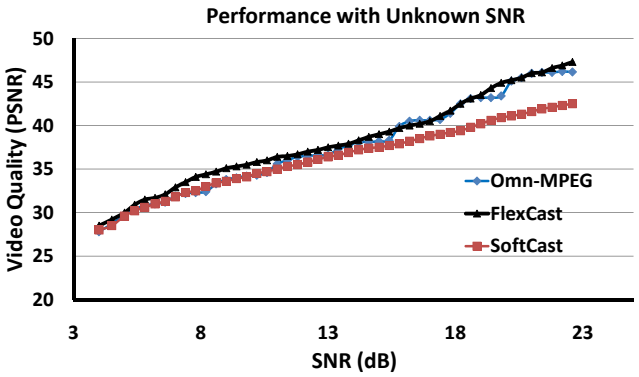

Figure 8: FlexCast performance with unknown SNR. FlexCast performs almost as well as the omniscient MPEG scheme at all SNRs. It outperforms SoftCast by upto 4dB at high SNRs.

7.1.1 Performance with Unknown SNR

Method: In this experiment, we randomly place two USRP2 nodes in our testbed and measure the SNR of the link. We then attempt to transmit our test video between the two nodes. For Omn-MPEG scheme, we transmit the video using all possible combinations of video encoding bitrates and wireless bitrates, and pick the one which achieves the maximum PSNR at the receiver. For FlexCast, we use FlexCast's video encoding and decoding algorithms along with the coarse rate adaptation protocol described above, and compute the reconstructed video's PSNR. Note that FlexCast's video encoder has no knowledge of the channel SNR between the sender and the receiver. Finally, we transmit the videos with SoftCast and compute the PSNR. We repeat this experiment 10 times for the same location of the nodes and take the average PSNR for each scheme. We then change the locations of the two nodes to get a different SNR and repeat the above procedure. We plot the average PSNR achieved by the three schemes vs SNR in Fig. 8.

Analysis: FlexCast performs well across a large range of SNRs, it matches the performance achieved by the omniscient MPEG scheme almost across the entire range. Surprisingly in the low SNR region it often outperforms the omniscient scheme. The reason is that the rateless video code in FlexCast which is based on raptor codes is also a good channel code, especially at low SNRs [24]. The omniscient scheme on the other hand uses convolutional codes which do not perform as well at low SNRs.

FlexCast outperforms SoftCast across the entire range by nearly $2-4$dB. The reason is that SoftCast cannot take advantage of channel coding, it is in essence an uncoded transmission reminiscent of analog TV. To understand this better, we conduct a different experiment. We pick a particular configuration of the sender and receiver such that their channel SNR is roughly 12dB, and we vary the channel airtime budget given to both schemes. For each budget, we let both schemes transmit video and measure the received signal's PSNR. Fig. 9 plots the variation in PSNR vs the normalized channel airtime budget (w.r.t the budget used in the previous experiment). As we can see SoftCast cannot take advantage of increased airtime to improve the video PSNR. The reason is unlike digital systems that can spread information over multiple channel symbols via channel coding to reduce errors, SoftCast is essentially an ana-

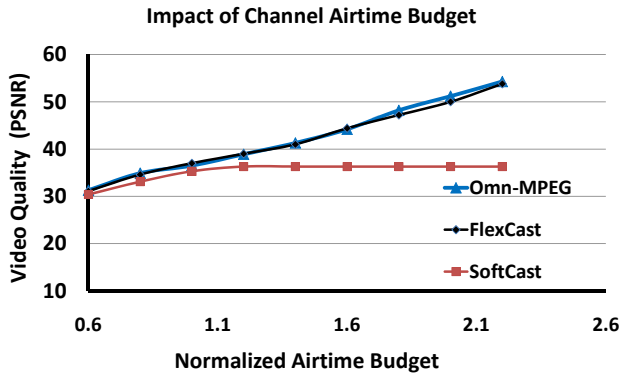

Figure 9: FlexCast performance with varying channel airtime budget. PSNR tracks the airtime budget, but SoftCast cannot take advantage of increased airtime because it is an analog scheme that cannot perform channel coding.

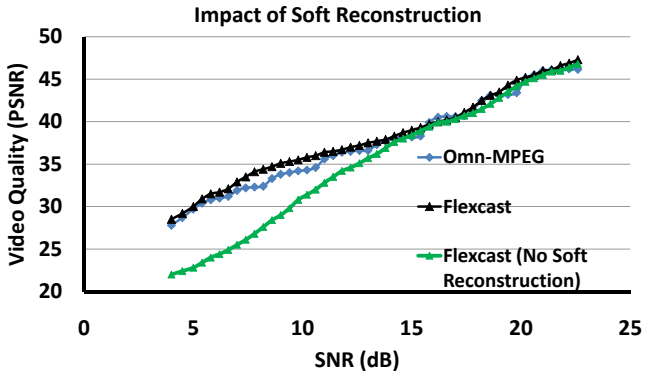

Figure 10: FlexCast performance without soft reconstruction. PSNR drops especially at low SNRs due to high BERs.

log technique that cannot exploit extra airtime via coding even if it is available. For example, if there is a 64 length DCT vector representing a frame, SoftCast creates a 64 length vector of real numbers for transmission, which only require 32 symbols on the wireless channel to transmit. Hence even if the wireless capacity was high and one could transmit much more than 32 symbols in the given video delay budget, SoftCast cannot take advantage of it due to its analog structure.

7.1.2 Impact of Soft Reconstruction

Method: In this experiment, we modify FlexCast to not use SoftPHY hints from the PHY at the client to evaluate the impact soft reconstruction has on FlexCast's performance. The rest of the experiment is conducted similar to the previous experiment. We plot the average PSNR achieved by Omn-MPEG, SoftCast and FlexCast without soft reconstruction vs SNR in Fig. 10.

Analysis: FlexCast performs worse without soft reconstruction, especially at low SNRs. The reason is that at low SNRs we are likely to have lots of bit errors, and not weighting the contribution of a bit by its confidence leads to large distortion. As channel SNR increases, the BER reduces and there is less uncertainty in each bit decision. Consequently the distortion is reduced and approximates normal FlexCast at high SNRs.

7.1.3 Impact of Proportional Representation

Method: In this experiment, we modify FlexCast to not use proportional representation, i.e. each distortion group is given an equal amount of space in the packet regardless of their importance. The rest of the experiment is conducted similar to the above experiments. We plot the average PSNR achieved by the three schemes vs average channel SNR in Fig. 11.

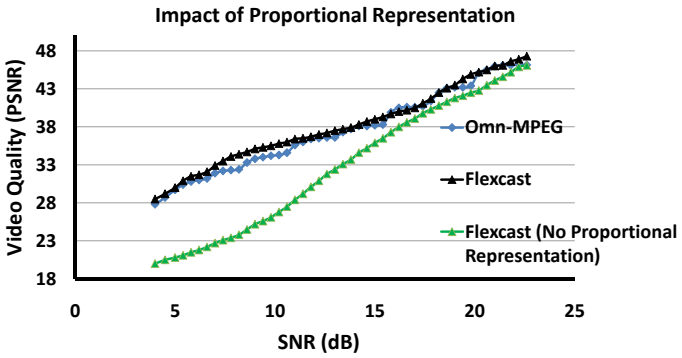

Figure 11: FlexCast performance without proportional representation. PSNR drops especially at low SNRs because its more likely important bits are decoded with low confidence.

Analysis: FlexCast performs worse without proportional representation, again at low SNRs. The reason is that at low SNRs BER is high and it is important to avoid making large errors on the most significant bits. However, without proportional representation all bits are treated equally, hence there is significant distortion due to errors in MSBs. As expected the problem is mitigated as SNR increases, since the BER reduces and the confidence of the MSBs increases, leading to lower distortion.

7.2 Trace Driven Emulation

Although FlexCast can run in real time on a USRP2 connected node, similar to prior work [23] we turn to trace driven emulation to compare FlexCast with Apex, a state of the art video delivery technique that works with modifications to the wireless APs PHY. This is for two reasons. First, as discussed earlier, Apex requires dynamic rate adaptation and MAC protocols operating in the AP's PHY, which cannot be implemented given the timing constraints on the USRP2. Second, we want to compare the schemes over varied channel conditions, from static to rapidly changing to assess how consistently they perform across all scenarios. However, it is hard to generate controllable high-mobility and high-contention in experimental settings. Note that to be fair, Apex is allowed to use a state of the art rate adaptation protocol SoftRate [26] that uses SoftPHY hints for accurate rate adaptation. SoftRate outperforms SampleRate [5], the rate adaptation scheme which Apex originally used in its evaluation.

Trace: **Trace**: We collect real channel information for the simulations via two traces: one for mobility and the other for contention. We use the Stanford RUSK channel sounder [17] to collect channel state information for a 20MHz 802.11 wireless channel. The channel sounder is an equipment designed for high precision channel measurement, and provides almost continuous channel state information over the entire measurement period, and can measure channel SNRs as low as -3dB. Our experiments are conducted at night on the band between 2.426 and 2.448GHz which corresponds to WiFi channel 6, and include some interference from the building's WiFi infrastructure which operates on the same channel. To collect the trace, a mobile channel sounder node is moved at normal walking

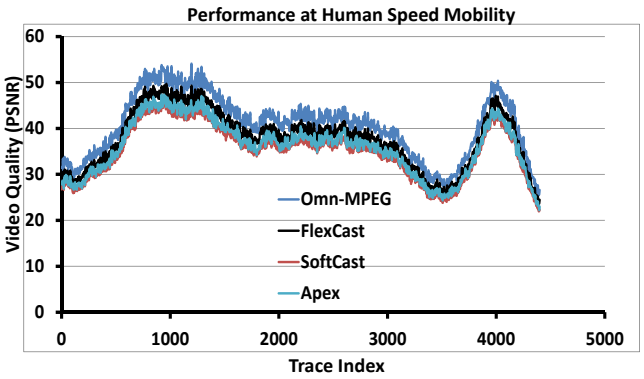

Figure 12: Performance with human speed mobility vs trace index. All schemes perform relatively well.

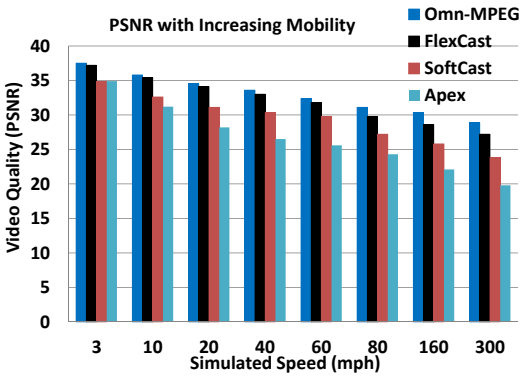

Figure 13: Performance with increasing mobility. FlexCast performs almost as good as the omniscient MPEG scheme, but Apex performance degrades with increasing mobility.

speed (≈3mph) in the testbed and the channel sounder node at the center (the blue node at the center of the testbed figure 7) measures the channel from the mobile node. These nodes record and estimate detailed channel state information for all frequencies in the 20Mhz channel, and therefore include frequency selective fading which we would not have seen with USRP2s that operate on 6.25Mhz bands. We collect around 100000 measurements over a 100 second period, and get a CSI sample every $1ms$ for one trace. We use 10 different walking paths to collect 10 different mobility traces.

Emulator: We feed this trace to a custom emulator written using the the MIT C++ Gnuradio OFDM Code [26] and FlexCast's C++ implementation. For Apex, the emulator implements a 802.11 style PHY augmented with the SoftRate rate adaptation algorithm as discussed before. The other MAC parameters are the same as traditional WiFi, but of course Apex controls how packets are retransmitted and how bits are mapped for an transmission.

Simulating Mobility: To vary mobility, we replay the trace at different speeds. For example, $4\times$ mobility implies the channel measurements that spanned T seconds now span $T/4$ seconds. When a packet is transmitted at time t in simulation, the symbols in the packet are distorted using the corresponding channel measurement from the trace at time t. If the trace has been sped up $4\times$ to simulate mobility, the channel measurement at time t in the new trace will be the channel measurement in the original trace at time $4t$.

7.2.1 *Low Mobility*

Method: We first examine the performance of FlexCast at the normal speed of the trace, i.e. human speed mobility. Note that the Omn-MPEG scheme has advance knowledge of all the channel states that affect each packet transmission, and picks the highest bitrate and video encoding rate that maximizes the PSNR at every instant. The other schemes are implemented as described before. The performance metric is the average PSNR over an interval, smoothed to improve readability. Fig. 12 plots the average PSNR with the trace index.

Analysis: As expected all schemes perform well. At low mobility the wireless channel is largely predictable. Hence rate adaptation algorithms as well as video bitrate adaptation work well. Again SoftCast performs worse (by ≈ 4dB) due to its inability to take advantage of digital channel coding and its reliance on essentially analog transmission.

7.2.2 *High Mobility*

Method: Next, we compare the performance of FlexCast under varying mobility by playing the trace at increasing speeds, from $1\times$ walking speed (3mph) to $20\times$ corresponding to vehicular speeds (60-80mph) to $100\times$ corresponding to 300mph. We run the simulations and compute the average PSNRs achieved by Omn-MPEG, FlexCast, SoftCast and Apex. Fig. 13 plots the average PSNR vs simulated speeds.

Analysis: FlexCast performs relatively well with increasing mobility. At vehicular speeds for example, FlexCast outperforms Apex by nearly 6dB in PSNR and SoftCast by 3dB. FlexCast's gains come from two aspects. First, it does not require accurate rate adaptation at the PHY layer, it can work with incorrect decisions and recover video even from erroneous frames. Second, its video bitrate adaptation is rateless and graceful. Hence even though goodput might be fluctuating constantly, the receiver recovers a video quality corresponding to instantaneous channel quality.

Apex on the other hand performs worse by nearly 6dB. There are two key reasons. First, Apex requires somewhat accurate rate adaptation, since it requires that at least the I-frames (which are mapped to the well protected constellation positions) are decoded without error. If there are bit errors, the I-frame is lost and Apex has to retransmit. Second, in highly mobile scenarios the rate adaptation algorithm picks resilient constellations such as QPSK. For such constellations Apex provides no gain at the PHY layer and reduces to traditional MPEG, though Apex still provides some gains due to MAC layer modifications. Hence overall Apex performance suffers.

SoftCast works slighly better compared to Apex, but is still 4dB worse than FlexCast. SoftCast does not require accurate rate adaptation or video bitrate adaptation, hence it is not that sensitive to wireless link variations. Nonetheless, it performs worse than FlexCast again because of its inability to use digital channel coding techniques, and being forced to live with the raw instantaneous channel quality.

8. CONCLUSION

Current video codec's properties are not compatible with the realities of the harsh wireless channel, hence its not surprising that they do not perform well for mobile video delivery. FlexCast provides a video codec that exploits the unique properties of video as well as wireless channels to deliver graceful performance. FlexCast's design is layered, modular and end-to-end. In the future, we plan to

extend FlexCast to multicast as well as broadcast video streaming. Further, we believe that FlexCast's code design has applications to other Internet video delivery systems such as P2P streaming, exploring them is future work.

9. ACKNOWLEDGEMENTS

This work was supported by a Stanford Graduate Fellowship and a Stanford Terman Fellowship. We would like to thank Aditya Gudipati, Nick Mckeown and Dina Katabi for valuable feedback. Finally, we sincerely thank the anonymous reviewers for their invaluable comments.

10. REFERENCES

[1] Cisco visual networking index. `http://www.cisco.com/`.

[2] ffmpeg. `http://ffmpeg.org`.

[3] libdai - a free and open source c++ library for discrete approximate inference in graphical models. `http://goo.gl/cKmqV`.

[4] Xiph.org test media. `http://media.xiph.org/video/derf/`.

[5] BICKET, J. Bit-rate selection in wireless networks. Master's thesis, MIT, 2005.

[6] BYERS, J., LUBY, M., AND MITZENMACHER, M. A digital fountain approach to asynchronous reliable multicast. In *IEEE JSAC, 20(8):1528âĂŞ1540* (Oct 2002).

[7] CAIRE, G., SHAMAI, S., SHOKROLLAHI, A., AND VERDÃŽ, S. Fountain codes for lossless compression of binary sources. In *IEEE Workshop on Information Theory* (2004).

[8] COVER, T., AND THOMAS, J. Elements of information theory.

[9] GALLAGHER, R. Ldpc codes.

[10] GOYAL, V. K. "multiple description coding: Compression meets the network". *IEEE Signal Processing Mag.* (Sep 2001).

[11] HE, Z., CAI, J., AND CHEN, C. W. Joint source channel rate-distortion analysis for adaptive mode selection and rate control in wireless video coding. *IEEE Trans. Circuits Syst. Video Techn.*.

[12] JAKUBZAK, S., RAHUL, H., AND KATABI, D. Softcast: One size fits all wirless video. In *HotNets 2009*.

[13] JAMIESON, K., AND BALAKRISHNAN, H. Ppr: Partial packet recovery for wireless networks. In *Proc. ACM SIGCOMM* (2007).

[14] LUBY, M. Lt codes. In *Proc. of FOCS 2002* (2002).

[15] MAJUMDAR, A., SACHS, D., KOZINTSEV, I., RAMACHANDRAN, K., AND YEUNG, M. Multicast and unicast real-time video streaming over wireless lans. In *IEEE Trans. Circuits and Systems for Video Technology, 12(6):524âĂŞ534* (2002).

[16] MCCANNE, S., VETTERLI, M., AND JACOBSON, V. Low-complexity video coding for receiver-driven layered multicast. In *IEEE JSAC, 15(6):983âĂŞ1001* (Aug 1997).

[17] N. CZINK, B. BANDEMER, G. V. L. J., AND PAULRAJ, A. Stanford july 2008 radio channel measurement campaign. In *COST 2100* (October 2008).

[18] O. BURSALIOGLU, M. FRESIA, G. C., AND POOR, H. V. "lossy joint source-channel coding using raptor codesâĂİ. *International Journal of Digital Multimedia Broadcasting* (2008).

[19] PEARL, J. Reverend bayes on inference engines: A distributed hierarchical approach. In *Second National Conference on Artificial Intelligence. AAAI-82* (Pittsburgh, PA, 1982).

[20] RAMCHANDRAN, K., ORTEGA, A., UZ, K., AND VETTERLI, M. Multiresolution broadcast for digital hdtv using joint source-channel coding. *IEEE Journal on Selected Areas in Communications, Special issue on High Definition Television and Digital Video Communications* (1993).

[21] RICHARDSON, I. *H.264 and MPEG4*. Wiley & Sons, 2003.

[22] SAID, A., AND PEARLMAN, W. An image multiresolution representation for lossless and lossy compression. In *IEEE Trans. Image Processing, 5(9):1303âĂŞ1310* (Sep 1996).

[23] SEN, S., GILANI, S., SRINATH, S., SCHMITT, S., AND BANERJEE, S. Design and implementation of an "approximate" communication system for wireless media applications. In *ACM SIGCOMM 2010*.

[24] SHOKROLLAHI, A. Raptor codes. *IEEE/ACM Trans. Netw. 14*, SI (2006), 2551–2567.

[25] TSE, D., AND VISHWANATH, P. *Fundamentals of Wireless Communications*. Cambridge University Press, 2005.

[26] VUTUKURU, M., BALAKRISHNAN, H., AND JAMIESON, K. Cross-layer wireless bit rate adaptation. *SIGCOMM Comput. Commun. Rev. 39*, 4 (2009), 3–14.

[27] WU, D., HOU, Y., AND ZHANG, Y.-Q. Scalable video coding and transport over broadband wireless networks. In *Proc. of the IEEE, 89(1):6âĂŞ20* (2001).