Fast Lossy Internet Image Transmission

John M. Danskin Geoffrey M. Davis Xiyong Song

Dartmouth College
6211 Sudikoff Laboratory
Hanover NH, 03755 USA
+1-603-646-2206
{jmd,gdavis,xsong}@cs.dartmouth.edu

ABSTRACT

Images are usually transmitted across the Internet using a lossless protocol such as TCP/IP. Lossless protocols require retransmission of lost packets, which substantially increases transmission time. We introduce a fast lossy Internet image transmission scheme (FLIIT) for compressed images which eliminates retransmission delays by strategically shielding important portions of the image with redundancy bits. We describe a joint source and channel coding algorithm for images which minimizes the expected distortion of transmitted images. The algorithm efficiently allocates quantizer resolution bits and redundancy bits to control quantization errors and expected packet transmission losses. We describe an implementation of this algorithm and compare its performance on the Internet to lossless TCP/IP transmission of the same images. In our experiments, the FLIIT scheme transmitted images five times faster than TCP/IP during the day, with resulting images of equivalent quality.

KEYWORDS

forward error correction, image compression, image transmission, Internet, lossy transmission, World Wide Web.

1 INTRODUCTION

World Wide Web requests are currently estimated to comprise some 25% of all bytes sent over the Internet. This fraction is growing rapidly, and WWW requests are projected to become the single largest consumer of Internet bandwidth in late 1995[7]. Images, most of which are examined for a only a few seconds, undoubtedly constitute the bulk of the 10 terabytes of monthly Web requests. For such interactive applications as web browsers, the responsiveness gained from rapid image transmission is more important than perfect image fidelity, since relatively few images are closely examined, and because many are already distorted by compression.

The usual method for transmitting images over the Internet is to first compress the images using a lossy scheme such as JPEG, and then to transmit them across the intrinsically lossy Internet using the lossless TCP/IP protocol. JPEG and related lossy schemes are very sensitive to bit errors and hence require lossless transmission. The price paid for lossless transmission over a lossy medium is excessively lengthy transmission times due to retransmissions of lost packets. A more efficient means of transmitting the data is via some form of redundant transmission (forward error correction) which will make serious transmission errors unlikely. Redundancy must be applied selectively, however, since the addition of redundancy increases the amount of information to be transmitted.

Lossless transmission schemes are even more problematic for Internet video broadcasting. Retransmission is impractical with broadcasting because the receivers will not in general experience the same losses. A broadcaster attempting to respond to all of these different losses will quickly be overwhelmed. Again, what we need to cope with packet losses is some form of forward error correction.

Some of the bits resulting from JPEG compression and other schemes are not perceptually as important as some of the other bits. Flipping high order bits in the DC channel makes a big difference. Flipping low order bits in the fine detail channel makes almost no difference. Protecting these bits equally is not efficient.

FLIIT uses a subband compression scheme with a Lagrange multiplier technique to allocate bits to coding subband data and to forward error correction. Bits devoted to image data give us an image to transmit,

and bits devoted to forward error correction increase the likelihood of the image arriving intact. FLIIT allocates bits from a given budget so that image distortion, resulting from the combined effects of compression and transmission losses, is minimized. The result is that forward error correction bits are concentrated in subbands where losses would be catastrophic, while less important subbands receive less protection.

FLIIT also addresses network issues such as rate and congestion control, and startup. FLIIT allocates a fixed number of bits between redundancy and data depending on the expected loss rate. When the loss rate is high, bits are shifted from data to redundancy, but the total number of bits transmitted remains constant. During heavy congestion when packet loss rates are high, TCP retransmits more and more packets. This positive feedback worsens congestion. FLIIT can avoid this positive feedback by sending packets exactly once using a fixed total number of bits, trading quantizer resolution for forward error correction according to current network conditions.

The buffer capacity of the Internet between any two well separated nodes is usually greater than the size of a well compressed image. Sending at a rate appropriate for such a connection, the server could transmit the entire image in less than one round-trip time. This is impossible with TCP because TCP starts up each connection slowly, taking many round trips to get up to full speed. This is fine for megabyte transfers, but inappropriate for 8 kilobyte images. We propose that the FLIIT server remember effective transfer rates across connections, effectively removing slow startup as an issue.

An important issue for lossy protocols is determining when to stop waiting for data packets that may have been lost or delayed. If we don't wait long enough, we lose image data. On the other hand, if we wait too long, we lose responsiveness. We describe a method for incorporating the optimization of the waiting time into our resource allocation algorithm.

1.1 Related Work

A number of strategies have been explored for incorporating redundancy into network packets. Turner and Peterson [16] propose a scheme in which errors are corrected by making use of naturally occurring redundancy within images. Image pixels are reordered for transmission in such a way that packet losses cause the loss of isolated pixels rather than large contiguous blocks of pixels. Missing pixels are reconstructed by applying a filter to their surviving neighbors. This technique can hide a limited number of missing packets since there is usually a high correlation between neighboring pixels. A network video transmission scheme proposed by Karlsson and Vetterli [8] also makes use of naturally occurring image redundancy for error correction. Using intrinsic image redundancy to correct losses is problematic, since the number of losses that can be sustained is highly image dependent. Furthermore, when efficient compression schemes are used, very little usable redundancy remains for error correction.

Better control of transmission errors is obtained by adding redundancy bits to the bitstream rather than relying solely on naturally occurring redundancy. Biersack [3] evaluates the effect of adding redundancy at a fixed rate to video transmissions over ATM networks. As we discuss below, this fixed rate addition of redundancy is inefficient, and indeed [3] obtains mixed results. For heterogeneous traffic scenarios the loss rates were reduced by several orders of magnitude, but for more homogeneous traffic the performance was unchanged or worsened. In the homogeneous scenarios, the increase in the network load from the transmission of redundancy bits caused an increase in the loss rate not compensated for by the error correction.

A more effective method of adding redundancy is joint source/channel coding. Fixed redundancy rate strategies are inefficient for image transmission because bits in the compressed image do not all have the same effect on image quality. Joint source/channel coding schemes such as the FLIIT scheme presented in this work assign levels of redundancy to portions of the compressed image according to their relative contributions to image fidelity. Our experiments show that the control of redundancy achieved by FLIIT yields substantial improvements in image quality over non-adaptive schemes.

Similar techniques of joint source/channel coding for continuous bitstreams have been developed in Tanabe and Farvardin[14]. The error calculations for these continuous streams are extremely difficult, and the algorithms presented rely on computationally expensive simulations during bit allocation. FLIIT's network packet implementation uses a simple and fast allocation scheme.

A related source/channel coding scheme for networks, Priority Encoding Transmission (PET), has been developed by Albanese *et al* [1][9]. The implementation of PET for MPEG allows the user to set different levels of error protection for different portions of the MPEG stream, but unlike FLIIT provides no explicit mechanism for allocating these levels. The level of redundancy in PET affects the network packet size, so in some networks PET may have less flexibility than FLIIT in specifying a level of redundancy.

The layered transmission schemes in Garret and Vetterli [6] and Posnak *et al* [11] also make use of joint source/channel coding ideas. Layered schemes require networks which treat packets differently according to their priorities. Visually important data is sent with a high priority and experiences a small loss rate; less important data has a low priority and is the first to be

discarded by switches during congestion. Our FLIIT scheme, in contrast, requires no prioritization of packets by the network and can function on any network supporting a simple datagram protocol.

The contribution of our FLIIT scheme is to provide a simple, low-complexity mechanism for obtaining a near-optimal partitioning of bits between image quantization and redundancy for a given set of image transform quantizers, a parity protection scheme, and a packet loss model. We describe in detail an implementation of the algorithm, demonstrate its effectiveness for image transmission on the Internet and its robustness under heavy loss conditions.

1.2 Outline

Section 2 describes the subband compression scheme we use. Section 3 outlines the allocation bits for data representation and protection. Section 4 gives details for implementation. Sections 5 presents experimental results, and Section 6 concludes with a discussion.

2 IMAGE COMPRESSION

2.1 Subband Coding

We use a simple wavelet transform coding scheme to compress images for transmission. We perform a discrete wavelet transform on an image, quantize the coefficients using uniform quantizers, and entropy-code the resulting coefficients using an arithmetic coder. The resolution of the quantizers is determined by a Lagrange multiplier procedure we describe below. We use the 7/9-tap biorthogonal wavelet from [17] for our experiments below.

The FLIIT scheme can easily be modified to work with DCT-based schemes such as JPEG. We have chosen to work with wavelet-based coder because of its simplicity and superior performance. Our low-complexity scheme yields PSNR's for the 512×512 Lena image within 0.3 to 0.9 dB of state of the art coders [12].

2.2 Bit Allocation

The discrete wavelet transform partitions an image into a set of subbands ranging from fine scales (high frequency) to coarse (low frequency). Typically the bulk of the visually important information is concentrated in the coarse-scale subbands, with the fine-scale subbands contributing mostly at sharp edges. We obtain a compressed image by finely quantizing coefficients that contribute heavily to image fidelity and coarsely quantizing others. Determining the resolutions for each subband is basically a problem of resource allocation. We have a tradeoff between quantization error and total storage requirements, and we must allocate quantizer bins to obtain minimal distortion for our given bit expenditure.

Let $D_j(q)$ be the sum of the squared errors incurred in quantizing each wavelet coefficient in band j to q bits, and let $C_j(q)$ be the cost of storing the entropy-coded quantized values. We use the mean squared error as our measure of distortion to allow comparison to competing algorithms, but the scheme we describe will function equally well with perceptually weighted metrics such as that of [10]. For an image decomposed into n subbands, our goal is to find a vector $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ of quantizer bin allocations which minimize the total distortion $D_{total}(\mathbf{q}) = \sum_{j=0}^{n} D_j(q_j)$ subject to the constraint that the total cost in bits, $C_{total}(\mathbf{q}) = \sum_{j=0}^{n} C_j(q_j)$ must be less than or equal to some given bit budget C_{max}. In addition, the components of $\mathbf{q}$ must all be positive integers, and for practical reasons we impose an upper bound on the components of $\mathbf{q}$. Thus we seek a minimization over $\mathbf{q} \in Q$ where Q is some set of valid integer-valued quantizer bin allocations.

Shoham and Gersho [13] describe an algorithm which solves precisely this problem. They show that any unconstrained minimum of $C_{total}(\mathbf{q}) + \lambda D_{total}(\mathbf{q})$ is also the solution to a constrained problem of the form we require. These unconstrained problems are much easier to solve, but we must determine which value of λ yields the appropriate constrained problem. The constrained problem is thereby transformed into a search through a family of unconstrained problems. The algorithm in [13] gives optimal or near-optimal bit allocations for our problem.

In our implementation, we use a uniform quantizer for subband coefficients. The limits of the quantization are determined by the range of the coefficients, and the resolution is taken from the set $\{2^k - 1\}_{0 \le k \le K}$ for a fixed positive K. We know *a priori* that the wavelet coefficients will be distributed roughly symmetrically around 0, so we restrict ourselves to odd numbers of quantizer bins to ensure that the very frequent near-zero coefficients will not be divided into two bins. We find in numerical experiments that allowing q_j to assume intermediate values does not give a substantial improvement in performance. For a $2^N \times 2^N$ image, the minimization algorithm iteratively searches through an array of size $(3N + 4)K$, and convergence is typically achieved in roughly 30 iterations. The complexity of the allocation algorithm is quite small with respect to that of the transform and the entropy coding.

3 CHANNEL CODING AND EXPECTED IMAGE DISTORTION

3.1 Joint Source-Channel Coding

The goal of our compression scheme is to minimize the image distortion incurred from quantizing transform coefficients. Transmission of an image over a network introduces a second source of distortion: network packet losses. Our compression scheme controls quantization error by adaptively allocating quantizer resolution. In

the same way, we can control packet loss errors by selectively adding redundancy to our bitstream.

Traditionally the costs and distortions associated with quantization and transmission have been treated separately. This separation is motivated by Shannon's joint source channel coding theorem [5], which states that source coding followed by channel coding can be made to be as good as any single-stage source/channel coding procedure. Shannon's result (which is asymptotic and requires infinite computational resources) is not applicable in our case, since it deals with the problem of transmitting an infinite bitstream *losslessly*. In fact, as our experiments show below, separate source and channel coding is inefficient for lossy transmission.

We have already incurred loss during compression and are willing to accept a little more during transmission, provided that we lose bits which are visually less important. The channel coder needs access to the source to know the relative values of the bits. We can very easily combine both quantization and transmission errors into our cost and distortion functions above and use our allocator to find an efficient distribution of quantizer resolution and redundancy bits.

The problem we address is that of transmitting images as a collection of packets of bits of a maximum size K over a network. The class of network protocols we consider has two important properties.

- Packets may be delivered out of order, so each packet contains a unique identifier.
- The contents of all packets are verified during transmission.

Packets are lost for one of two reasons: a node somewhere on the network runs out of buffer space and drops the packet, or the packet is corrupted and fails a verification procedure somewhere in transit. Because of the first property, we know exactly which packets have been lost. Because of the second property, we can assume that all packets which are delivered are error-free because they have passed the protocol's verification procedure.

We add redundancy to our transmission by adding parity bits to our data stream at a rate which depends on the current subband. Since we can tell which packets were lost, a single block of parity bits can protect a group of any number of blocks of data against single-packet loss. Given a group of data blocks, a parity block is set to the exclusive-or of the blocks in the group. Any single lost block can be reconstructed as the exclusive-or of the survivors. There is a tradeoff between protection and cost. We can obtain greater protection of data by decreasing the size of the groups protected by parity blocks, but this increased protection comes at the price of having to transmit the additional parity blocks. Our goal is to allocate quantizer bins and parity blocks to subbands so that we minimize the *expected* distortion of the image.

In practice, network packet losses occur in bursts. We send our packets in random order so that the probability of packet losses within parity groups can be approximated as being independent (supposing that the expected length of a burst is less than the size of a transmission). For our optimization, then, we assume that packets are lost independently with a known probability p_{loss}. We can control the variance of our distortion by adjusting the probability of loss assumed by the optimizer. Increasing the assumed probability of loss beyond the network's true packet loss rate has the effect of shifting bits from data to redundancy, thus increasing the quantization distortion at a given bit-rate, but also increasing the degree of protection from lost packets. Since the only source of variance in the expected distortion is lost packets, increasing the degree of redundancy will reduce the variance and increase image consistency.

We emphasize that parity-protection is not the only form of redundancy which will work with our optimization scheme; we have chosen it for simplicity. More sophisticated coding techniques which offer a better ratio of unrecoverable loss rate to coding overhead exist. These schemes will still fit into the FLIIT expected distortion framework because there will always be a tradeoff between coding overhead and the unrecoverable loss rate that can be exploited by the bit-rate allocator.

3.2 Expected Distortion

Data blocks are grouped together in one of three ways. Multiple data blocks are shielded by a single parity block, a single data block is replicated multiple times, or data blocks are unshielded. Consider a subband consisting of n data blocks. Let D_q be the average quantization error incurred per block, let D_m be the error incurred in replacing all coefficients in a data block by the quantized subband mean, and let D_z be the error incurred in replacing all coefficients in a data block by zero. We have $D_q \leq D_m \leq D_z$, so that zero-replacement is the worst-case scenario.

Data and parity blocks from each subband are distributed so that no two blocks from the same band are contained in the same network packet. Hence losses of data blocks are independent events. Every successfully transmitted or lost-but-recoverable block in a data group produces an error of D_q on average. If the subband mean is available (i.e. if at least one packet from the group is successfully transmitted), lost blocks produce an average error of D_m; otherwise lost blocks produce an average error of D_z.

The expected distortion for a band consisting of n data blocks is

$$E(D) = nD_q \quad + \quad np_{unrecoverable}(D_m - D_q)$$

$$+ \quad np^{n}_{unrecoverable}(D_z - D_m) \quad (1)$$

where $p_{unrecoverable}$ is the probability of an unrecoverable packet loss. For unshielded data blocks, $p_{unrecoverable}$ is simply p_{loss}, the probability of losing a packet. In the case of replicated blocks, a loss will be unrecoverable only if all copies are lost. Hence, in this case $p_{unrecoverable} = p^{m}_{loss}$, where m is the total number of copies of each block transmitted. For subbands in which each k blocks are shielded by a single parity block, the probability of losing any single block unrecoverably is $p_{unrecoverable} = p_{loss}[1-(1-p_{loss})^{k}]$, the probability of losing that block and at least one of the other k blocks. In our packet loading scheme we impose the restriction that no 2 blocks from the same subband may occupy the same packet, so block losses are truly independent.

Let r_j indicate the manner of parity shielding for the subband j. We replace the cost and distortion functions $C_j(q_j)$ and $D_j(q_j)$ with the functions $\hat{C}_j(q_j, r_j)$ and $\hat{D}_j(q_j, r_j)$ which incorporate the cost of the parity packets and the expected distortion incurred in transmission. The new cost function $\hat{C}_j(q_j, r_j)$ will equal the old $C_j(q_j)$ plus the number of bits used for the parity blocks. The new distortion function $\hat{D}_j(q_j, r_j)$ is obtained from one of the three expected distortion functions derived above. We now solve the constrained minimization problem as before, using these new cost and distortion functions.

4 IMPLEMENTATION

4.1 Encoding

A description of the encoding process follows:

1. Apply a wavelet subband decomposition to the image. In a complete wavelet decomposition, the coarse-scale subband would be a single pixel value corresponding to a weighted average of all pixels in the image. Because of the necessity of maintaining some header information with each subband, it is cheaper to stop the transformation at some point short of a single pixel (say a 32×32 coarse-scale image), and to transmit this image untransformed. This base image is referred to as the coarse scale band, or band 0. It corresponds roughly to the DC band of a JPEG image. The other (detail) bands are refinements of this image, each successive band providing the information necessary to double the image resolution.

2. Assign quantizer redundancies and levels of parity according to the algorithm described in the previous section.

3. For transmission, it makes sense to distribute each band across as many packets as possible using a pixel interleaving scheme like the one described in [16] so that a lost packet will not cause a catastrophic band loss. Distributing subbands does not reduce expected distortion (because the chance of some loss in a given subband is *increased* by distribution), but it reduces the variance in the expected distortion by increasing the population subject to the transmission experiment.

   This desire for subdivision is tempered by the need for a descriptive header for each independent image block. Since image transmission is lossy, and any subset of network packets could arrive, we added a header to each block which describes the block in enough detail to permit (lossy) recontruction of the subband corresponding to the block.

   We settled on image blocks of up to about 150 bytes each. Header information worked out to about 15 bytes per block, but because there were many small subbands that were not broken into blocks, roughly 20% of the compressed image ends up as header information.

   A future implementation might localize this header information in a few heavily shielded packets, which would be more efficient since much of the header information is replicated between blocks from the same subband.

4. The compressed wavelet coefficients follow the header in the compressed block. The wavelet coefficients are compressed using adaptive arithmetic coding. Arithmetic coders emit $\Sigma - \log_2 p_i$ bits where p_i was the predicted probability of the ith event. In adaptive coding, the relative frequencies of past events, as remembered in histograms, are used to estimate the probability of future events for the purposes of coding. So that no event is predicted with zero probability, histograms are usually initialized so that all possible events have frequency one. As the input is read, the histogram adapts to the actual frequencies encountered. For a large dataset, the inertia represented by the initial flat histogram is unimportant.

   By blocking, we have reduced the amount of time available for the histogram to adapt to the input distribution. To compensate for this effect, we use the following scheme:

   - Initially, there are two histograms, one flat histogram (F) with every possible value initialized to one, and one empty histogram (H) with a single symbol, the escape symbol with initial probability and frequency equal to 1.
   - Whenever an input symbol appears with nonzero probability (frequency) in histogram H, it

is coded using histogram H, and its frequency in histogram H is incremented.

- Whenever an input symbol appears with zero probability in histogram H, the escape symbol is coded using histogram H, and the symbol is coded using histogram F (in which it must appear). This new symbol is added to histogram H with frequency 1, and histogram F is never again used to code this symbol.

This two histogram scheme adapts much more quickly than the single histogram scheme. Similar schemes are used for blending high order contexts in text coding [2], but we are unaware of a prior use of this scheme in image coding.

5. After the compressed blocks are generated, we add redundancy. The blocks are sorted by protection level and size, in order of decreasing protection and size. Blocks requiring replication are simply replicated. A given block and its replicas will all have the same parity group number, which will prevent them from being included in the same network packet.

   Blocks requiring the same level of parity protection will be grouped together by the sort. Usually it will not be possible to meet the requirements exactly. For instance, there might be just 4 data blocks which want to be in a group of 5 data blocks protected by a parity block. In this case, we use a greedy algorithm to promote a block with less stringent protection requirements (if one is available) to round out the group. This promotion procedure is more efficient than the alternative of leaving parity groups unfilled, and effectively promoting all of the members of a group to a higher level of protection.

   Sorting by size helps to keep similarly sized blocks in the parity group, which is valuable because the parity block must be as large as the largest data block in the group.

6. Finally, we have compressed data and parity blocks ready for transmission. If (when) the network is congested, throughput will be gated by router scheduling, rather than by bandwidth. Routers schedule communication channels using a round robin algorithm which is insensitive to packet size. This means that users of packets smaller than the largest packet transmitted by the network will pay a heavy throughput penalty for their poor judgement. Conversely, if a user sends such large packets that they are fragmented en-route, then they will lose their entire large packet whenever a single fragment is lost, also resulting in reduced throughput.

   The largest Internet packet which is guaranteed not to be fragmented is 576 bytes [15].

   We use the largest-first first-fit heuristic to pack blocks into 550 byte UDP packets. Additional restrictions are that blocks from the same parity group are not allowed in the same packet, and blocks from the same band are not allowed in the same packet.

4.2 Decoding

Decoding is essentially encoding in reverse, except that all the bit allocation decisions have been made, and some of the packets may have been lost.

1. Read the surviving packets.
2. Sort into parity groups.
3. If there are any parity groups with one missing member, reconstruct the missing member.
4. Decode all of the data blocks into their respective subbands. When decoding a coarse band block, if it is the first one, fill in the whole band with values from this block so that if any other blocks from this band turn out to be missing, their values will be replaced by nearby values from this block. Missing detail band values are replaced by the subband means.
5. Reconstruct the image.

4.3 Rate Control

The Internet is a shared medium. Programs have to control the rate at which they send data or they risk causing congestion in the network, which results in lost packets, and reduced performance for everybody. The TCP protocol implemented in the RENO release of BSD Unix controls its rate by starting out very slowly, slowing down when packets are dropped (indicating congestion), and speeding up otherwise. The problem with this strategy is that it induces a certain level of packet losses on the Internet.

A second method for rate control is to compare expected throughput rates with actual throughput rates. Whenever the rate of packet reception drops below the rate of packet transmission, the network must be storing or dropping the excess data. This is the strategy implemented in Brakmo *et al*'s TCP Vegas protocol [4].

Neither of these rate control schemes is appropriate "as is" for the transmission of compressed images across the Internet. The problem is that compressed images are much smaller than the dataset size required to achieve steady state transmission. In [4] we see a delay of 2.5 seconds before steady state and heavy packet

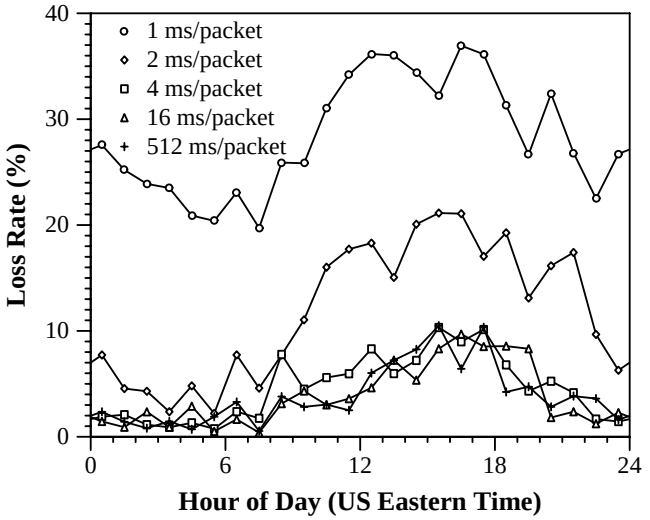

Figure 1: This graph plots the Internet packet drop rate, for packets sent from Dartmouth College to a Stanford University echo server and back, as a function of time of day. Each curve corresponds to a different transmission rate. Packets contained roughly 550 bytes each. 2ms per packet was an effective transmission rate between 03:00 and 04:00 EDT, but generates high loss rates when the net becomes busy during the day. During the day, the loss rate never drops much below 5% no matter how slowly data is transmitted.

losses at around 750 ms because the TCP RENO overshoots the channel's actual throughput by a factor of two at the end of slow-startup.

For the network experiment in this paper, we used an offline process to choose a fair transmission rate for FLIIT packets. This rate was chosen by picking the knee on the load/loss curve [18]. Streams of packets containing roughly 550 bytes were sent at various transmission rates, and the loss rate was measured for each rate. As can be seen in Figure 1, the loss rate as a function of transmission rate was roughly constant at rates below about 4ms per packet. Above 4ms per packet, the loss rate increased sharply. We chose to run our experiments using a transmission rate of 4ms per packet to avoid high loss rates, and to avoid impacting other applications.

In future work, we plan to incorporate an automatic Vegas-like rate control strategy into our image server. The main difference in rate control from TCP-Vegas would be that rate control information would be retained for each network address between image transmissions. This would eliminate startup effects from image transmission, except for the first image sent to a given site.

4.4 Stopping Criterion

FLIIT packets may be lost or delayed for long periods of time. If we wait too long for slow packets, we lose responsiveness. If we don't wait long enough, we lose packets. This tradeoff between transmission speed and packet loss has an elegant resolution: we can incorporate the tradeoff into our resource allocation algorithm and choose an optimal stopping point.

The server sends packets at a constant rate, which we assume to have been chosen in some pro-social manner conducive to keeping congestion down. If we send a packet every b time units, where b is chosen to be less than or equal to the throughput of the network, and the network delivers all packets after a fixed delay, then the n-th packet will arrive at time $T_n = a + (n-1)b$. Here a is the arrival time of the first packet. On the Internet, packets are delayed by variable lengths of time. We can incorporate this variability into our arrival time model by adding a random delay variable X_n. Now we have $T_n = a + b(n-1) + X_n$. Our goal is to find a stopping time T_{stop} after which we stop waiting for packets and reconstruct our image.

Packets arriving after time T_{stop} will be considered lost. Given the distributions of the X_n, we can determine $P(T_n > T_{stop})$, the probability that packet n will be lost due to excessive delay. We have randomized the order of the packets we transmit, so the probability of any given packet being the n-th packet transmitted is $\frac{1}{N}$, where N is the total number of packets sent. The probability of a particular packet being lost due to delay is $p_{delay}(T_{stop}) = \frac{1}{N}\sum_{k=1}^{N} P(T_n > T_{stop})$. The overall probability that a packet is lost, then, is $p_{loss} = 1 - (1 - p_{drop})(1 - p_{delay})$ where p_{drop} is the probability of the packet being dropped in transit.

We see that our choice of stopping time thus affects the loss rate observed by the receiver. The reconstructed image distortion is a function, then, not only of the number of data and redundancy bits, but also of the stopping time. Because our constraint is on the number of bits sent, and not on the length of time required to receive the image, the optimal value of T_{stop} is infinite. If our goal is to maximize responsiveness, we need to constrain the time required to receive the image rather than the total number of bits we send. We can do this by setting our cost function to be the sum of the time required to send the bits in the image plus the time spent waiting. The result is that we obtain a new set of cost and distortion functions which depend on the bit allocations as well as the stopping time. By varying the stop time in our allocation algorithm, we can obtain jointly optimized bit allocations and a stopping time.

In our experiments we model the delays X_n as a set of independent, identically distributed Poisson random variables with parameter λ. Figure 2 shows that the

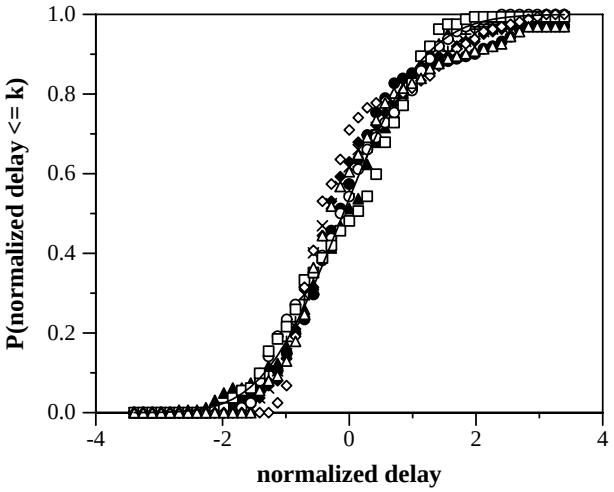

Figure 2: Observed and fitted cumulative density functions for the packet delays X_n. The data was gathered from ten 160-packet transmissions. Offsets and sending rates a and b were determined by least squares so that the delay X_n could be isolated. The resulting delay was normalized to have mean 0 and variance 1. The superimposed solid curve is the cdf for an equivalently normalized Poisson random variable.

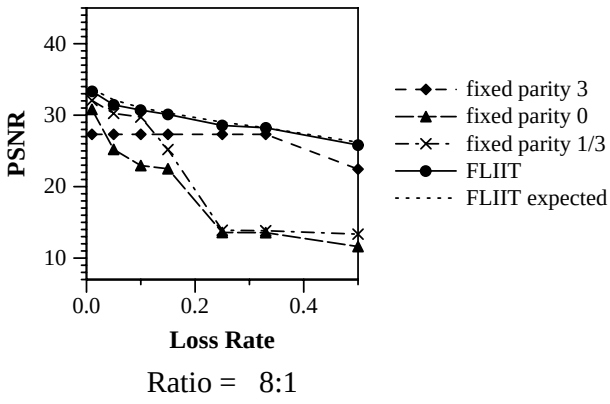

Figure 3: The expected and measured performance of FLIIT, and 3 fixed parity schemes. Fixed parity 3 means that data blocks are always replicated 3 times. Fixed parity 0 means there was no redundancy. Fixed parity 1/3 means that there is one parity block for every 3 data blocks. The Y axis is the peak signal to noise ration (PSNR), a logarithmic indicator of image quality, and the X axis is the expected loss rate. FLIIT dominates the other schemes, usually by several dB.

model does a good job of describing the distribution of delays. Although the assumption of independence is certainly too strong, it does not appear to affect our results significantly. The offset a and sending rate b can be determined by the receiver via least squares and the parameter λ via the method of moments. The server can update its knowledge of network conditions by periodically obtaining these quantities from the receiver. In our experiments, we found typical stopping times to be the expected time of arrival of the last packet, $a+b(N-1)+\lambda$ plus a delay ranging from 0 to $\sqrt{\lambda}$, the standard deviation of the delay.

5 EXPERIMENTS

We present two experiments. In the first, we compare uniform vs. non-uniform distribution of redundancy in forward error correction with simulated network losses. In the second experiment we compare image quality as a function of transmission time for FLIIT and TCP, using a real network.

5.1 Uniform vs. Non-uniform Forward Error Correction

5.1.1 Experiment

Using the well known Lena image at 256×256 resolution, we generated sets of packets using the FLIIT algorithm, as well as three different fixed-parity schemes. The fixed-parity schemes used the same bit allocator as FLIIT in order to determine quantizer resolutions for each subband, but no adaptive coding was done for the parity bits. Our experiments will therefore show only the effects of adaptive versus fixed distribution of redundancy. In one fixed scheme each data block was replicated 3 times (fixed parity 3), in another scheme groups of three data blocks were protected by a single parity block (fixed parity $\frac{1}{3}$), and in the last scheme no parity blocks were used (fixed parity 0).

Packets were generated using each scheme using 8:1 compression, and with expected loss rates ranging from 0% to 50%. For each combination of parity scheme, compression ratio, and loss rate we ran simulated transmission experiments in which packets were deleted by subjecting each packet to an independent pseudo-random Bernoulli trial. Images were then reconstructed from the remaining packets, allowing image comparisons and calculations of actual image distortions.

5.1.2 Results

Figure 3 displays the results of our experiments. As can be seen, the FLIIT scheme has the overall best performance for all loss rates tested.

The fixed parity 3 scheme (each block is assigned 3 copies) performs best for high loss rates because of the large amounts of transmitted redundancy. At high loss rates, FLIIT also uses large amounts of redundancy, but it distributes these redundancy bits more selectively

Figure 4: We ran 400 experiments at 8:1 compression with expected 50% packet loss. From the left, the images represent the 90th percentile, 50th percentile and 10th percentiles of reconstructed image quality. This is a very severe test: images are reduced to roughly 9Kbytes (18-20 packets with overhead) and then packets are randomly eliminated in independent trials, so that often well under 50% of the packets survive.

than the fixed scheme. In particular, FLIIT shields the low-frequency portions of the image especially heavily, since for the Lena image (and in general) the loss of a low frequency data block results in a much larger error than the loss of a high frequency block. The extra shielding is inexpensive, since there are relatively few low frequency coefficients.

Figure 4 shows the effects of compression and transmission losses on the 256×256 Lena image under FLIIT. and the fixed parity 3 scheme. These images have been compressed from 64K to 9.5K (8:1 compression plus a roughly 20% packet header overhead cost), including the parity blocks, and all packets have a 50% probability of being lost. In effect, these images have been reconstructed from 4K of randomly selected data.

These data show that FLIIT performs well even at very high error rates.

5.2 FLIIT vs. TCP

5.2.1 Experiment

In this experiment, we measured image quality (PSNR) as a function of transmission time for both FLIIT and TCP. The clock starts when a client requests an image, and ends when the client decides that it has received an image. We did not include decode time which is the same for both clients, and nearly negligible in any case. For FLIIT transport, the client makes its request with a single UDP packet. For TCP transport, the client makes its request over a TCP connection. FLIIT images are returned using UDP. TCP images are returned using TCP.

The TCP images were generated using the same compression routines as the FLIIT images, but there was no redundancy or blocking, eliminating all of the overhead which FLIIT needs for reconstruction after lossy transmission, but which are unnecessary after lossless transmission.

As discussed above in Section 4.4, the FLIIT client calculates a running estimate of the expected time of arrival of the last packet. The client waits some period beyond this time, typically one standard deviation of the interpacket arrival time, and decodes the image. The exact amount of extra time to wait is calculated and specified by the FLIIT server.

The TCP client stops when the complete image has arrived.

We used a real Internet connection for this experiment. The connection was between Dartmouth College in Hanover New Hampshire, and Stanford University in Stanford California. The participating computers were separated by 20 hops. For convenience, we ran the client and the server locally, but sent the data across the continent, by routing network packets from our local client to a local pseudo server, which bounced these packets off of the Stanford machine's echo server, and forwarding the returning packets to our local server. Traffic from the server to the client was also similarly redirected through the remote echo server. Figure 5 illustrates this setup.

The image used was again, Lena at 256×256 resolution. We transmitted Lena at different compression ratios, 160 times for each sample.

We ran the experiment under two different sets of circumstances: daytime and nighttime, both on weekdays. Daytime was 12:00-18:00EDT. Nighttime was 02:00-08:00EDT.

We set the expected loss rate, expected packet arrival rate, and standard deviation of interpacket delay to 1.3%, 4.4ms per packet, and 10.4ms for the night

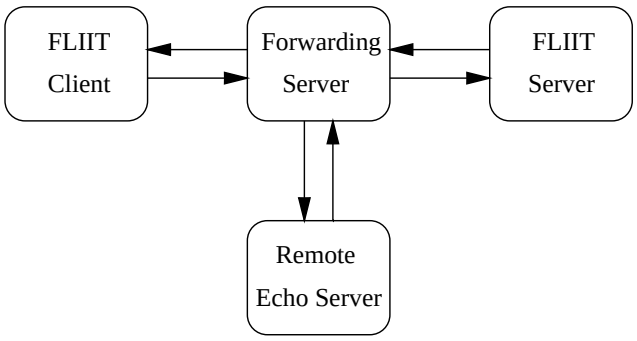

Figure 5: Packets (FLIIT) or stream data (TCP) originating from a local client at Dartmouth College is forwarded through an echo server at Stanford University, and then to the local server. Transmissions from the server to the client are forwarded similarly.

experiments, and 8.2%, 4.6ms per packet, and 12.3ms during the day.

5.2.2 Results

The results of this experiment are graphed in Figure 6. FLIIT uniformly outperformed TCP for equivalent image quality. Tiny FLIIT images were transmitted over twice as fast as their TCP counterparts, presumably because fewer round trips are necessary to establish a FLIIT connection. Larger images were transmitted more quickly because FLIIT does not retransmit dropped packets, slow down when packets are dropped, or wait multiple round trip times for the last few packets. During the day, when the Internet is congested, FLIIT is more than five times faster than TCP, even for high quality images.

FLIIT accepts some variance in quality for a tremendous improvement in performance and almost complete elimination of the multisecond variance in time accepted by TCP. TCP makes the right tradeoff for applications requiring perfect transmission. FLIIT makes the right tradeoff for interactive and real-time applications.

6 COMPUTATIONAL COMPLEXITY

FLIIT encoding and decoding both run in linear time in the size of the input image. Encoding requires generating a constant size set of bit-rate/distortion pairs, which is relatively expensive. Decoding requires only a wavelet transform which is cheap. In practice, the 256x256 image used in our experiments was encoded in 2.0 seconds, and decoded in 0.4 seconds on a DEC ALPHA 3000. These delays, especially the decoding time, are small compared to the extra delays experienced by TCP transmitted images, and TCP transmitted images need decompression too.

There are a number of performance optimizations remaining which we have not pursued at this point: a

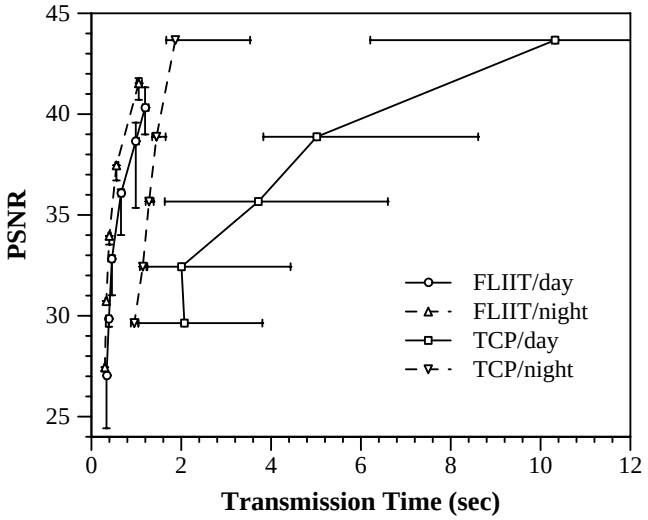

Figure 6: This graph plots image quality (Peak Signal to Noise Ratio, a logarithmic function of the mean squared error), as a function of transmission time. Plotted points correspond to median values, while error bars indicate first and third quartiles. TCP curves have error bars only in the time dimension because they deliver consistent quality. FLIIT has error bars in both dimensions because both quality and time are variable. For equivalent quality, FLIIT is roughly twice as fast as TCP at night, and five times faster than TCP during the day. FLIIT has almost no variation in transmission time, while TCP transmission times vary widely, especially during the day.

DCT could be substituted for the wavelet transform. A faster modified Huffman coder similar to JPEG's could be substituted for our arithmetic coder. Matrices of encoding parameters coupld be computed offline, allowing online encoding to run as fast as decoding. We are confidant that FLIIT is, or can be made to be, appropriate for CPU performance limited applications.

7 DISCUSSION

We have demonstrated a technique which combines source and channel coding, as well as an appreciation of Internet characteristics, producing an image transfer protocol which transfers images of a given quality twice as fast as the TCP protocol at night, and five times faster than TCP during the day.

The FLIIT technique is appropriate for image previewing, progressive image transmission, transmission of moving pictures, and broadcast applications, although some work would be necessary to efficiently integrate FLIIT into a moving picture broadcast system such as MPEG.

One current concern is that FLIIT may be achieving some of its stellar daytime performance by being

more aggressive than TCP connections. In our experiments, FLIIT didn't increase the packet loss rate on the network, but it did not slow down when packets were dropped, while TCP connections did. On the other hand, TCP (RENO) connections force dropped packets at the end of slow every second or so when they overshoot the available bandwidth, so it isn't clear who is being a worse network citizen. The current implementation of FLIIT should coexist gracefully with TCP Vegas implementations, because neither protocol will force dropped packets.

8 FUTURE WORK

As we mentioned above, we plan to implement an automatic rate control procedure similar to TCP Vegas's in the near future. Our procedure would differ from the Vegas's in that we would retain throughput estimates for network destinations between connections, mostly eliminating slow startup. Vegas, and all other TCP implementations we are aware of, derive network throughput information for each connection from scratch, forfeiting reasonable performance for short sessions. We believe that the retention of network throughput information between TCP connections would improve the performance of the very short sessions typically generated by World Wide Web clients.

We are currently implementing a FLIIT image server and a client that can be invoked by a generic web viewer so that WWW users can use FLIIT to access images stored in our server.

We are also interested in better measures of perceptual error, continuous lossy transmission, and improved channel coding techniques.

9 ACKNOWLEDGMENTS

Thanks to Qin Zhang for speeding up the wavelet transform, Sumit Chawla for helping with wavelet kernels, and Pat Hanrahan for letting us bounce packets off his computer. Thanks also for the reviewers for making us do better work.

References

[1] A. Albanese, J. Bloemer, J. Edmonds, M. Luby, and M. Sudan, "Priority encoding transmission", *Proc. 35th Annual Symposium on Foundations of Computer Sciences*, Santa Fe, NM, pp. 604-612, 1994.

[2] T.C. Bell, J.G. Cleary, and I.H. Witten, "Text Compression," Prentice Hall, Englewood Cliffs, NJ, 1990

[3] E.W. Biersack, "Performance evalutaion of forward error correction in ATM networks," *Proceedings of the SIGCOMM 92 Symposium*, Baltimore, 248-257, 1992 .

[4] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. "TCP Vegas: New techniques for congestion detection and avoidance." *Proceedings of the SIGCOMM '94 Symposium*, Aug. 1994.

[5] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, John Wiley & Sons, Inc., New York, 1991.

[6] M. W. Garrett and M. Vetterli, "Joint source/channel coding of statisticallyl multiplexed real-time services on packet networks," *IEEE Transactions on Networking*, 1:1, 71-80, Feb 1993.

[7] Georgia Tech Graphics, Visualization, & Usability Center, "Third degree polynomial curve fitting for bytes transferred per month by service," NSFNET Backbone Statistics Page, August 1995, http://www.cc.gatech.edu/gvu/stats/NSF/-merit.html.

[8] G. Karlsson, and M. Vetterli, "Subband coding of video for packet networks," *Optical Engineering*, 27(7), 574-586 1988

[9] C. Leicher, "Hierarchical encoding of MPEG sequences using priority encoding transmission (PET)," TR-94-058, ICSI, Berkeley, CA, Nov. 1994.

[10] A. S. Lewis and G. Knowles, "Image compression using the 2-D wavelet transform," *IEEE Transactions on Image Processing*, Vol. 1, No. 2, pp. 244-250, April 1992.

[11] E.J. Posnak, S.P. Gallindo, A.P. Stephens, and H.M. Vin, "Techniques for resilient transmission of JPEG video streams," preprint.

[12] J. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Transactions on Signal Processing*, Vol. 41, No. 12, pp. 3445-3462.

[13] Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers," *IEEE Trans. Acoustics, Speech, and Sig. Proc.*, 36:9, 1445-1453, 1988.

[14] N. Tanabe and N. Farvardin, "Subband image coding using entropy-coded quantization over noisy channels," *IEEE Journal on Selected Areas in Communications*, 10:5, 926-943, 1992.

[15] A. Tanenbaum, "Computer Networks," Prentice-Hall, Englewood Cliffs, N. J. 1981.

[16] Turner, Charles J., and Larry L. Peterson, "Image transfer: and end-to-end design," *SigComm 92*, 258-268.

[17] J.D. Villasenor, B. Belzer, J. Liao, "Wavelet filter evalution for image compression," *IEEE Trans. Image Processing*, Aug. 1995.

[18] C.L. Williamson and D. R. Cheriton, "Loss-load curves: Support for rate-based congestion control in high-speed datagram networks," *Proceedings of SIGCOMM 91*, pp. 17-28, 1991.

[19] I. Witten, R. Neal, and J. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, 30:6, 520-540, 1987.