Image Transfer: An End-to-End Design

Charles J. Turner and Larry L. Peterson[1]

TR 92-1

Abstract

The transfer of digital images between data archives and scientific workstations is likely to consume a significant amount of network bandwidth in the very near future. This paper examines the image transfer problem from an end-to-end perspective, that is, it describes a complete image transfer protocol that takes into account both the nature of digital imagery and the properties of the underlying network. Specifically, it describes a simple algorithm for encoding images into network packets in such a way that the receiver can recover from dropped packets without requiring the sender to retransmit them. Avoiding retransmissions has the advantages of improving response time and eliminating the need to buffer data at the sender.

July 2, 1992

Department of Computer Science
The University of Arizona
Tucson, AZ 85721

---

[1] This work supported in part by National Science Foundation Grant IRI-15407 and DARPA Contract DABT63-91-C-0030.

1 Introduction

In the near future, members of the scientific community will require the ability to transfer extremely large amounts of digital imagery—still images typically ranging in size from 1MByte to 25MBytes. Earth observing platforms will be collecting data continuously at a rate of several hundred gigabytes per day [7], with researchers around the world performing interdisciplinary, multi-sensor experiments using data from distributed archives. At this collection rate, these users could become one of the largest consumers of wide-area network bandwidth. In order to use this volume of data effectively, it is necessary to give researchers the fastest possible transfer methods.

This paper addresses the problem of effectively transferring digital images across high-speed, wide-area networks. In contrast to the way images are currently transferred using general purpose bulk transfer protocols [11, 3, 4], our approach is based on an end-to-end design that considers all aspects of the problem, including user requirements, characteristics of the data, data compression, and network performance. In other words, rather than use protocols designed to reliably transport arbitrary data, our design is tailored to the transport of digital images. Although the techniques described in this paper are general enough to apply to all types of images, for concreteness we focus on satellite imagery. Consider the following two properties of satellite imagery, and how our image transfer mechanism takes advantage of them.

First, almost every aspect of remote sensing involves high degrees of uncertainty. Atmospheric turbulence, variations in solar illumination, and viewing geometry all influence the image formation process. Furthermore, errors are introduced by sensor optics and detectors, as well as during down-link transmission. At typical image resolutions, there is also a high degree of surface brightness variation due to differences in material reflectance, topography, lighting, climate, and so on. The net result is that the remote sensing community has designed applications that are able to deal with systematic uncertainty in the input data. For example, browsing large databases for candidate tests images, quicklook viewing prior to sensor activation, and large scale thematic mapping are applications that are particularly tolerant of imperfect data. Because the data already contains substantial errors and the applications can tolerate these errors, our image transfer protocol is able to relax the reliability guarantees of conventional bulk transfer protocols.

The second significant property of satellite images is spatial redundancy, or highly correlated brightnesses within local neighborhoods. When observed from space, surface brightnesses tends to change gradually in most scenes. Additionally, many of the same processes that contribute to image uncertainty do so in such a way as to smooth out irregularities at the pixel and local neighborhood levels—individual pixels actually represent an average brightness of the underlying substances. Because the images contain redundant information, our image transfer protocol is able to recover from errors (i.e., dropped packets) introduced by the network. This is very much like forward error correction [2], except we take advantage of the spatial redundancy already present in the images instead of explicitly adding redundant information to the data prior to transmission.

Our approach is to not retransmit lost packets, but instead to use the spatial redundancy in the data to recover the missing information. Although intuitively simple, our design is complicated by three factors. First, it is necessary to minimize the impact of each lost packet on image quality. This is accomplished by encoding images in such a way as to insure that the pixels contained in a single packet are scattered throughout the image. Then, when a packet is lost, redundancy in the remaining image data can be used to reconstruct lost information. Second, in order to support a wide range of network bandwidths, it is sometimes necessary to compress the data. Therefore, our encoding algorithm also preserves those qualities of the image that support data compression. Third, we have designed our protocol in such a way that all packets can be processed independently and in any order [5]. This has benefits with respect to parallel protocol design and overall system simplification.

The key characteristic of our design is that by properly encoding images in network packets, we are able to avoid the retransmission of lost packets. Avoiding retransmission has two important advantages.

- Retransmission-free protocols can improve end-to-end response time. Although raw network bandwidth is improving, the effective throughput rate of a retransmit-based protocol will not improve proportionally if the network suffers from either: (1) high latency, or (2) significant packet loss. For example, a 1Gbps network is able to transport a 10Mbyte image in 100ms. If even one packet needs to be retransmitted, then the response time doubles on a wide-area network with 100ms round-trip latency. Should a significant number of packets be dropped, then multiple round-trip times may be necessary to successfully transmit the data, further degrading performance.

- Retransmission-free protocols drastically reduce the buffer space required on the sending machine. Protocols that retransmit lost packets must buffer all transmitted data until it has been acknowledged. For example, on a 1Gbps network with 100ms round trip latency, a reliable protocol like TCP would either need to maintain 12.5 Mbytes of buffer space for each image it is transmitting, or reduce its window size, and consequently, its effective throughput rate.

Note that although this paper focuses on the transmission of digital satellite imagery, we also have some experience with other types of digital images, including digitized text and bit-maps. In particular, we plan to use the protocol to transfer journal page images as part of a nationwide “collaboratory” for a community of Molecular Biologists [13]. We also believe the techniques presented are applicable to the transfer of real-time video, speech, music, and instrument data. Finally, even though we concentrate on the transfer of images between a data archive and an end user’s workstation, the same techniques could be applied to the transfer of images form remote sensing devices (e.g., satellites) to the archives.[1]

2 Algorithms

This section outlines a suite of algorithms for efficiently encoding digital images into network packets so as to maximize information recovery when packets are dropped. Intuitively, our approach is to eliminate the temporal to spatial relationship among packets. Furthermore, because our image transfer protocol is designed to work on networks with a variety of bandwidths, our algorithms are constrained by the need to preserve those properties of images required to support data compression.

Information that is lost during transmission must be recovered through some other means. This is accomplished by encoding the source images in such a way as to insure that information that is local in “image space” is not local in “packet space”. Specifically, we have devised an encoding scheme that guarantees that no two adjacent image pixels will ever be in the same network packet, or more strongly, in two packets within the same group of N transmitted packets. When a packet is lost, the corresponding lost bytes will be scattered over the full image. This provides the optimal conditions under which to run various reconstruction algorithms around the missing pixels.

---

[1] Network latency can be a significant issue in such cases. For example, the Venus-to-Earth latency is over 4 minutes. The protocol currently used to transfer images from Magellan to Earth does not retransmit lost data, but at the expense of worse image quality than could be achieved by the encoding algorithm described in this paper.

Because response time is our primary concern, and all recovery operations must be performed on-line, it is important to limit their execution time. Additionally, it is desirable to be able to process incoming data incrementally and in parallel. This is accomplished by decomposing the source image into independent packets, each of which contains enough information to determine its correct location in the destination image. Even though packets are transmitted sequentially, there is no temporal relationship between adjacent packets. The loss of one or more consecutive packets does not prevent the protocol from processing subsequent packets. Loss bursts due to network congestion become less important because it is *average* packet loss rate that ultimately determines image quality. This temporal robustness allows us to adjust the transmission rate with less sensitivity to short term congestion conditions.

Before presenting the encoding algorithm itself, we first consider the issues involved in data compression and image recovery. Note that the purpose of this section is to identify the key parameters of the system. The next section puts the pieces together by describing the complete image transfer protocol.

2.1 Compression Algorithms

The question of whether image compression is appropriate for a particular application is one that can only be answered by considering network bandwidth, quantity of data to be transmitted, and available computer processing power of the communicating hosts. Consequently, it is important to look at the interaction of a variety of image compression techniques with our encoding algorithm. The concepts employed in our image encoding algorithms rely on many of the same properties of images that are employed in compression techniques. Also, a system design that handles all packets independently may affect the performance of certain compression techniques. If compression is required for a particular network configuration, then it is important to insure that the encoding scheme being used does not destroy the potential to achieve significant compression results.

The key to obtaining significant compression in digital imagery is to exploit local redundancy. In the image processing community, redundancy is expressed using the autocorrelation function, which measures the similarity between samples as a function of separation (lag). Autocorrelation can be computed in both the horizontal and vertical directions for stationary imagery, and in the inter-frame dimension for video data. Typically, the function has the same shape in all three directions: it drops off exponentially from a lag of zero and peaks again when the lag is the length of the full line (RowLength), which corresponds to vertical correlation. Figure 1 plots the horizontal autocorrelation function for a typical satellite image. In other words, compression algorithms work best when run on consecutive bytes within the same row or column, with decreasing performance as the distance between pixels increases. Due to the radially symmetric nature of image autocorrelation, they still behave reasonably well along diagonal tracks across the image.

There are two basic classes of compression algorithms: loss-less and lossy. Loss-less compression algorithms are capable of recovering all original information during the decompression phase. The cost of this complete data recovery capability is a limitation in the maximum achievable compression ratio—the best that can be achieved by loss-less algorithms on image data is typically 2:1. Lossy algorithms are employed when higher compression ratios are required. Using lossy algorithms, it is not difficult to obtain compression ratios of 5:1, with much higher rates under more ideal circumstances. The cost of achieving this higher compression ratio is increased computational requirements. The most commonly used lossy compression

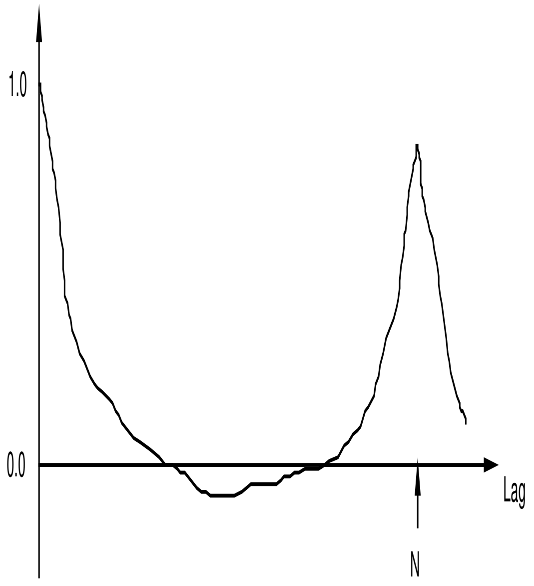


Figure 1: Autocorrelation Function for a Typical Image.

algorithms are transform domain algorithms that perform some transformation on the data, compressing this transformed data. An advantage of this approach is that information removed in the transform domain will be evenly distributed over the reconstructed image, making the lost information less noticeable.

We have considered the following loss-less and lossy algorithms in conjunction with our encoding algorithm:

- Run length encoding (RLE) [12] is a loss-less algorithm that compares adjacent pixels in gray level, encoding only changes. Each packet is encoded as a sequence of ordered pairs of bytes, each pair containing a gray level and a run length. For images that have large homogeneous regions, (e.g. binary data, maps, text) this technique is quite effective. However, for images with even a small degree of local variation, it is not uncommon for image sizes to increase.
- Differential Pulse Code Modulation (DPCM) [9] is also a loss-less algorithm. It is better suited for digital satellite data in that it takes advantage of the local correlation between adjacent pixels. Due to this correlation, the dynamic range of the differences between the adjacent pixel gray levels can be significantly less than the dynamic range of the original image, and can therefore be represented using fewer bits. The technique begins by outputting a reference pixel value and then a sequence of differences in gray level from the previous pixel value. If the difference is too great then the sequence is reset and a new reference value is output.
- Discrete Cosine Transform (DCT) [9, 16] is the most commonly used lossy compression algorithm. It transforms the image into its constituent spatial frequencies. The number of bits allocated to each frequency is determined by the degree to which that frequency contributes to the overall image. For example, in a very smooth image with little detail, there is little information contained in the higher spatial frequencies, whereas the opposite is true of images with lots of detail and sharp edges. Using

the fastest computational techniques, it is about an order of magnitude slower to compute than the two loss-less techniques.

Since our image encoding algorithm treats each packet independently and is tolerant of random packet loss and out of order packet delivery, there can be no state dependent compression information that involves multiple packets. Consequently, all compression operations are performed within a single packet, and reset at packet boundaries. In order to achieve optimum compression within a single packet, it is best to assign bytes to each packet in a manner that most closely matches the image's autocorrelation function. Again, this favors sending packets in strict row or column order. Unfortunately, this is exactly the encoding that we are attempting to avoid. By insuring that no two spatially adjacent bytes are in the same packet, we preserve the spatial redundancy for later use during recovery. As a compromise, we must look for encoding algorithms that are as close to the optimal as possible from a compression standpoint, while still preserving the desired scattering properties.

2.2 Reconstruction Algorithms

The key to error recovery techniques is the local redundancy found in most types of imagery. Any given pixel of data in an image is highly correlated with its surrounding neighbors. This is because surface brightness tends to change gradually in most natural scenes, and many of the processes involved in image formation have a smoothing effect. Additionally, images like maps and documents contain large uniformly colored regions. These local similarities can be used to reconstruct or approximate the value of each missing pixel.

The reconstructed value for the lost pixel can be obtained in several ways from the surrounding valid values. Because this operation must be performed during image transmission, it is desirable to make it as fast as possible. Like many other parts of an end-to-end system, the right recovery method is a function of required image quality and execution speed requirements. The following list identifies a range of recovery algorithms ordered by increasing accuracy and processing requirements [1, 14, 12]. To date, we have implemented and experimented with the first six approaches.

No-Recovery: Rely on the distributed nature of the lost pixels to minimize the overall impact on subjective image quality.

Row-Based: Use neighboring rows to compute new values for a pixels in a lost row. Appropriate only when all the pixels in an entire row are lost. Works well for a small number of lost rows.

Mean Image Gray Level: Substitute the mean image gray level for all lost pixels. This is the fastest method, requiring only that the average Gray Level for the entire image be precomputed and transmitted to the receiver before recovery can begin. This method performs best in low contrast images.

Mean Packet Gray Level: Substitute the mean gray level value for all bytes in a lost packet. This requires the computation and transmission of the average gray level for each packet sent, a data increase of 1%. This method may behave better than using the mean image gray level, depending on the local structure of the image.

Linear Interpolation: Use between one and eight of each lost pixel's neighbors to compute a new value for each missing pixel. This method relies only on available information, but is more computationally expensive than simple average value substitutions.

Median Filtering: Use the median pixel value of the immediate neighbors as the new value for the lost pixel. This method is more sensitive to lost pixels along edges within the image. Instead of taking a simple average of two distinctly different regions, the lost pixel is assigned the value of the region with greatest representation within the window.

Cubic Interpolation: Use a larger neighborhood around each lost pixel to perform cubic interpolation. While much more costly than linear interpolation, this method is known to produce higher quality results as it more closely mirrors the behavior of the original image forming optics.

Morphological Filters: Use morphological (shape-based) filters in an attempt to associate each lost pixel with more complex structures within the image. This can be particularly beneficial when dealing with text or images with well defined structure.

2.3 Encoding Algorithms

The success of most recovery algorithms is determined by the amount of information available for recovery around each lost pixel. The encoding algorithm used prior to image transmission determines the manner in which lost data is distributed throughout the receiver's image. If pixels associated with a given packet can be scattered uniformly over the image in both horizontal and vertical dimensions, then for any single lost packet, no two pixels will be adjacent in the reconstructed image. Ideally, a fully random scattering of pixels across the image should be employed. Unfortunately, random scattering is undesirable from the point of view of image compression. Instead, we devise a simple compromise.

Consider the consequences of packet loss in a standard file transfer. Typically, images are laid out on disk in raster order. When the image is packetized for network transport, bytes are extracted from the file sequentially. This leads to packet formation in which complete rows of image data occupy a single packet. Thus, the loss of one packet results in the loss of a complete row of image data. While correction algorithms for this type of loss exist—using adjacent rows to perform interpolation—in general they have poorer performance than algorithms which have access to a complete surrounding neighborhood. When a full row is lost, each pixel's surrounding information is reduced by 25% of that available to an isolated pixel. More importantly, the loss of multiple adjacent packets—e.g., due to a burst error or network congestion—causes row-based interpolation performance to degrade rapidly.

While there are many options for packet construction that satisfy the goal of distributing local redundancy, the number of options is severely constrained by the need to maintain compression potential within packets. Our approach is to find the encoding closest to the compression optimal with the desired packet non-locality encoding behavior. It is also important to consider the inverse mapping so that the receiver can decode packets efficiently. Distributions of pixels that are regular—i.e., with constant offsets—are more efficient to decode than random mappings or more elaborate functions of pixel number.

Given an image of size `RowLength x ColLength` pixels, our encoding algorithm is expressed using two key parameters: `ByteOffset` and `PacketOffset`. `ByteOffset` controls how bytes within a single packet are distributed throughout the image; it specifies the number of bytes between adjacent pixels in the same packet. It is this parameter that insures that no two bytes within the same packet are ever adjacent in image space. `PacketOffset` addresses the problem of bursty packet loss; it specifies the number of bytes between

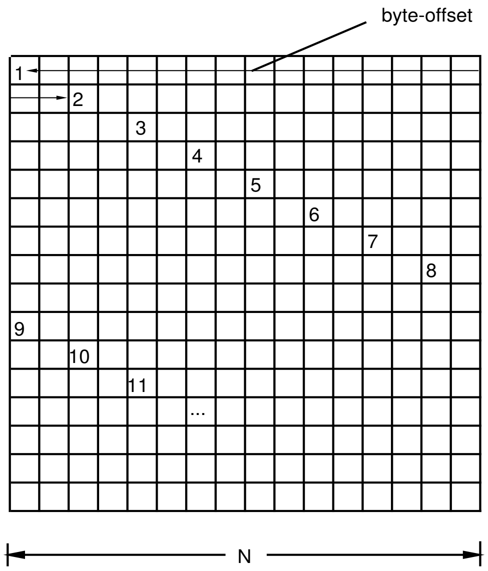


Figure 2: Assigning Pixels to Packets Using ByteOffset = RowLength + 2

starting pixels of adjacent packets. This parameter controls how far apart temporally adjacent packets are in image space.[2]

To illustrate the effects of changes in the ByteOffset parameter, consider an encoding in which ByteOffset = RowLength. In this case, one byte is selected from each image row for inclusion in a packet, perfectly aligned in columns below the first byte of the image. As another example, setting ByteOffset = RowLength + 1 results in diagonal pattern moving from the top left to bottom right, shifting to the right one pixel each row. Note that while these two encodings are good from a compression standpoint, neither exhibits the desired properties of non-locality in that the loss of any one packet will result in more than one lost pixel within each 3×3 window. Figure 3 shows a packet constructed using ByteOffset = RowLength + 2. In this case, successive pixels are selected from one row down and two columns to the right. This encoding has the desired property that no two adjacent pixels from the image are in the same packet. If the packet shown in Figure 3 were to be dropped, a 3×3 window could be drawn around each missing pixel without having more than one hole in each window.

The second parameter used in our encoding algorithm, PacketOffset, is intended to compensate for bursty packet loss. When congestion occurs in the network, it is often the case that multiple adjacent packets will be lost [8]. In experiments on the TCP/IP Internet we observed an average loss burst of 3.3 packets, with 12% of the losses having a burst size of greater than four packets.[3] Figure **??**, shows the effect of adding PacketOffset bytes between the start of each subsequent packet. With a PacketOffset of one, adjacent

---

[2] To simplify the discussion, we equate pixels with bytes. In general, pixels can range from a byte to one or more words.

[3] We measured these loss bursts while transmitting 558 independent 1-MByte messages between two pairs of Internet sites over a three month period. The messages were transmitted as 1024 1-kbyte packets. The packets were transmitted without flow control, but at a slow enough rate so that we experienced an overall packet drop rate of only 2-4%.

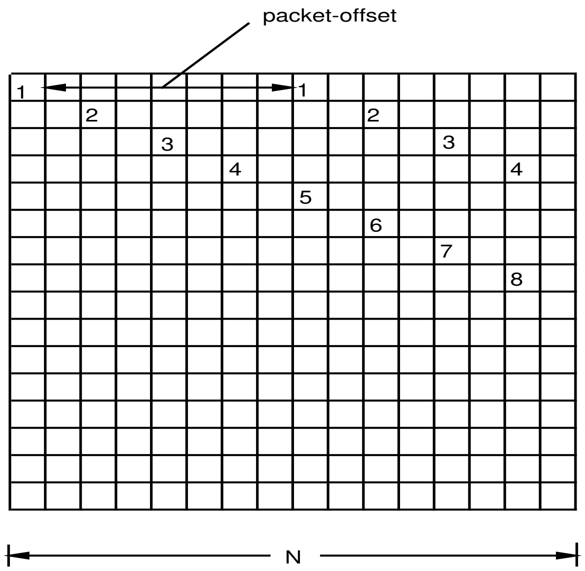


Figure 3: Consecutive Packets Selected from Image Using PacketOffset

packets occupy adjacent diagonals across the image, and result in spatially adjacent pixels being lost. For this reason, a PacketOffset that is greater than the average burst loss size is added to all packet encodings. This results in temporally adjacent packets being selected from the source image in increments large enough to protect against groups of lost packets.

3 Protocol

This section sketches the design of the complete image transfer protocol based on the encoding, compression, and reconstruction algorithms discussed in Section 2. It also discusses the relationship between the image transfer protocol and the service guarantees offered by the underlying network.

3.1 Defining Encode-Decode Tables

An initial analysis phase defines a static set of encode-decode tables based on different settings of the following parameters:

- SubimageSize = RowLength x ColLength
- PacketLength
- ByteOffset
- PacketOffset

SubimageSize is the size into which all larger images will be divided for network transfer and higher-level application presentation. It is a rectangular shaped block intended to be a sufficiently large amount of imagery, yet manageable by most receivers. A convenient size is 1 Mbyte, or 1024 × 1024 pixels. Thus, an image of size 5K x 5K would be divided into 25 subimages. Once the subimage size and shape have been decided, the ByteOffset parameter is set to the RowLength + 2. Finally, the other two parameters are set according to the characteristics of the underlying network: PacketLength is set to the expected network MTU, and PacketOffset is set to the expected burst loss size.

Each encode-decode table maps packet sequence numbers to starting offsets within the subimage block. The same set of tables are available at both the sending and the receiving hosts. The sender uses the encode-decode table to format packets for network transfer. The receiver uses the table to find correct placement within the reconstruction buffer of each packet as it arrives.

3.2 Image Pre-processing

An off-line phase takes in a raw image from some sensor system, encodes it for later transmission using the appropriate encode-decode table, optionally compresses the encoded packets, and stores the resulting packets in a persistent archive. Note that a single image can be encoded multiple times using different encode-decode tables, thereby preparing it for transmission under different circumstances.

The main idea in performing the off-line encoding is to place the image bytes in the exact order in which they will later be sent over the network when a transfer is requested. This eliminates the need to select bytes from noncontiguous locations within the image, as well as the need to compress the packet at transmission time. Instead, by pre-encoding the data in packet format, it is possible to copy data directly from the disk to the network interface in the exact format of a transmitted packet. Because the image is preformatted, the on-line operation performed by the sender is exactly the same as that performed by any sequential file-transfer protocol.

3.3 Transfer Time

When a particular image is requested from the archive, the sender simply reads the encoded/compressed packets from disk and transmits them with their associated sequence numbers. The sender also indicates which decode table the receiver should use.[4] The receiver pre-allocates a reconstruction buffer equal to the size and shape of the subimage block. This is the granularity at which image data is supplied to the application. When a packet arrives, it is first decompressed, if necessary. Next, the packet sequence number is used to index into the encode-decode table to obtain the starting offset within the subimage reconstruction buffer. The first byte of the packet is placed at this offset, with each subsequent byte placed at ByteOffset bytes away. The placement of all bytes in an image is still $O(\mathsf{SubimageSize})$. The difference between this placement algorithm and a standard sequentially-oriented placement operation like **bcopy**, is that when pixels are 8-bits, they must be stored one byte at a time rather than by word, resulting in a performance penalty. We measured this penalty to be 2.83 on a Sparc 2.

The most costly operation required on the receiver's end prior to delivering the completed reconstruction buffer to the application is the process of recovering missing data. For the better reconstruction algorithms, some number of bytes from a 3 x 3 window around each lost pixel must be accessed in order to perform a

---

[4] We assume a small, static set of encode/decode tables, making it realistic for the receiver to have a copy of each table.

linear interpolation of the missing value. This operation is $O(M)$ where M is the number of bytes lost from the full subimage. Although one can imagine performing recovery incrementally—that is, at the same time as additional packets are arriving—there is the added complication of deciding when a particular packet is missing as opposed to delayed [6]. On a uniprocessor, our approach is to not perform any recovery until the complete subimage is received, but to overlap the reconstruction of one subimage with the receipt the packets in the next subimage. Since all packets and subimages are independent, additional parallelism is possible on a multiprocessor.

3.4 Relationship to the Underlying Network

As just described, the image transfer protocol remains detached from any particular network architecture. We now complete the picture by discussing the relationship between the image transfer protocol and the service guarantees offered by the network.

In networks like the TCP/IP Internet, the network protocol (IP) provides only best-effort delivery. Bandwidth is not reserved, and should a gateway become congested, it discards packets without regard to the requirements of the competing users. In this case, there is no resource allocation to speak of, and the transport protocol (TCP) is responsible for coping with highly variable delays and avoiding congestion [10]. A significant amount of recent research has focused on various resource allocation schemes. The idea is to implement a smarter queuing discipline in the gateways, and to allow end applications to request (and be guaranteed) a particular quality of service from the network. The various service guarantees made by these schemes include throughput rate, delay, jitter, and loss rate. Consider both of these scenarios—current dumb gateways and proposed smart gateways—starting with the latter.

Recent efforts to design new resource allocation schemes have concentrated on the needs of real-time video. While this makes sense—real-time video makes resource allocation a hard problem—it is important to understand the relationship between these schemes and image transfer. First, real-time video is concerned with network jitter, but jitter guarantees are not particularly meaningful to an image transfer protocol. Resource allocation schemes that use non-work conserving techniques to minimize jitter are overly conservative with regard to image transfer. Second, whereas real-time video applications are able to request from the network (and need to have guaranteed by the network) a certain transmission rate, the amount of bandwidth requested by an image transfer protocol is best stated as “as much as is available”. The more interesting property to the image transfer protocol is the loss rate it will realize if it transmits at a particular rate. Thus, load-loss curves proposed by Williamson and Cheriton [17] offer the ideal resource allocation support to an image transfer protocol.

On a best-effort delivery network like the current TCP/IP Internet, the image transfer protocol sends IP datagrams and computes the load-loss curve itself. That is, the receiver measures the loss rate it is experiencing, and uses feedback to instruct the sender to increase or decrease its transmission rate accordingly. Obviously, such an algorithm does not interact well with the current TCP congestion control algorithm: our algorithm not only tolerates dropped packets but actually increases the transmission rate until a maximum acceptable loss rate is achieved, whereas such drops cause TCP’s algorithm to give up bandwidth. While one could argue that such an aggressive algorithm should not be run at the expense of the TCP algorithm, the fact that these two algorithms cannot co-exist on the same IP network points to the need for more robust resource allocation strategies in the IP gateways.

4 Experimental Results

This section presents the results of using different compression, image encoding, and recovery algorithms. We have tested our system on samples from four different types of of digital images:

TEXT: Digitized text.

MSS: Landsat Multispectral Scanner.

TM: Landsat Thematic Mapper.

SPOT: French SPOT Satellite.

The last three image types are satellite images of different resolutions and spectral properties. We used five sample images of each type; the numbers reported for each type are the average of the measurements for the five samples of that type. All experiments were performed at the subimage level, with images ranging in size from 512 to 1024 squared.

For the purpose of this discussion, we refer to the encoding of an image into packets with `ByteOffset` equal to 1 or `RowLength` as a *raster-based* encoding, and we refer to our proposed encoding using `ByteOffset = RowLength + 2` as a *scattered* encoding.

4.1 Compression

In order to determine the extent to which encoded packets are compressible, we experimented with the three compression algorithms described in Section 2: RLE, DPCM, and one-dimensional DCT. To establish a baseline, we ran the three compression algorithms on the raw input images. This establishes a best-possible value that measures how well the algorithm works on a contiguous input image with the highest possible autocorrelation. The results of this test are shown in Table 1. Table values are compression ratios, which measure the average number of input bytes that can be packed into a single output byte. Ratios are not given in certain cases because the compression algorithm does not work well on that particular image type; i.e., the compressed image is actually larger than the original image. In the case of DCT, because it is lossy, it is possible to get any desired compression ratio, but at the expense of image quality. The ratios reported for DCT result in a reasonable error, using both global and subjective quality measures [15].

	RLE	DPCM	DCT
TEXT	8.13	—	—
MSS	—	1.60	6.56
TM	—	1.59	4.34
SPOT	—	1.53	6.10

Table 1: Compression Ratio Before Encoding

As can be seen, different compression techniques work best for different types of images. The RLE technique works best with high degrees of homogeneity or long stretches of like-valued pixels, such as those commonly found in digitized text. The DPCM method does much better with the slight local fluctuations found in most satellite imagery. DCT works well with low frequency data, but not with sharp edges as is commonly found in text.

Next, we tested the scattered encoding algorithm to determine what impact it has on compression algorithm performance. Table 2 shows the compression ratios for the same images after encoding, where now compression was performed on individual encoded packets. Table entries show overall compression ratio, as well as the percentage compression achieved relative to the raw image tests used as a baseline. For example, after encoding, we were able to achieve a compression ratio of 5.45 for MSS imagery using DCT, which is 83% of that obtainable when using the full unencoded image.

	RLE	DPCM	DCT
TEXT	5.04 (62%)	—	—
MSS	—	1.46 (91%)	5.45 (83%)
TM	—	1.36 (86%)	3.21 (74%)
SPOT	—	1.30 (85%)	4.00 (66%)

Table 2: Compression Ratio After Encoding

Note that we tested encodings with ByteOffset set to values other than RowLength + 2, but this setting gave the best compression performance while still achieving spatial non-locality within the packets. Also note that in the case of DCT, we had to use an alternative bit quantization map due to the redistribution of frequency data.

4.2 Recovery

We next measured the effectiveness of various image recovery algorithms. In this test, we assume a burst-oriented packet loss model, in which a variable percentage of packets are lost during the transfer. Packets are dropped in randomly located groups uniformly distributed with a gaussian distributed group size with a mean of 8 and variance of 1 packet. This test was performed for all images using both the raster-based and scattered encoding. In the case of the latter, we set PacketOffset = 8.

There are two basic types of image fidelity criteria commonly used to measure image quality: subjective and objective. Subjective measures are concerned with how an image looks to a human observer. A commonly used objective technique is the *root mean squared error* (RMS error), which measures the average absolute error between the original image and the corrected image. An RMS error of 0 indicates perfect recovery. Table 3 shows the average RMS error for five SPOT images with packet loss rates of 1, 5, 10, and 20 percent; a more thorough analysis of the impact the encoding has on image quality can be found in a companion paper [15]. The row labeled “No-Recovery” shows the RMS error for the unrecovered image and the row labeled “Row-Based Recovery” corresponds to the recovery of a raster-based encoded image. The other four rows correspond to different recovery algorithms run on an image encoded using the scattered encoding.

As can be seen, there is an inverse relationship between error rate and recovery algorithm sophistication. No-Recovery has the highest RMS error, increasing with packet loss. The eight neighbor linear interpolation is one of the more expensive recovery algorithms and achieves the best results for SPOT imagery.

	1 Percent	5 Percent	10 Percent	20 Percent
No-Recovery	6.9	15.5	21.8	30.8
Row-Based	1.3	3.3	4.8	9.6
Mean Image Gray Level	1.7	3.7	5.3	7.5
Mean Packet Gray Level	1.7	3.7	5.3	7.5
4 Neighbor Interpolation	0.9	2.0	2.9	4.1
8 Neighbor Interpolation	0.7	1.6	2.4	3.6
Median Filter	0.8	1.9	2.9	4.2

Table 3: RMS Error for Recovery Algorithms on SPOT Imagery

4.3 Computation Time

We now consider the execution times associated with the various decoding and recovery algorithms. These times were collected by running the algorithms on a Sparc 2 using the same images and packet loss models described above. The most significant cost of our approach is the error recover stage. Table 4 indicates the recovery rates for each of the five recovery algorithms in “megabytes-per-second”. Recovery rates refer to the time required to correct *actual lost data*. For example, at a packet loss rate of 1%, 4 neighbor interpolation can process an image at a rate of 27.2 MBytes/sec (or 36ms for a 1MByte image), and at a 10% loss rate, it can process the image at a rate of 2.7 MBytes/sec (or 360ms for a 1MByte image).

	Recovery Rate
No-Recovery	∞
Mean Image Gray Level	0.435
Mean Packet Gray Level	0.435
4 Neighbor Interpolation	0.272
8 Neighbor Interpolation	0.196
Median Filter	0.027

Table 4: Recovery Algorithm Execution Rate (MBytes/sec)

As can be clearly seen, there is a direct relationship between execution time and recovery performance. No-Recovery and average gray level substitution require no or little time, whereas the best recovery methods impose a higher cost. The dependence of recovery on percentage packet loss is linear in the number of lost bytes.

5 End-to-End Performance

To get a handle on the performance impact of using recovery instead of retransmission, this section establishes two simple models of transfer time: one for retransmission-based protocols and one for retransmission-free protocols. Both models are examined under a variety of network and processor conditions and attempt to highlight the impacts of bandwidth, latency and packet drop rate. In order to do this in a meaningful way, the most conservative assumptions of protocol behavior have been assumed at all stages. We have also been able to confirm our results experimentally on low-bandwidth networks by comparing our image transfer protocol to FTP.

The most significant cost associated with our image transfer protocol is the interpolation required to perform lost pixel recovery. In the case of eight neighbor interpolation, for example, recovery requires eight indirect references, eight adds, and one division for each lost byte. Because the total cost of recovery is a function of the percentage of dropped packets, we have modeled both transmission-based and retransmission-free protocols under two different network packet loss situations: 1 and 10 percent packet loss. One additional variable that can be expected to change drastically over the next few years is processor performance. Therefore, in addition to using the performance times of a Sparc 2 reported in Section 4, we also hypothesize the existence of a workstation with 10 times the performance of a Sparc 2. The impact of this change in execution speed has been restricted to recovery rate.

In the equations that follow, constant latencies and bandwidths are assumed for both protocols over the duration of a transmission. This is unlikely to be accurate, as most protocols will alter their sending speeds based on packet loss. Additionally, the effects of network queuing have not been considered. Instead, we assume that all data is sent over the link at the average bandwidth. During that transfer, a certain percentage of the packets are dropped. At a minimum, a retransmission-based protocol must detect and make one request for retransmission of all the lost packets. As a further generous simplification, we assume that all lost packets can be successfully transmitted in one additional transfer that occurs immediately following the initial transfer. With these assumptions, the total elapsed time for a retransmission-based protocol, denoted T_r, is:

$$T_r = \frac{N}{B} + \frac{N * ¡}{B} + L \tag{1}$$

where N is the number of bytes in the transfer, B is the network bandwidth, $¡$ is the packet drop rate, and L is network latency. The first term, $N{=}B$, is the initial transfer of N bytes over a channel with bandwidth B. The percentage of lost packets is measured by $N * ¡$. This becomes the new transfer size in the second term of (1). The final term L is the one-time cost of requesting that all lost packets be retransmitted.

In contrast, the total elapsed time for the transfer of N bytes using our encoding and recovery algorithms instead of packet retransmission, denoted T_e, can be expressed as follows:

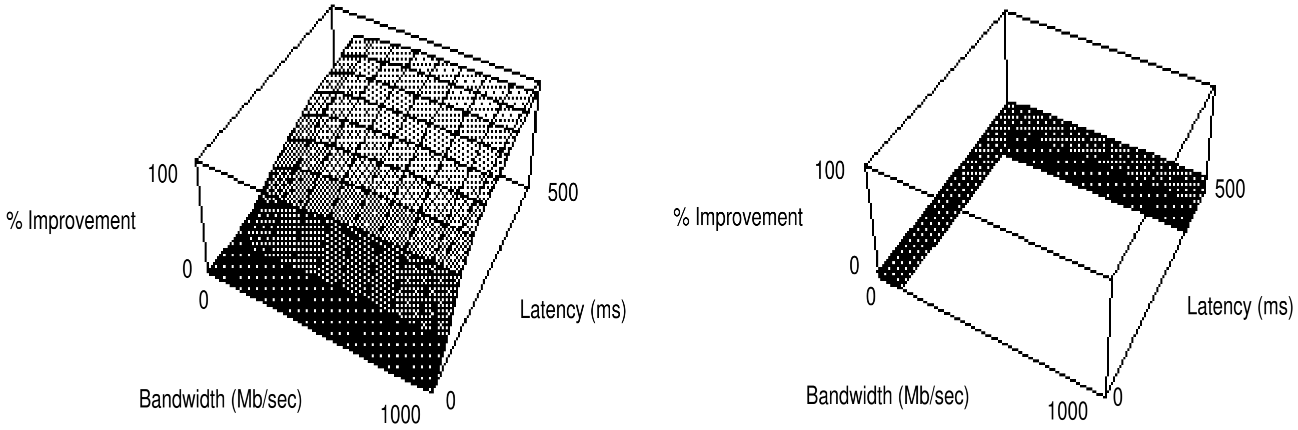
$$T_e = \frac{N}{B} + \frac{N * ¡}{æ} \tag{2}$$

where $æ$ is the rate of lost byte recovery (e.g., MBytes/sec).

Finally, to quantify the conditions under which lost packet recovery becomes more economical than retransmission, the difference in transmission times can be calculated as $T_d = T_r - T_e$, so that

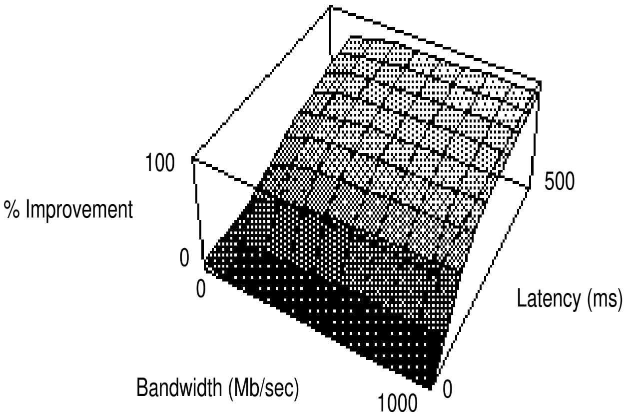
$$T_d = N * ¡(\frac{1}{B} - \frac{1}{æ}) + L \tag{3}$$

Figure 4 gives three graphs that plot percent improvement of the retransmission-free protocol model over the transmission-based approach as a function of network bandwidth and latency. Those conditions that produce positive time differences are shown as shaded areas in the plots, where a 50% improvement implies that the retransmission-free protocol is two times faster than the retransmission-based protocol. In an attempt to restrict the problem, a data transfer size of 1 Megabyte was assumed, latency (L) was varied



Sparc2 10% packet loss

Sparc2 1% packet loss



10 x Sparc2 10% packet loss

Figure 4: Performance Improvement under Various Conditions.

between 0 and 500 ms, bandwidth (B) was varied between 1 and 1000 Megabits per second. A four neighbor linear interpolation recovery algorithm was used to establish the value of *æ* as $0{:}27$ Megabytes per second on a Sparc 2.

The first graph shows the percent improvement for a packet drop rate (¡) of 1%. At such low packet drop rates, the effect of latency dominates in (3), so there is a significant transfer speed improvement associated with the retransmission-free protocol under most conditions. Experiments comparing FTP with our image transfer protocol over the Internet (1.5Mbs and 200-300ms latency) confirm this result: our protocol performed two to three times better than FTP.

The second graph shows the percent improvement for a 10% packet drop rate. In this case, a larger percentage of packets must be retransmitted and recovered. Under these conditions, bandwidth is more significant, quickly dominating the latency term. This leads to a situation where recovery is only profitable under a narrow range of low bandwidth or high latency conditions. Keep in mind that we are making the conservative assumption that the retransmission of lost packets can be done in one round trip time.

The last graph shows the percent improvement for 10% packet loss when the recovery algorithm is performed by a processor that is ten times the speed of a Sparc 2; the improvements are even more pronounced at lesser drop rates. In this case, the value of *æ* is 2.7 MBytes/second. Now, under most bandwidth and latency conditions, a time savings is demonstrated relative to the retransmission-based protocol. For example, a 78% improvement is realized on a 1Gbps network with 200ms latency; i.e., our protocol would be nearly five times faster than one based on retransmission. It should also be noted that speeds greater than this hypothetical *æ* are achievable today using parallel processors, or by the use of simpler recover algorithms.

There are two caveats to the analysis presented in this section. First, these calculations are based on a 1MByte subimage size. While the performance gains are less dramatic for larger image sizes, 1MByte is a natural *subimage* size, and we are able to overlap the receipt of one subimage with the recovery of another. Consider the transfer of a 10MByte image (ten 1MByte subimages) on a 1Gbps network with 1% packet loss and a 10 × Sparc 2 processor. It takes approximately 100ms to receive each subimage and 30ms to recover each subimage, meaning the recovery is for free. Thus, at a 100ms latency, our protocol still runs twice as fast as a retransmission-based protocol, even assuming that only one retransmission is required.

Second, our model does not consider the impact of the memory architecture—the cost of copying the data in each packet to the appropriate memory location once it arrives at the destination host. On modern RISC processors, memory-to-memory copy times range from 100-300Mbps, making memory bandwidth, not network bandwidth, the bottleneck on a gigabit network. Moreover, the encoding used in this study does byte stores, whereas the raster-based encoding can use **bcopy**, which does word stores. Thus, should memory bandwidth remain the bottleneck, this would become an important factor in end-to-end performance. On the other hand, we believe this will not be an issue for two reasons. First, our encoding algorithm can be adapted to work on word-sized blocks of pixels rather than single pixels. Second, pixel sizes are increasing in all new sensor systems, and as a consequence, they better match word sizes.

6 Discussion

This section discusses two topics that are indirectly related to our image encoding algorithms. Until now, we have focused on static digital images, providing algorithms for scattering packet data across the *x* and *y* dimensions. If this idea is abstracted to the point of simply insuring that adjacency in "packet space" is uncorrelated with adjacency in "information space" for a more general definition of "information space", then these techniques can be extended to other domains. We require only local redundancy within the information space, and a tolerance of small interpolation errors. In this section we describe two additional areas that might benefit from the application of our image encoding algorithms: real-time video transmission and progressive image transmission using resolution pyramids.

6.1 Real-time Video

Real-time video, because it cannot tolerate retransmission delays, does not retransmit lost packets. To accommodate lost packets, either the corresponding frame is not updated or some form of forward error correction is used. Also, real-time video protocols are able to achieve significant compression by taking advantage of inter-frame correlation. An alternative would be to use the redundancy within the video imagery to correct for lost packets. That is, in the same manner that the video compression community has exploited inter-frame correlation for compression, our encoding and recovery algorithms could be applied to successive frames. Instead of working in the two dimensional space of a single image, the encoding would be performed using adjacency within a volume, where the third dimension is time or successive video frames. The pixels assigned to any single packet would now be scattered over multiple video frames (time) and the standard *x* and *y* dimensions within each frame. This would increase the amount of information available for error recovery, leading to higher tolerances for dropped packets and better overall image quality after recovery.

We do not have experience applying our scattered encoding algorithm to real-time video. In the case of static images, our work indicates that the encoding algorithm does not interfere significantly with compression, but it is not clear how the use of our encoding algorithm would conflict with inter-frame compression. Our belief is that it would not.

6.2 Progressive Image Transmission

The problem of transmitting large amounts of imagery across slow networks has traditionally been addressed using data compression techniques. An alternative approach incorporates image resolution pyramids and progressive transmission. An image pyramid is a set of related images of decreasing resolution, beginning with a full resolution image at the lowest level of the pyramid. Each new layer is constructed using some form of down sampling operation on the higher resolution image below. The net effect is that the size of each successive layer is smaller than all previous layers. A typical approach is to down sample by a factor of two in each dimension, reducing the total image area by a factor of four at each layer.

Progressive image transmission is the process of incrementally transmitting a full resolution image using an image pyramid. The first image to be transmitted is the top layer (lowest resolution) in the pyramid. The next layer is transmitted while the receiver displays the first image. This allows the user to begin working with the image to the extent possible using lower resolutions, while the higher resolution data is being transmitted. Successive packets from the higher resolution layers are incrementally added to the original display, filling in additional detail over time. This process can continue until the entire pyramid has been transmitted, or the user has seen enough detail of that image and stops the process.

Our encoding scheme is orthogonal to, and complimentary with, progressive transmission: our encoding algorithm can be applied independently to each layer within the image pyramid as it is transmitted, resulting in faster transmission of that layer. The smoothing effect of down sampling performed during image pyramid construction will result in improved performance of the image recovery algorithms.

Alternatively, the image pyramid could be used to improve the performance of the recovery algorithm. The goal of progressive transmission is to get some of the information to the user as fast as possible, filling in the details over time. The cost of this progressive transmission is that a great deal of redundant information is actually sent. This redundant information could be used to improve the accuracy of the lost pixel recovery operations. If the down sampling algorithm used to produce each successive layer is known, then the process can be inverted when pixels are lost in higher resolution layers. Consider the case where each successive layer is created using a neighborhood averaging operation. For each neighborhood window in the next higher resolution layer, the original average of that window is already present at the receiver. In the event that only a single pixel within the neighborhood is lost, there is an exact solution available for the missing pixel. In the event of multiple lost pixels, the range of possible values is constrained. One could also use values from lower resolution layers to replace lost pixels without any computation.

7 Conclusions

This paper describes a method for encoding digital images in network packets in such a way that the data contained in lost packets can be recovered at the receiver without the need for retransmission. The more global lessons to be learned from this work concern three important tradeoffs in the design of image transfer protocols.

First, it is important for image transfer protocols to be able to trade packet loss for throughput. That is, the transfer protocol should be able to send packets at an aggressive rate with the expectation that some percentage of the packets will be dropped, or conversely, network architectures that do not allow the transport protocol to trade loss rates for bandwidth are overly restrictive. Stated another way: *if through some feedback scheme the sender detects that the network isn't dropping enough packets, it should be able transmit packets at a faster rate until an adequate loss rate is achieved.* Loss-load curves seem to provide the best resource allocation model for this sort of traffic.

Second, it has long been recognized that imagery, both static and video, contains a large amount of redundant information. Whereas, the conventional way to take advantage of this redundancy is through the use of compression algorithms, we take advantage of this redundancy by using it for reconstruction in lieu of retransmission. In a sense, we are letting the network (as it drops packets) compress for us. Regarding the apparent tradeoff between compression and recovery, our conclusion is that you can have your cake and eat it too: our experiments indicate that it is possible to spread neighboring pixels across different network packets (thereby improving recovery), yet be able to achieve significant compression ratios. This is an important factor in the success of our scattered encoding algorithm.

Third, the users of a digital image transfer protocol are often willing to trade image quality for improved response-time. In our scheme, they can select image quality as a function of the loss rate they are willing to tolerate and the expense of the recovery algorithm that they run. Our performance analysis indicates that users can achieve a two to ten time improvement in response time on both low- and high-bandwidth, high latency networks.

References

[1] R. H. T. Bates and M. J. McDonnell. *Image Restoration and Reconstruction*. Clarendon Press, Oxford England, 1986.

[2] E. Biersack. Performance evaluation of forward error correction in ATM networks. In *Proceedings of SIGCOMM 92*, Aug. 1992.

[3] D. Cheriton. VMTP: Versatile message transaction protocol. Request For Comments 1045, DARPA, Feb. 1988.

[4] D. Clark, M. Lambert, and L. Zhang. NETBLT: A high throughput transport protocol. In *Proceedings of the SIGCOMM '87 Symposium*, pages 353–359, Stowe, Vermont, Aug. 1987.

[5] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the SIGCOMM '90 Symposium*, pages 200–208, Sept. 1990.

[6] P. J. Downey. Distribution-free bounds on the expectation of the maximum with scheduling applications. *Operations Research Letters*, (9):189–201, 1990.

[7] A. Fleig. The EOS data and information system. In *Proceedings of Scientific Data Compression Workshop*, pages 73–83, NASA Office of Management. Scientific and Technical Information Division, Washington, D.C., 1989.

[8] S. Floyd and V. Jacobson. Traffic phase effects in packet-switched gateways. In *ACM SIGCOMM Computer Communications Review*, volume 21, Apr. 1991.

[9] R. C. Gonzales and P. Wintz. *Digital Image Processing, Second Edition*. Addison-Wesley, Reading, MA., 1989.

[10] V. Jacobson. Congestion avoidance and control. In *Proceedings of the SIGCOMM '88 Symposium*, pages 314–32, Aug. 1988.

[11] J. Postel and J. Reynolds. File Transfer Protocol (FTP). Request For Comments 959, USC Information Sciences Institute, Marina del Ray, Calif., Oct. 1985.

[12] W. K. Pratt. *Digital Image Processing*. John Wiley and Sons, New York, 1978.

[13] B. R. Schatz. Building an electronic community system. *Journal Management Information Systems*, 8(3):87–107, 1991.

[14] R. Schowengerdt. *Techniques for Image Processing and Classification in Remote Sensing*. Academic Press, Inc., 1983.

[15] C. J. Turner and L. L. Peterson. The effects of transfer encoding on image quality. In *Proceedings of the 2nd International Conference on Image Processing*, Sept. 1992.

[16] G. K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, Apr. 1991.

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