ITP: An Image Transport Protocol for the Internet

Suchitra Raman, Hari Balakrishnan, *Member, IEEE*, and Murari Srinivasan

Abstract— **Images account for a significant and growing fraction of Web downloads. The traditional approach to transporting images uses TCP, which provides a generic reliable, in-order byte-stream abstraction, but which is overly restrictive for image data. We analyze the progression of image quality at the receiver with time and show that the in-order delivery abstraction provided by a TCP-based approach prevents the receiver application from processing and rendering portions of an image when they actually arrive. The end result is that an image is rendered in bursts interspersed with long idle times rather than smoothly.**

This paper describes the design, implementation, and evaluation of the Image Transport Protocol (ITP) for image transmission over loss-prone congested or wireless networks. ITP improves user-perceived latency using application-level framing (ALF) and out-of-order Application Data Unit (ADU) delivery, achieving significantly better interactive performance as measured by the evolution of peak signal-to-noise ratio (PSNR) with time at the receiver. ITP runs over UDP, incorporates receiver-driven selective reliability, uses the Congestion Manager (CM) to adapt to network congestion, and is customizable for specific image formats (e.g., JPEG and JPEG2000). ITP enables a variety of new receiver post-processing algorithms such as error concealment that further improve the interactivity and responsiveness of reconstructed images. Performance experiments using our implementation across a variety of loss conditions demonstrate the benefits of ITP in improving the interactivity of image downloads at the receiver.

I. Introduction

IMAGES constitute a significant fraction of traffic on the World Wide Web. For example, one recent study showed that JPEG images account for about 31% of all bytes transferred and 16% of documents downloaded in a client Web trace [1]. The ability to transfer and render images on screen in a timely fashion is an important consideration for content providers and server operators because users surfing the Web care about interactive latency. At the same time, download latency must be minimized without compromising end-to-end congestion control, since congestion control is vital to maintaining the long-term stability of the Internet infrastructure. In addition, appropriate reaction to network congestion also allows image transfer applications to adapt well to available network conditions, perhaps by changing the format of transferred images to suit prevailing network conditions.

The HyperText Transport Protocol (HTTP) [2] uses the Transmission Control Protocol (TCP) [3] to transmit images on the Web. While the use of TCP achieves both reliable data delivery and good congestion control, these come at a cost—interactive latency is often significantly large and leads to images being rendered in "fits and starts" rather than in a smooth way. The reason for this is that TCP is ill-suited to transporting latency-sensitive images over loss-prone networks where losses occur because of congestion or packet corruption. When one or more segments in a window of transmitted data are lost in TCP, later segments often arrive out-of-order at the receiver. In general, these segments correspond to portions of an image that may be handled upon arrival by the application, but the in-order delivery abstraction imposed by TCP holds up the delivery of these out-of-order segments to the *application* until the earlier lost segments are recovered. As a result, the image decoder at the receiver cannot process information even though it is available at the lower transport layer. The image is therefore rendered in bursts interspersed with long delays, rather than smoothly. This motivates our work.

The TCP-like in-order delivery abstraction is indeed appropriate for certain image encodings, like the Graphical Interchange Format, GIF [4], in which incoming data at the receiver can only be handled in the order it was transmitted by the sender. However, while some compression formats are constrained in this manner, several others are not. Notable examples of formats that encourage out-of-order receiver processing include JPEG [5], [6] and the emerging JPEG2000 standard [7]. In these cases, a transport protocol that facilitates out-of-order data delivery allows the application to process and render portions of an image as they arrive, improving the interactivity and perceived responsiveness of image downloads. Such a protocol also enables the image decoder at the receiver to implement effective error concealment algorithms on partially received portions of an image, further improving perceived quality.

One commonly suggested approach to tackling this problem of in-order delivery is to extend existing TCP implementations and its application programming interface so that received data can be consumed out-of-order by the application. However, merely tweaking an in-order byte-stream protocol like TCP without any additional machinery is not adequate because out of order TCP segments received by the application in this manner do not correspond in any meaningful way to processible data units at the application level. Adapting TCP and providing an API for out-of-order delivery with receiver-driven reliability is a non-trivial task and the design of such a protocol would likely require significant changes to TCP.

We propose the Image Transport Protocol (ITP), a transport protocol in which application data unit (ADU) boundaries are exposed to the transport module, making it

This work was supported in part by an NSF CAREER award (No. 9984921), DARPA (Grant No. MDA972-99-1-0014), and IBM Corporation. An earlier version of this paper appeared at the International Conference on Network Protocols (ICNP), Osaka, Japan, November 2000.

Suchitra Raman is with Acopia Networks, Chelmsford, MA 01824 USA (e-mail: suchi@lcs.mit.edu).

Hari Balakrishnan is with the MIT Laboratory for Computer Science, 200 Technology Square, Cambridge, MA 02139 USA (e-mail: hari@lcs.mit.edu).

Murari Srinivasan is with Flarion Technologies, Bedminster, NJ 07921 USA (e-mail: murari_srinivasan@yahoo.com).

possible to perform meaningful out-of-order delivery. Because the transport protocol is aware of application framing boundaries, our approach expands on the application-level framing (ALF) philosophy, which proposes a one-to-one mapping from an ADU to a network packet or protocol data unit (PDU) [8]. However, ITP deviates from the TCP-like notion of reliable delivery and instead incorporates *selective reliability*, where the receiver is in control of deciding what is retransmitted from the sender.

Selective reliability is especially appropriate for heterogeneous network environments that will include a wide variety of clients with a large diversity in processing power, and allows the client to request application data that would benefit it the most, depending on its computational power and available suite of image decoding algorithms. Furthermore, image standards such as JPEG2000 support region-of-interest (ROI) coding that allows receivers to select portions of an image to be coded and rendered with higher fidelity.

Any deployable transport protocol must perform congestion control for the Internet to remain stable, which suggests that a significant amount of additional complexity would have to be designed and implemented in ITP. Fortunately, we are able to leverage the recently proposed Congestion Manager (CM) [9], [10] to perform stable, end-to-end congestion control.

In this paper, we describe the motivation, design, implementation, and evaluation of ITP, an ALF-based image transport protocol. Our key contributions are as follows.

- We present the design of ITP, a transport protocol that runs over UDP, incorporating out-of-order data delivery and receiver-controlled selective reliability. We have designed ITP so that it can be used with no modifications to higher layer protocols such as HTTP [11], [2] or FTP [12].
- We show how to tailor ITP for JPEG image transport, by introducing a framing strategy and tailoring the reliability protocol by scheduling request retransmissions.
- ITP's out-of-order delivery enables many receiver optimizations. We describe one such optimization in which missing portions of an image are interpolated using a simple error concealment algorithm.
- We present the measured performance of a user-level implementation of ITP across a range of network conditions that demonstrate that the rate of increase in PSNR with time is substantially higher for ITP compared to the in-order delivery of JPEG data.

The remainder of this paper is organized as follows. In Section II, we present empirical evidence in favor of our approach and discuss our design goals for ITP. Section III describes various aspects of the ITP protocol—out-of-order delivery, receiver-reliability, and congestion management. This is followed by a discussion on applying ITP to JPEG transport in Section IV. In Section V, we present the measured performance of ITP that demonstrates the advantages over the traditional TCP approach under a variety of conditions. Finally, we discuss related work in Section VI and conclude in Section VII.

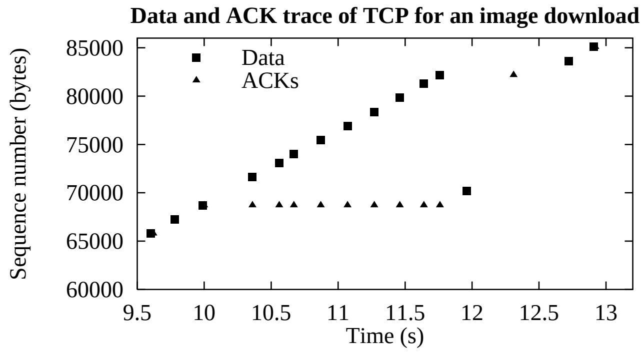

Fig. 1. Portion of packet sequence trace of a TCP transfer of an image.

II. Design Considerations

We start by motivating our approach by highlighting the disadvantages of using TCP for image transfers. The main drawback of using TCP for image downloads is that its in-order delivery model interferes with user interactivity. To demonstrate this, we conducted an experiment across a twenty-hop Internet path to download a 140 KByte image using HTTP/1.1 [2] running over TCP. The loss rate experienced by this connection was 2.3%, only three segments were lost during the entire transfer, and there were no sender retransmission timeouts.

Figure 1 shows a portion of the packet sequence trace obtained using `tcpdump` [13] running at the receiver. We see a transmission window in which exactly one segment was lost, and all subsequent segments were received, causing the receiver to generate a sequence of duplicate acknowledgments (ACKs). There were ten out-of-sequence segments received and waiting in the TCP socket buffer, none of which was delivered to the image decoder application until the lost segment was received via a (fast) retransmission almost 2.2 seconds after the loss. During this time, the user saw no progress, but a discontinuous spurt occurred once this lost segment was retransmitted to the receiver, and several kilobytes worth of image data were passed up to the application. This is the behavior we would like to avoid in the interest of better user interactivity.

To understand how ordering semantics influence the perceptual quality of the image, we conducted a second experiment where the image is downloaded over TCP and studied the evolution of image "quality," as measured by peak signal-to-noise ratio (PSNR) [14] with respect to the original transmitted image. Figure 2 shows this for a transfer that experiences a 15% loss rate. We find that the quality remains unchanged for most of the transfer, due to an early segment loss, but rapidly rises upon recovery of that lost segment. A smoother evolution in PSNR, as in the "ideal" transfer which does out-of-order delivery is desirable for better interactivity.

We observe that a design in which the underlying transport protocol delivers out-of-sequence data to the application might avoid the perceived latency buildup. In order to

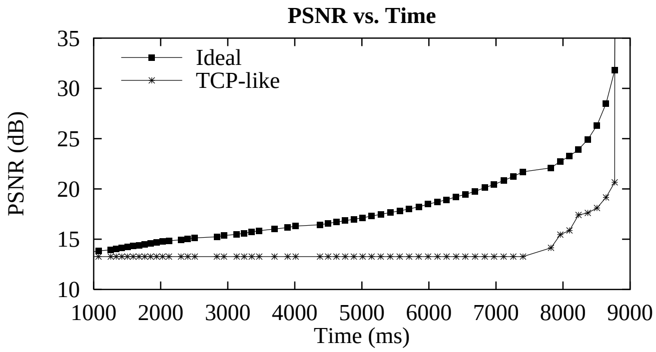

Fig. 2. PSNR evolution of the rendered image at the receiver for a TCP transfer with 15% loss rate.

do this, the transport "layer" (or module) must be made aware of the application framing boundaries, such that each data unit is independently processible by the receiver.

The following considerations directed the design of ITP.

1. *Support out-of-order delivery of ADUs to the application, while efficiently accommodating ADUs larger than a single unfragmented packet.*

Our first requirement is that the protocol accommodate out-of-order delivery, but does so in a way that allows the receiver application to make sense of the mis-ordered data units it receives. In the pure ALF model [8], each ADU is matched to the size of a protocol data unit (PDU) used by the transport protocol. This implies that there is no "coupling" between two packets and that they can be processed in any order. Unfortunately, it is difficult to ensure that an ADU is always well matched to a PDU because the former depends on the convenience of the application designer and what is meaningful to the application, while the latter should not be too much larger (if at all) than the largest datagram that can be sent unfragmented.

2. *Support receiver-controlled selective reliability.*

When packets are lost, there are two possible ways of handling retransmissions. The conventional approach is for the sender to detect losses and retransmit them in the order in which they were detected. While this works well for protocols like TCP that simply deliver all the data sequentially to a receiver, interactive image transfers are better served by a protocol that allows the receiving application (and user) to control the retransmissions from the sender. For example, a user should be able to express interest in a particular region of an image, causing the transport protocol to prioritize the transmission of the corresponding data over others.

3. *Support customization to different image formats.*

There are many different image formats that can benefit from out-of-order processing, each of which may embed format-specific information in the protocol. For example, the JPEG format uses an optional special delimiter called a *restart marker*, which signifies the start of each independently processible unit to the decoder. Such format- or application-specific information should be made available to the receiver in a suitable way, without sacrificing generality in the basic protocol.

In ITP, this is done as in the Real-time Transport Protocol (RTP) [15]; a base header is customized by individual application protocols, with profile-specific extension headers incorporating additional information.

4. *Application and higher-layer protocol independence.*

While this work is motivated by interactive image downloads on the Web, our goal is for ITP to be useful as a transport protocol for not just HTTP but other higher-layer protocols as well. Furthermore, we do not require any changes to the HTTP specification, and would like to be able to replace HTTP's use of TCP with ITP at the transport layer for image data. We use a duplex ITP connection to carry HTTP request messages such as GET and POST, as well as HTTP responses, in much the same way that HTTP uses bi-directional TCP connections for this.

5. *Sound congestion control.*

Finally, congestion-controlled transmissions are important for deploying any transport protocol on the Internet. But rather than reinvent complex machinery for congestion management (a look at many of the subtle bugs in TCP congestion control implementations that researchers have discovered over the years shows that this is not straightforward [16]), we leverage the recently developed Congestion Manager (CM) architecture [9]. The CM abstracts away all congestion control into a trusted kernel module independent of transport protocol, and provides a general API for applications to learn about and adapt to changing network conditions [10].

III. ITP Design

In this section, we describe the design and internal architecture of ITP, and the techniques used to meet the aforementioned design goals. ITP is designed as a modular user-level library that is linked by the sender and receiver application. The overall system architecture is shown in Figure 3, which includes an example of an application protocol such as HTTP or FTP using ITP for data with MIME type "image/jpeg" and TCP for other data. It is important to note that ITP "slides in" to replace TCP in a way that requires no change to the specification of a higher-layer protocol like HTTP or FTP. A browser initiates an ITP connection in place of a TCP connection if a JPEG image is to be transferred. The HTTP server initiates an active open on UDP port 80 and waits for client requests that are made using the HTTP/ITP/UDP protocol.

A. Out-of-order Delivery

Providing an out-of-order delivery abstraction at the granularity of a byte, makes it hard for the application to infer what application data units an arbitrary incoming sequence of bytes corresponds to. The application handles data in granularities of an ADU, so ITP provides an API by which an application can send or receive a complete ADU.

The sending application invokes `itp_send()` to send an ADU to the receiver. Before shipping the ADU, ITP incorporates a header, shown in Figure 4 that includes an incrementing ADU sequence number and ADU length. The

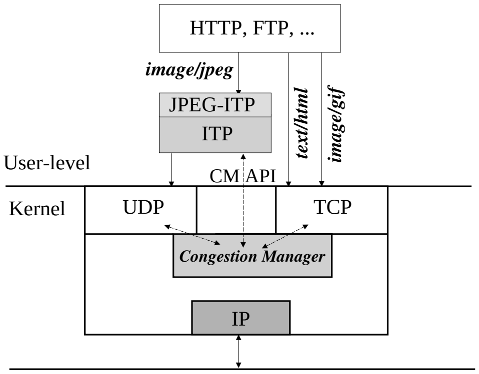

Fig. 3. The system architecture showing ITP, its customization for JPEG, and how HTTP uses it instead of TCP for MIME type "image/jpeg" while using a conventional TCP transport for other data types. All HTTP protocol messages are sent over ITP, not just the actual image data, which means that ITP replaces TCP as the transport protocol for this data type.

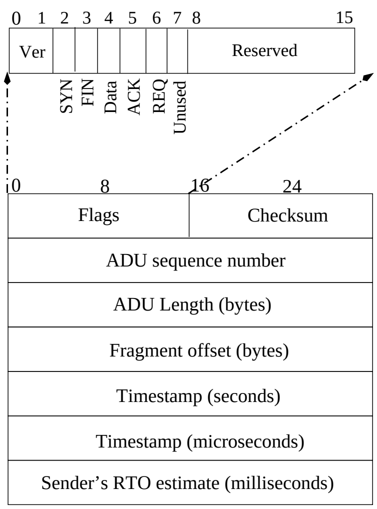

Fig. 4. The 28-byte generic ITP transport header contains meta-data pertaining to each fragment, as well as the ADU that the fragment belongs to, such as the ADU sequence number and length, the fragment offset within the ADU, a sender timestamp, and the sender's estimate of the retransmission timeout.

sequence number and length of an ADU are used by the receiver to detect the loss of an ADU or the loss of a sequence of bytes within the ADU, perform reassembly within an ADU, and verify that the complete ADU has arrived.

When a complete ADU arrives at the receiver, the ITP receiver invokes a well-known callback function implemented by the application, called `itp_app_notify()`. In response, the application calls an ITP library function `itp_read()` to read the incoming ADU into its own buffers, and returns control to ITP. This interaction is shown in Figure 5. The important point to note is that this sequence of steps occurs when a complete ADU arrives at the receiver, *independent* of the order in which it was transmitted from the sender.

Unfortunately, not all ADUs are small enough to fit in

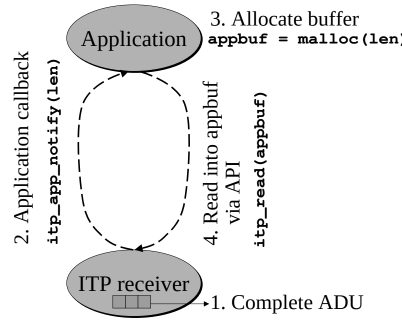

Fig. 5. The sequence of operations when a complete ADU arrives at the ITP receiver.

one PDU. This requires that any ADU larger than a PDU be fragmented into PDU-sized units before transmission. Using arbitrarily-sized ADUs as the granularity of loss recovery is inefficient. Consider for example an ADU transmitted by the transport protocol that was fragmented by a lower layer for transmission, and exactly one of the fragments was lost in transit. The receiver must ask for the entire ADU to be retransmitted if the unit of naming and transmission by the transport layer is an ADU, thereby degrading protocol goodput. Rather than suffer poor performance caused by redundant retransmissions, ITP bridges the mismatch between network-supported packet sizes and application-defined data units by breaking up an ADU into *fragments* no bigger than the maximum transmission unit of the path and identifying each fragment by its byte-offset and length within an ADU as well as the ADU sequence number. Path MTU discovery [17] can be used to determine this value between a pair of hosts on the Internet. We emphasize that this is done to avoid inefficiencies in retransmission, but is not exposed to the receiving application. As a result, applications are not forced to limit their framing to network packet sizes, and partial ADU data are not visible to them.

B. Reliability

One of the design goals in ITP is to put the receiver in control of loss recovery which suggests a protocol based on *retransmission request* messages sent from the receiver. In addition to loss recovery, ITP must also reliably handle connection establishment and termination, as well as host failures and subsequent recovery without compromising the integrity of delivered data. We incorporate TCP-like connection establishment and termination mechanisms for this; details of this are in [18].

All retransmissions in ITP occur only upon receipt of a retransmission request from the receiver, which names a requested fragment using its ADU sequence number, fragment offset, and fragment length. While many losses can be detected at the receiver using a data-driven mechanism that observes gaps in the received sequence of ADUs and fragments, not all losses can be detected in this manner.

In particular, when the last fragment or "tail" of a burst of fragments transmitted by a sender is lost, a retransmission timer is required. Losses of previous retransmissions similarly require timer-based recovery.

One possible design is for the receiver to perform all data-driven loss recovery, and for the sender to perform all timer-based retransmissions. However, this contradicts our goal of receiver-controlled reliability because the sender has no knowledge of the fragments most useful to the receiver. Unless we incorporate additional complex machinery by which a receiver can explicitly convey this information to the sender, the sender may retransmit old and uninteresting data upon a timeout.

Our solution to this problem is to move all timer handling to the receiver. If the receiver detects no activity for a timeout duration, a retransmission request is sent. If no gaps are detected in the received ADU stream, a retransmission request is sent for the next expected ADU, i.e., 1 + last ADU sequence number received, thereby initiating recovery from a tail loss. Since the retransmission timer is always active, this message is repeated periodically until the receiver is ready to terminate.

It is rather difficult for accurate round-trip time estimation to be performed at the receiver when data flows only from sender to receiver. Hence, the ITP sender calculates the retransmission timeout (RTO) as in TCP with the timestamp option [19], and passes this RTO to the receiver in the ITP header (Figure 4).

ITP also incorporates "data-driven" retransmission requests. To do this, the receiver maintains a list of incomplete and missing ADUs. When a fragment is received, missing fragments or ADUs are detected by looking up the data structure. The receiver now has three tasks: (i) decide whether it is time to ask for the fragment, (ii) decide how many fragments to ask for, and (iii) if at least one fragment can be requested at this time, decide which fragments to request.

Two considerations dictate whether it is time to ask for a fragment. First, if a request has already been made for the fragment, it should not be made again unless an RTO has elapsed since the first request. Second, packets may get reordered on the Internet [20], and the receiver must guard against asking for a reordered (but not lost) fragment. The approach in TCP is to wait for a threshold number (three) of duplicate ACKs and retransmit the first unacknowledged segment. Unfortunately, this does not work well when windows are small or when ADUs are small in size (as is often the case for ITP applications). Our solution to this problem is motivated by the observation by Paxson that a small delay before sending an ACK in TCP often accounts for reordered segments [21]. ITP modifies this approach by adapting it to the transmission rate r (in fragments/sec) from the sender, which it monitors using an exponentially-weighted moving average filter. The receiver waits for a duration equal to $3/r$ seconds before sending a request, allowing for a typical number of reordered fragments to arrive and cancel a pending retransmission request.

A difficult part of ITP loss recovery is to decide which fragment to request at any time among the missing ones. This is difficult because of the tension between application-specificity and generality. We would like to put the application in control of what to request, but save each application the trouble of writing the complex loss detection code. Furthermore, we would like to provide a reasonable default behavior to handle applications that do not care to customize their reliability schedules.

ITP provides a simple default scheduling algorithm for retransmission requests in which requests are made for fragments from all the missing ADUs starting from the most recent one and progressing in sequence to the least recent, subject to the above conditions of not requesting them too soon. More importantly, ITP also allows application-specific customization of reliability, as described in Section IV-B for JPEG.

C. Congestion Control

ITP uses the Congestion Manager (CM) for congestion control, using the CM API to adapt to network conditions and to inform the CM about the status of transmissions and losses [22], [10]. Since ITP reliability is receiver-based, there is no need for positive ACKs from the receiver to the sender for reliability. ACKs from the receiver are solely for congestion control and estimating round-trip times; these are needed because the CM congestion controller we use implements a window-based congestion control algorithm. The CM requires the cooperation of the application in determining the state of the network, as described in [10]. By informing the ITP sender about the status of transmissions, an ITP ACK allows the ITP sender to update CM state. When the ITP sender receives an ACK, it calculates how many bytes have cleared the "pipe" and calls `cm_update()` to inform the CM of this.

When a retransmission request arrives at the sender, the sender infers that packet losses have occurred, attributes them to congestion (as in TCP), and invokes `cm_update()` with the `lossmode` parameter set to CM_TRANSIENT, signifying transient congestion. In a CM-based transport protocol where timeouts occur at the sender, the expected behavior is to use cm_update() with the `lossmode` parameter set to CM_PERSISTENT, signifying persistent congestion. In ITP, the sender never times out, only the receiver does. The sender only sees a request for retransmission arriving after a timeout at the receiver, so when a retransmission request arrives, it needs to determine if that occurred after a timeout or because of out-of-sequence data. We solve this problem by calculating the elapsed time since the last time there was any activity on the connection from the peer, and if this time is greater than the retransmission timeout value, then the CM is informed about persistent congestion. Figure 6 shows what the ITP sender does when it receives a request for retransmission.

IV. JPEG Transport using ITP

In this section, we discuss how to tailor ITP for transmitting JPEG images. JPEG was developed in the early 1990s by a committee within the International Telecommunica-

```
PROCESSRXMITREQ(fragment)
    Send requested fragment via cm_send();
    InformCM();

INFORMCM()
    now ← current_time;
    if (now − last_activity > timeout_duration)
        cm_update(..., CM_PERSISTENT, ...);
    else
        cm_update(..., CM_TRANSIENT, ...);
```

Fig. 6. How the ITP sender handles a retransmission request.

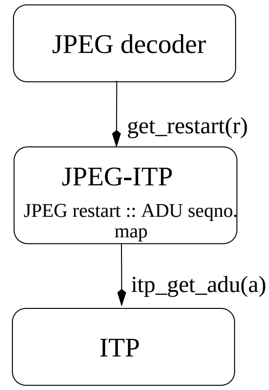

Fig. 7. JPEG-ITP maintains a mapping of restart intervals to ADU sequence numbers. The JPEG decoder specifies recovery priorities based on application-level considerations, which is used to guide ITP's request scheduling.

tions Union, and has found widespread acceptance for use on the Web. The compression algorithm uses block-wise discrete cosine transform (DCT) operations, quantization, and entropy coding [23]. JPEG-ITP is the customization of ITP by introducing a JPEG-specific framing strategy based on restart markers and tailoring the retransmission protocol by scheduling retransmission requests.

A. Framing

JPEG uses entropy coding and the resulting compressed bitstream consists of a sequence of variable-length code words. Packet losses often result in catastrophic loss if pieces of the bitstream are missing at the decoder. Arbitrarily breaking an image bitstream into fixed-size ADUs does not work because of dependencies between them. However, JPEG uses *restart markers* to allow decoders to resynchronize when confronted with an ambiguous or corrupted JPEG bitstream, which can result from partial loss of an entropy-coded segment of the bitstream. The introduction of restart markers helps localize the effects of the packet loss or error to a specific sub-portion of the rendered image. This segmentation of the bitstream into independent restart intervals also facilitates out-of-order processing by the application layer. The approach used by JPEG to achieve loss resilience provides a natural solution to our framing problem.

When an image is segmented into restart intervals, each restart interval is independently processible by the application and naturally maps to an ADU. The image decoder is able to decode and render those parts of the image for which it receives information without waiting for packets to be delivered in order. The base ITP header is extended with a JPEG-specific header that carries framing information, which includes the spatial position of a 2-byte restart interval identifier.

Our implementation of JPEG-ITP uses 8-bit gray-scale images in the baseline sequential mode of JPEG. We require that the image server store JPEG images with periodic restart markers. This requirement is easy to meet, since a server can easily transcode offline any JPEG image (using the `jpegtran` utility) to obtain a version with markers. When these markers occur at the end of every row of blocks, each restart interval corresponds to a "stripe" of the image. These marker-equipped bistreams produce exactly the same rendered images as the original ones when there are no losses. Since JPEG uses a blocksize of 8x8 pixels, each restart interval represents 8 pixel rows of an image. We use the sequence of bits between two restart markers to define an ADU, since any two of these intervals can be independently decoded. Our placement of restart markers achieves the effect of rendering an image in horizontal rows.

B. Scheduling

As discussed in Section III, ITP allows the application to specify the priorities of different ADUs during recovery. We describe how this is achieved in JPEG-ITP. Figure 7 shows the key interfaces between ITP and JPEG-ITP, and between JPEG-ITP and the decoder. ITP handles all fragments and makes only complete ADUs visible to JPEG-ITP. To preserve its generality, we do not expose application-specific ADU names to ITP. Thus, when a missing ADU needs to be recovered by the decoder, JPEG-ITP needs to map the restart interval number to an ITP ADU sequence number. To do this, the JPEG-ITP sender reliably transmits this mapping as the first ADU of the connection, before transmitting the image ADUs. This name map is used to schedule ITP retransmission requests.

ITP maintains a priority list of the retransmission schedule by exporting an asynchronous API function `itp_get_adu()` that customized protocols like JPEG-ITP and applications can use to inform ITP of the desired ADU. ITP uses this priority information to schedule requests for missing fragments from these ADUs ahead of others. In addition, JPEG-ITP exports an API function to the decoder that allows the latter to specify restart intervals that must be prioritized during recovery, e.g., if the decoder uses error concealment as in Section IV-C, this is used to preferentially request ADUs that have not been interpolated from the existing partial image.

C. Error Concealment

Out-of-order delivery allows the JPEG decoder to refine a partial image using error concealment based on interpolation techniques. Portions of the image corresponding to the received ADUs are decoded and rendered. Before rendering, a post-processing step is applied to the image to conceal lost stripes. Error concealment exploits spatial redundancy in images and aims to increase the perceptual quality of the rendered image.

Each missing pixel value is the result of a linear interpolation of its neighbors. This step is applied to all missing restart intervals at the receiver. Therefore, in 2-D, the missing pixel $x_{i,j}$ is given by:

$$x_{i,j} = \frac{x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1}}{4} \tag{1}$$

The boundary conditions are determined by the pixel values of neighboring blocks. Using the lexicographic ordering of pixels in a block, $\mathbf{x} = \{x_{0,0}, x_{0,1}, ...x_{0,B-1}, x_{1,0}..., x_{B-1,B-2}, x_{B-1,B-1}\}$, the estimate of the missing block may be computed as

$$\hat{\mathbf{x}} = A^{-1}\mathbf{c} \tag{2}$$

where A is a block tri-diagonal matrix given by

$$A = \begin{bmatrix} L & I & O & \cdots & \\ I & L & I & O & \cdots \\ O & I & L & I & O \\ \cdots & O & I & L & I \\ & \cdots & O & I & L \end{bmatrix} \tag{3}$$

and L is a 8x8 tri-diagonal matrix formed from $\{1, -4, 1\}$.

$\mathbf{c}$ is a vector that represents the boundary conditions imposed by the pixels *above*(u), *below*(d), to the $left(l)$ and to the $right(r)$ of the current block.

$$\begin{aligned} c(0,0) &= l(0) + u(0) \\ c(0,B-1) &= r(0) + u(B-1) \\ c(B-1,0) &= l(B-1) + d(0) \\ c(B-1,B-1) &= r(B-1) + d(B-1) \end{aligned}$$

D. Other Formats

We have described a simple framing strategy and further refinement using error concealment scheme for JPEG over ITP. The same techniques also extend to progressive JPEG images. In progressive JPEG, the quantized DCT coefficients corresponding to each block are divided into a series of scans. These scans may either represent different frequencies (low to high), or different bit-planes of the quantized coefficients (most significant to least significant bits). A coarse representation of the image is rendered with the receipt of the first scan, which is successively refined as subsequent scans arrive. Each scan can be segmented into restart intervals, which results in the ability to process and render out-of-order within a scan, leading to quicker response times and interactivity. Error-concealment can be carried out in a multi-resolution manner by performing concealment within one scan at a time.

Similar techniques are also possible for transmission of JPEG2000, which is a recent proposal for wavelet-based image coding scheme that results in higher compression ratios and better fidelity. The standard supports several features such as layered coding and "region of interest" (ROI) coding. Designing transport support for ROI coding requires customized scheduling of retransmission requests at the receiver, which is provided by ITP.

V. Performance

In this section, we evaluate our implementation of ITP under a variety of network loss rates. Our implementation of ITP performs out-of-order data delivery at the receiver and uses the averaging method to interpolate missing packets at the receiver. We have customized ITP for JPEG transport where the images contain restart intervals.

A. Peak Signal-to-Noise Ratio (PSNR)

Image quality is often measured using a metric known as the PSNR. Consider an image whose pixel values are denoted by $x(i,j)$ and a compressed version of the same image whose pixel values are $\hat{x}(i,j)$. The PSNR quality of the compressed image (in dB) is:

$$\mathrm{PSNR} = 10 \times \log_{10} \frac{255^2}{E||x(i,j) - \hat{x}(i,j)||^2} \tag{4}$$

In our experiments, we use PSNR with respect to the transmitted image as the metric to measure the quality of the image at the receiver. Note that PSNR is inversely proportional to the mean-square distortion between the images, which is given by the expression in the denominator of Equation 4. When the two images being compared are identical, e.g., at the end of the transfer when all blocks from the transmitted image have been received, the mean-square distortion is 0 and the PSNR becomes ∞. We recognize that PSNR does not always accurately model perceptual quality, but use it because it is a commonly used metric in the signal processing literature.

B. Experimental Results

We measure the evolution of instantaneous PSNR as the JPEG image download progresses. When JPEG-ITP receives a complete restart interval from ITP, it is passed to the decoder. The decoder output is processed to fill in missing intervals using the error concealment step explained earlier and the image is updated. We measure PSNR with respect to the original JPEG image transmitted under three scenarios: (i) when in-order delivery is enforced, (ii) when out-of-order delivery is allowed, and (iii) when error concealment is performed on the mis-ordered data units.

Figure 8 shows the results of this experiment under a variety of loss rates. We use a simple Bernoulli loss model where each packet is dropped at the receiver with an independent probability given by the average loss rate.

We find that across a range of loss rates between 5% and 30%, TCP-like delivery causes the quality of the rendered

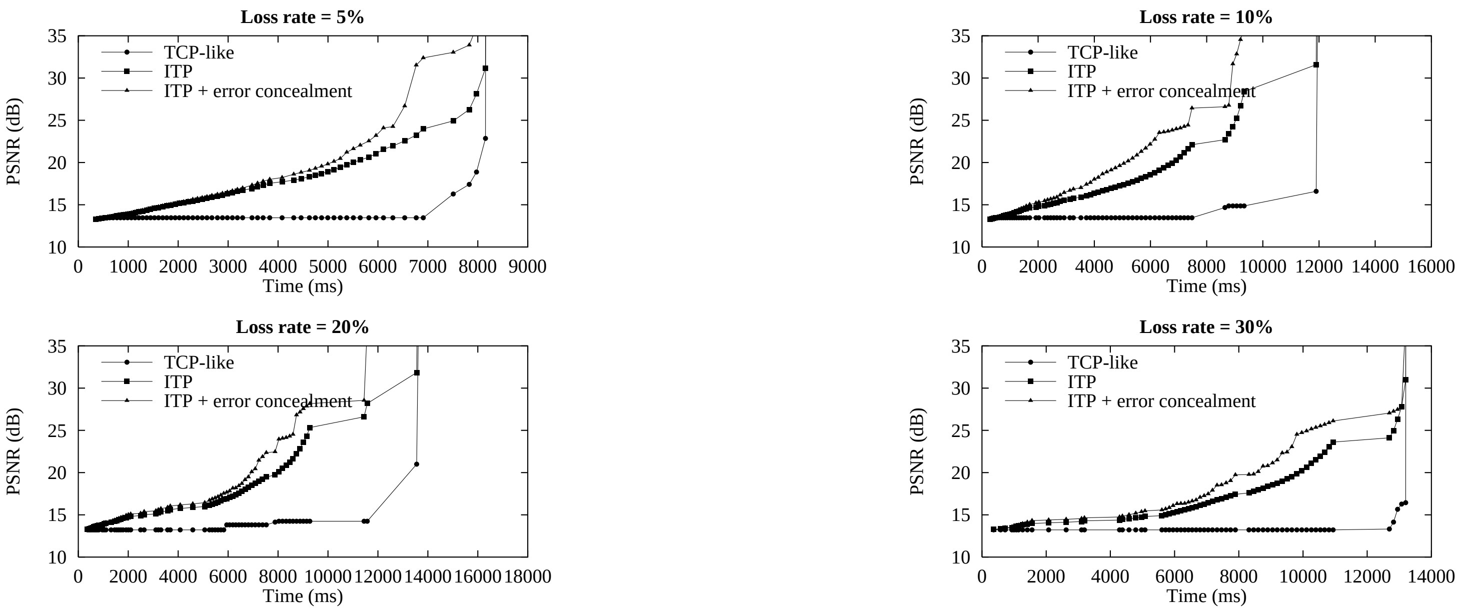

Fig. 8. PSNR vs. Time for ITP and a TCP-like transport that enforces in-order delivery. The quality of the image (as measured by PSNR) is identical in all three scenarios at the start and at the end of the transfer. However, the sample paths differ — the best performance is seen with ITP optimized with error concealment, while TCP shows the poorest performance. ITP shows a steady improvement in quality, and is therefore perceptually superior for interactive applications such as the Web.

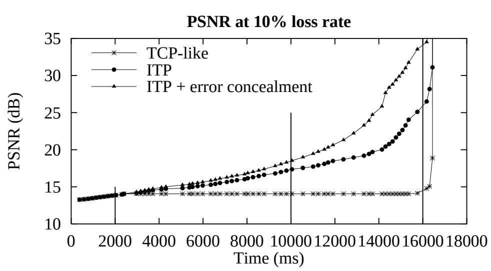

Fig. 10. PSNR corresponding to the snapshots shown in Figure 9. Starting at almost identical image snapshots at $2s$, the ITP image (with and without error concealment) progress steadily in quality, while the TCP-delivered image only catches up close to completion time.

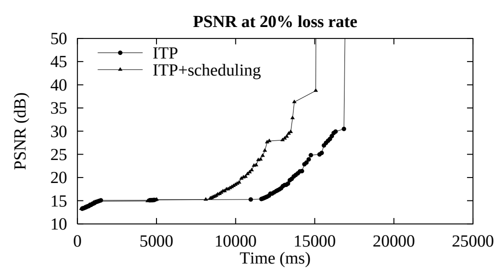

Fig. 11. When receiver request scheduling takes into consideration those "stripes" that cannot be interpolated, the quality of the rendered image can be improved by 5–15 dB.

image to remain low for extended intervals of time. In comparison, ITP with out-of-order delivery shows a smoother evolution of PSNR during the transfer. In addition, the PSNR of the ITP-delivered image is superior to that delivered by TCP while the transfer is in progress, becoming identical only at the end of the transfer, as expected. This smooth evolution of quality makes ITP better suited for interactive image downloads. When error concealment is applied as an added optimization on the partial image, we find that the benefits are between 2–8 dB. In combination, the two techniques outperform TCP by 10–15 dB.

Figure 9 shows the progression of displayed images for the three different scenarios and Figure 10 shows the corresponding PSNR values. Starting with almost identical image snapshots at $2s$, the ITP-delivered images (with and without error concealment) show steady improvement in quality relative to the TCP-delivered snapshot. At $10s$, the ITP image is 3.3 dB and a further improvement of 1.3 dB is achieved through interpolation on the partial image. As we can see from the image, the benefits of interpolation are greater when more of the image is available, which further strengthens the case for out-of-order delivery in ITP. The ITP images continue to improve and at $12s$, they are 12 dB (without error concealment) and 20 dB (with error concealment) better than the corresponding TCP-delivered images.

We also conducted a transfer across a 1.5 Mbps link to study the effect of receiver scheduling. Here, the receiver prioritizes requests for data items that cannot be concealed using interpolation. The results are shown in Figure 11.

VI. Related work

The so-called CATOCS debate on ordering semantics in the context of multicast protocols drew much attention a few years ago [24], [25], [26]. Cheriton and Skeen argued

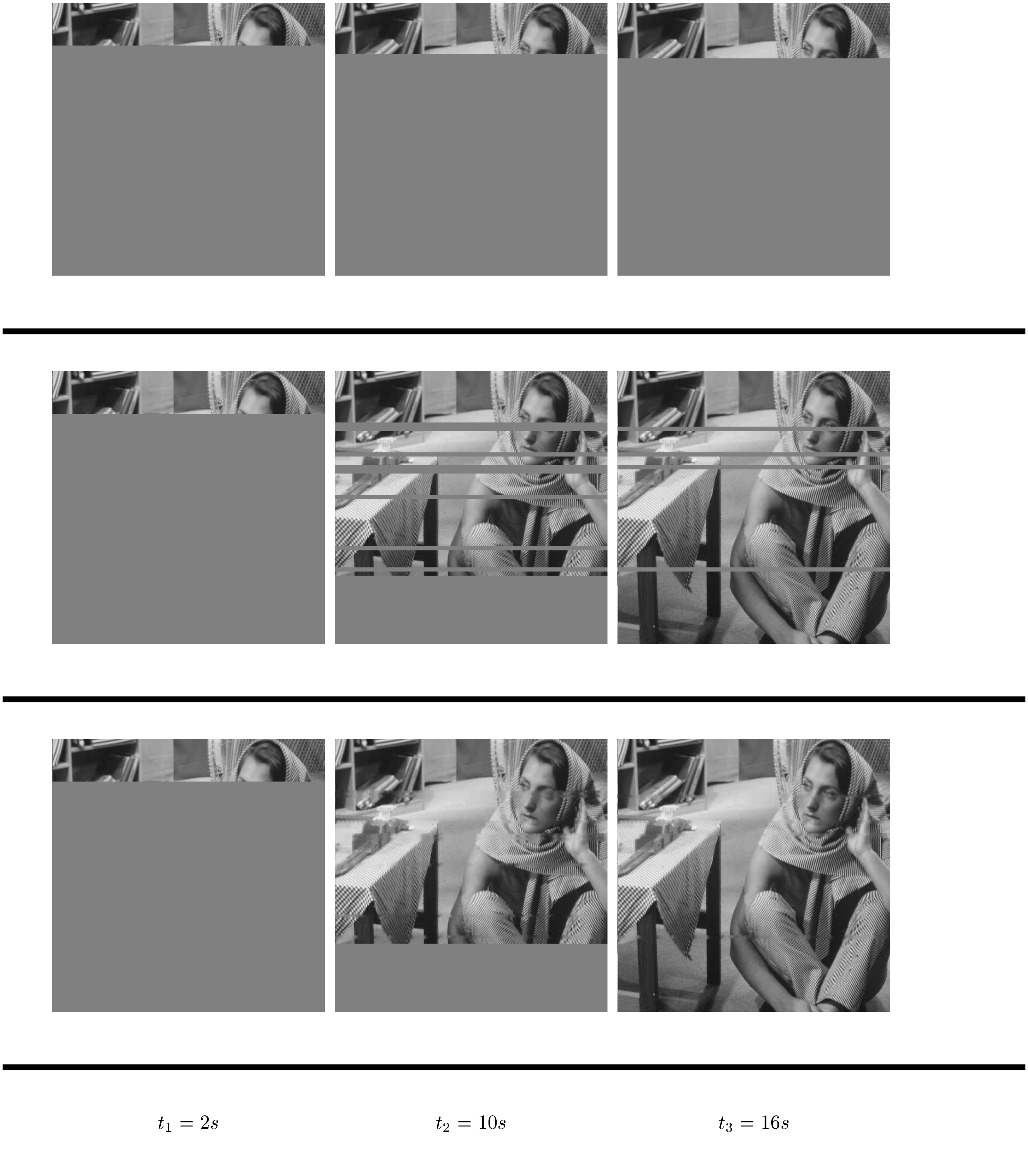

Fig. 9. Snapshots of the displayed image with a TCP-like transport (first row), with ITP (second row), and with ITP enhanced with error concealment (last row) at 10% loss rate. The entire transfer of the 184 KB image takes 16.57s to complete.

that ordering semantics are better handled by the application and that enforcing an arbitrarily chosen ordering rule results in performance problems [24]. In our work, we reinforce this approach to protocol design and refrain from imposing a particular ordering semantics across all applications.

RDP [27], [28] is a reliable datagram protocol intended for efficient bulk transfer of data for remote debugging-style applications. RDP implements sender-driven reliability and does not support receiver-tailored nor application-controlled reliability. NETBLT [29] is a receiver-based reliable transport protocol that uses in-order data delivery and performs rate-based congestion control.

There has been much recent work on Web data transport for in-order delivery, most of which address the problems posed to congestion control by short transaction sizes and concurrent streams. Persistent-connection HTTP [30], part of HTTP/1.1 [2], attempts to solve this using a single TCP connection, but this causes an undesirable coupling between logically different streams because it serializes concurrent data delivery. The MEMUX protocol (derived from Web MUX [31] proposes to deliver multiplexed bidirectional reliable ordered message streams over a bidirectional reliable ordered byte stream protocol such as TCP [32]. A recent proposal to extend RTP [15], an Internet standard for streaming media with a negative acknowledgment-based selective reliability is described in [33].

The WebTP protocol aims to replace HTTP and TCP with a single customizable receiver-driven transport protocol [34]. WebTP handles only client-server transactions and not other forms of interactive Web transactions such as "push" applications. It is not a true transport layer (like TCP) that can be used by different session (or application) protocols like HTTP or FTP, since it integrates the session and transport functionality together. In addition, WebTP advocates maintaining the congestion window at the receiver transport layer, which makes it hard to share with other transport protocols and applications.

In contrast, our work is motivated by the philosophy that one transport/session protocol does not fit all applications, and that the only function that *all* transport protocols *must* perform is congestion management. The CM extracts this commonality into a trusted kernel module [9], permitting great heterogeneity in transport and application protocols customized to different data types (e.g., it is appropriate to continue using TCP for applications that need reliable, in-order delivery). The CM API allows these protocols to share bandwidth, learn from each other about network conditions, and dynamically partition available bandwidth amongst concurrent flows.

While much work has been done on video transmission, image transport has received little attention in the past. Turner and Peterson describe an end-to-end scheme for image encoding, compression, and transmission, tuned especially for links with large delay [35]. They develop a retransmission-free strategy based on forward error correction. Han and Messerschmitt propose a progressively reliable transport protocol (PRTP) for joint source-channel coding over a noisy, bandwidth constrained channel. This protocol delivers multiple versions of a packet with statistically increasing reliability and provides reliable, ordered delivery of images over bursty wireless channels [36]. The Fast and Lossy Internet Image Transmission protocol (FLIIT) [37] improves the perceived delay of a download by eliminating retransmissions. Instead, the FLIIT sender strategically shields "important" portions of the image data, for example, by applying FEC to the high order bits of the DC channels of the image.

Finally, we observe that several highly sophisticated error concealment techniques have been proposed in the literature, especially for video. For example, in [38], the authors propose the use of a Markov Random Field image model and optimally interpolate the missing pixels. The essence of our scheme, however, is on simplicity and improving interactivity (rather than precision), for which we find empirically that our simple interpolation strategy seems to work well.

VII. Conclusion

We argued that the reliable, in-order byte stream abstraction provided by TCP is overly restrictive for richer data types such as image data. Several image encodings such as sequential and progressive JPEG and JPEG2000 are designed to decode partially received, out-of-order image data. To improve the perceptual quality of the image during a download, we proposed a novel Image Transport Protocol (ITP). ITP uses an application data unit (ADU) as the unit of processing and delivery to the application by exposing application framing boundaries to the transport protocol. This enables the receiver to process ADUs out of order. ITP can be used as a transport protocol for HTTP and is designed to be independent of the higher-layer application or session protocol. ITP relies on the Congestion Manager (CM) to perform safe and stable congestion control, making it a viable transport protocol for use on the Internet today.

We showed how ITP can be customized for specific image formats such as JPEG. Out-of-order processing facilitates effective error concealment at the receiver that further improve the download quality of an image. We have implemented ITP as a user-level library that invokes the CM API for congestion control. Our performance evaluation of ITP demonstrates its benefits over the traditional in-order delivery approach, as measured by the peak signal-to-noise ratio (PSNR) of the received image.

In summary, ITP is a general-purpose, selectively-reliable transport protocol that can be applied to diverse data types. Our design and implementation provide a generic congestion-controlled transport substrate that can be tailored for specific data types. We believe that the ideas embedded in ITP will be applicable to other application domains for applications requiring good interactive performance in the face of varying network bandwidths and packet loss rates. One example of this is in Internet video using inter-frame compression formats like MPEG-2 or MPEG-4, where the loss of certain important frames may be recovered via ITP-like retransmissions for better interactive response [39].

References

[1] S. Gribble and E. Brewer, "System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace," in *Proc. 1997 Usenix Symposium on Internet Technologies and Systems*, Dec. 1997.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, Jan 1997, RFC-2068.

[3] J. B. Postel, *Transmission Control Protocol*, SRI International, Menlo Park, CA, Aug. 1989, RFC-793.

[4] "Graphics Interchange Format (SM), Version 89a," 1990.

[5] Wallace, G., "The JPEG Still Picture Compression Standard," *Communications of the ACM*, April 1991.

[6] William B. Pennebaker and Joan L. Mitchell, *JPEG : Still Image Data Compression Standard*, Van Nostrand Reinhold, 1992.

[7] "JPEG2000 Links," http://www.jpeg.org/JPEG2000.htm.

[8] David D. Clark and David L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," in *Proceedings of SIGCOMM '90*, Philadelphia, PA, Sept. 1990, ACM.

[9] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," in *Proceedings of SIGCOMM 1999*, Cambridge, MA, Sep 1999, ACM.

[10] H. Balakrishnan and S. Seshan, *The Congestion Manager*, Internet Engineering Task Force, June 2001, RFC 3124.

[11] T. Berners-Lee, R. Fielding, and H. Frystyk, *Hypertext Transfer Protocol–HTTP/1.0*, Internet Engineering Task Force, May 1996, RFC 1945.

[12] J. B. Postel and J. Reynolds, *File Transfer Protocol (FTP)*, Internet Engineering Task Force, Oct 1985, RFC 959.

[13] Van Jacobson, Craig Leres, and Steven McCanne, *TCPDUMP(1)*, Available via anonymous ftp from `ftp.ee.lbl.gov`, June 1989.

[14] Khalid Sayood, *Introduction to Data Compression*, Morgan Kaufmann, 1996.

[15] Henning Schulzrinne, Steve Casner, Ron Frederick, and Van Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, Internet Engineering Task Force, Audio-Video Transport Working Group, Jan. 1996, RFC-1889.

[16] V. Paxson, "Automated Packet Trace Analysis of TCP Implementations," in *Proc. ACM SIGCOMM '97*, Sept. 1997.

[17] J. C. Mogul and S. E. Deering, *Path MTU Discovery*, SRI International, Menlo Park, CA, Apr. 1990, RFC-1191.

[18] Suchitra Raman, *A Framework for Interactive Multicast Data Transport in the Internet*, Ph.D. thesis, University of California, Berkeley, May 2000.

[19] V. Jacobson, R. Braden, and D. Borman, *TCP Extensions for High Performance*, Internet Engineering Task Force, May 1992, RFC 1323.

[20] V. Paxson, "End-to-End Routing Behavior in the Internet," in *Proc. ACM SIGCOMM '96*, Aug. 1996.

[21] V. Paxson, "End-to-End Internet Packet Dynamics," in *Proc. ACM SIGCOMM '97*, Sept. 1997.

[22] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan, "System support for bandwidth management and content adaptation in Internet applications," in *Proc. Symposium on Operating Systems Design and Implementation*, October 2000.

[23] William B. Pennebaker and Joan L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, 1993.

[24] D. Cheriton and D. Skeen, "Understanding the Limitations of Causally and Totally Ordered Communication Systems," *Proc. 14th ACM Symposium on Operating Systems Principles*, pp. 44–57, Dec 1993.

[25] K. Birman, "A Response to Cheriton and Skeen's Criticism of Causal and Totally Ordered Communication," in *Operating System Review*, Jan. 1994, vol. 28, pp. 11–21.

[26] D. Cheriton and D. Skeen, "Comments on the Responses by Birman, van Renesse and Cooper," *Operating Systems Review*, p. 32, Jan. 1994.

[27] D. Velten, R. Hinden, and J. Sax, *Reliable Data Protocol*, Internet Engineering Task Force, July 1984, RFC 908.

[28] C. Partridge and R. M. Hinden, *Version 2 of the Reliable Data Protocol (RDP)*, Internet Engineering Task Force, Apr 1990, RFC 1151.

[29] David D. Clark, "The Design Philosophy of the DARPA Internet Protocols," in *Proceedings of SIGCOMM '88*, Stanford, CA, Aug. 1988, ACM.

[30] V. N. Padmanabhan and J. C. Mogul, "Improving HTTP Latency," in *Proc. Second International WWW Conference*, Oct. 1994.

[31] J. Gettys, "MUX protocol specification, WD-MUX-961023," http://www.w3.org/pub/WWW/Protocols/MUX/WD-mux-961023.html, 1996.

[32] "Message Multiplexing (memux) Charter," http://www.w3.org/Protocols/HTTP-NG/1999/02/mux-Charter-222.html, 1999.

[33] M. Podolsky, K. Yano, and S. McCanne, "RTP Profile for RTCP-based Retransmission Request for Unicast Sessions," Internet-Draft draft-ietf-avt-rtprx-00.txt, July 2000, Work in progress, expires January 2001.

[34] Rajarshi Gupta, Mike Chen, Steven McCanne, and Jean Walrand, "A Receiver-Driven Transport Protocol for the Web," in *Proc. INFORMS 2000 Telecommunications Conference*, March 2000.

[35] C. J. Turner and L. L. Peterson, "Image transfer: an end-to-end design," in *Proc. ACM SIGCOMM*, August 1992.

[36] R. Han and D. G. Messerschmitt, "Asymptotically Reliable Transport of Multimedia/Graphics Over Wireless Channels," in *Proc. SPIE Multimedia Computing and Networking*, Jan. 1996.

[37] John Danskin, Geoffrey Davis, and Xiyong Song, "Fast Lossy Internet Image Transmission," in *Proceedings of ACM Multimedia '95*. ACM, Nov. 1995.

[38] P. Salama, N. B. Shroff, and E. J. Delp, *Error Concealment in Encoded Video Streams*, Kluwer Academic Publishers, 1998, Book Chapter in "Signal Recovery Techniques for Image and Video Compression and Transmission", edited by N. P. Galatsanos and A. K. Katsaggelos.

[39] Nick Feamster and Hari Balakrishnan, "Adaptive Video Streaming," http://nms.lcs.mit.edu/projects/videocm/, 2001.

Suchitra Raman received a B.Tech. from the Indian Institute of Technology (Madras) in 1996 and a Ph.D. in Computer Science from the University of California at Berkeley in 2000. She is a Principal Design Engineer at Acopia Networks. Her research interests are in network protocols and performance, storage systems, and wide-area Internet systems. She received an award paper at ACM SIGCOMM in 1999 for a paper analyzing soft-state techniques in network protocols.

Hari Balakrishnan received a B.Tech. from the Indian Institute of Technology (Madras) in 1993 and a Ph.D. in Computer Science from the University of California at Berkeley in 1998. He is an Assistant Professor in the EECS Department at MIT, where he holds the KDD Career Development Chair. He leads the Networks and Mobile Systems group at the Lab for Computer Science (http://nms.lcs.mit.edu/), exploring research issues in network protocols and architecture, mobile computing systems, and pervasive computing. Balakrishnan is the recipient of a Sloan Foundation Fellowship (2002), a National Science Foundation CAREER Award (2000), the ACM doctoral dissertation award (1998), and award papers at the ACM MOBICOM (1995 and 2000), IEEE HotOS (2001), and USENIX (1995) technical conferences. He was awarded the MIT School of Engineering Ruth and Joel Spira Award for Distinguished Teaching in 2001. He is a member of ACM.

Murari Srinivasan received a B.Tech. from the Indian Institute of Technology (Madras) in 1993 and a Ph.D. in Electrical Engineering from the University of Maryland, College Park in 1999. He was a Post-Doctoral Fellow at the Division of Engineering and Applied Sciences at Harvard University between 1999 and 2000. He is currently a Member of Technical Staff at Flarion Technologies, a wireless infrastructure company whose technology enables end-to-end broadband IP connectivity over wide-area cellular networks. At Flarion, he is involved with the design and implementation of the link layer protocol and radio resource management algorithms. His research interests include digital communications, wireless networks and communications, radio resource management, network protocols, and joint source-channel coding.