Application Level Framing Applied to Image Compression

Compression d'images par ondelettes selon une approche de type ALF

Sami IREN

Marconi Plc.
1000 Marconi Drive
Pittsburgh, PA 15237
USA
sami.iren@marconi.com

Paul D. AMER

University of Delaware
Department of Computer and
Information Sciences
Newark, DE 19716-2586
USA
amer@cis.udel.edu

ABSTRACT. *Two well-known wavelet zerotree encoding algorithms, Embedded Zerotree Encoding (EZW) and Set Partitioning in Hierarchical Trees (SPIHT), provide excellent progressive display when images are transmitted over reliable networks. However, both algorithms are state-dependent and can perform poorly over unreliable networks. We apply the concept of network-conscious image compression to the SPIHT wavelet zerotree encoding algorithm, to improve its performance over unreliable networks. Experimental results confirm the utility of network-conscious image compression concept.*

RÉSUMÉ. *L'encodage par ondelettes avec quantification hiérarchique de type « zerotrees » constitue une façon efficace de compresser les images. Dans une perspective de transmission d'images, deux algorithmes d'encodage classiques, EZW (Embedded Zerotree Encoding) et SPIHT (Set Partitioning in Hierarchical Trees), permettent un affichage progressif de bonne qualité lorsque les images sont transmises sur un réseau fiable. Cependant, les deux algorithmes sont basés sur l'utilisation d'états et une erreur introduite par le réseau peut altérer sérieusement le résultat. Nous appliquons le concept de compression d'image consciente du réseau à l'algorithme SPIHT pour l'amélioration de ses performances dans des environnement réseau non fiables. Nos résultats confirment l'avantage de cette technique dans certaines situations de qualités de service.*

KEY WORDS: *ALF, network conscious, wavelets, zerotree encoding, image compression, transport protocols, unordered no-loss transport service.*

MOTS-CLÉS : *ALF, approche consciente du réseau, ondelettes, encodage zerotree, compression d'image, protocoles de transport, service de transport fiable sans garantie d'ordre*

An abbreviated version of this paper appeared in [IRE 00].

1. Introduction

Wavelet zerotree encoding is an algorithm that utilizes the correlation between coefficients of different scales to provide good image compression in the wavelet domain. Wavelet zerotree encoding is based on the hypothesis that, at a given threshold level, if a wavelet coefficient at a coarse scale is insignificant, then all wavelet coefficients of the same orientation in the same spatial location at finer scales are likely to be insignificant. The embedded zerotree (EZW) encoding, introduced in [SHA 93] has proven to be an efficient yet not complex encoding scheme. The embedded nature of EZW, a representation in which a high resolution image contains all coarser resolutions, effectively sorts bits in order of importance, thus permitting an effective progressive display when transmitted over low-bandwidth networks.

Set Partitioning in Hierarchical Trees (SPIHT), introduced in [SAI 96] as a refinement to EZW, differs from EZW in the way subsets of coefficients are partitioned and in the way significance information is conveyed. SPIHT is so effective that even binary uncoded transmission achieves equal or better performance than EZW [SAI 96].

Both EZW and SPIHT provide excellent progressive display. These algorithms produce a bit stream in which the bits corresponding to different trees are interleaved. When this bit stream is decoded, coefficients in all trees are restored in parallel, thus yielding a progressive display of the entire image. These algorithms encode the most significant bit of every coefficient of every tree before encoding the next significant bit. They are designed to provide the maximum PSNR for a given significance level. Assuming PSNR is our measure of image quality, zerotree encoding provides the maximum achievable image quality at each significance level allowing the image to progress as more bits are decoded.

EZW and SPIHT are clearly excellent *progressive display* wavelet zerotree encoding algorithms available today for *reliable* networks. For unreliable networks, however, EZW and SPIHT have a major drawback; both are highly state-dependent and therefore susceptible to bit errors. Even a single changed, missing, or extra bit ruins the decoding process often destroying an entire image. Recent studies concentrate on composing noise-robust zerotree encoders. Most of these studies are based on the idea of dividing the bitstream into several sub-streams each of which receive different amounts of error protection based on their noise sensitivity [MAN 97], or interleaving separately encoded substreams such that any single bit error will corrupt only one substream [CRE 96A, CRE96B].

Recently, Rogers and Cosman [ROG 98] introduced an innovative packetized zerotree encoding (PZW) method on still images that produces fixed 53-byte ATM-compatible packets and is robust against packet loss. A similar study by Crump and Fischer [CRU 97] produced variable-length independent packets for video transmission. Sherwood and Zeger [SHE 99] improved the PZW algorithm using a technique called "Macroscopic Multistage Image Compression".

Unlike these previous studies which focus on robustness of the algorithm, our research on wavelet zerotree encoding concentrates on the progressive display aspect as well as robustness, when images are transmitted over low-speed, lossy, packet-switched networks (e.g., battlefield networks). Our study of wavelet zerotree encoding is part of a broader research effort in *network-conscious image compression* [IRE 99].

A network-conscious compressed image is one that is encoded *not* simply to give the smallest size for a specified image quality, but to give the *best (i.e., smallest) response time - image quality* combination to an end user retrieving the image over a packet-switched network [IRE 98B]. Network-conscious image compression is based on the networking concept of Application Level Framing [CLA 90]. An image is divided by the application into path-MTU-size [1] pieces, called *Application Data Units (ADUs)*, so that each piece carries its semantics, that is, each ADU contains enough information to be processed independently of all other ADUs. As a result, each ADU can be delivered to the decoding application immediately upon arrival at the receiver without regard to order, thereby potentially enabling faster progressive image display.

We present a network-conscious version of the SPIHT algorithm called SPIHT-NC. SPIHT-NC changes the structure of the encoded bit stream to produce independent ADUs. Obviously any modification to the original structure of the SPIHT algorithm may diminish progressive display performance when there is no loss as explained in Section 3. Our objective is to quantify the gain in making the SPIHT algorithm network-conscious for faster progressive display over lossy networks. Previous work has shown the advantages of network-conscious image compression for other image compression techniques [AME 99, IRE 99, IRE98A, IRE98B].

Section 2 describes our development of a network-conscious SPIHT algorithm. Section 3 presents experimental results, and Section 4 concludes the paper with a summary of the experimental results.

2. SPIHT-NC: Network-Conscious Wavelet Zerotree Encoding

SPIHT is one of the best progressive display wavelet zerotree encoding algorithms available today. Over unreliable networks, however, SPIHT suffers from the same phenomenon every other non-network-conscious progressive compression algorithm suffers; it delays presenting out-of-order data to the user until missing pieces arrive. In this section we present SPIHT-NC, a modified version of SPIHT that performs better over unreliable networks.

---

[1] MTU (Maximum Transmission Unit) is the maximum frame size that a link layer can carry. A path-MTU-size ADU is one that can be transmitted end-to-end over a network without the need for IP layer fragmentation and reassembly.

To be able to create independent, SPIHT-encoded ADUs, one has to prohibit state dependency between ADU boundaries. Since the smallest independent unit in zerotree encoding algorithms is a tree, we first encoded each tree separately rather than interleaving all trees together. The most primitive version of SPIHT-NC packetized these independent trees to give independent ADUs. Roger and Cosman's PZW algorithm [ROG 98] originated this approach to introduce robustness against packet loss. This technique, called *uninterleaved spiht*, however, lacks an important feature of network-conscious image compression: progressive display. Although some basic techniques can be applied to give the illusion of progressive display (such as making sure that trees in each packet come from widely dispersed locations in the image, and interpolating the missing coefficients from the available coefficients), our experience shows that the resulting progressive display is unsatisfactory.

Among other methods we tried for improving the progressive display of SPIHT-NC, multistage image compression proposed by Sherwood and Zeger [SHE 99] gave the best results. This multistage technique consists of three phases as shown in Figure 1. Each phase encodes the residual wavelet coefficients from the previous phase. Phase 0 is intended to provide a rough image with a PSNR close to that of SPIHT at the same bit rate. Phase 1 is optimized to refine the rough image to give an image quality that is recognizable and can be used for low bit rate applications. Phases 0 and 1 together are intended to overcome the problem of slow increase in PSNR in the early moments of the file/transmission that we observed in our earlier efforts to design SPIHT-NC. Phase 2 further refines the image obtained from phases 0 and 1 to give a high quality image.

As seen in Figure 1, each phase of the encoding process inputs several parameters. These parameters are used to optimize a phase for a particular purpose. For example, for phase 0, the *levels* parameter is set to its largest possible value [2] so that an efficient zerotree coding of the whole image can be performed. In phase 2, a smaller number of wavelet decomposition levels is used to create smaller and more spatially diverse trees. The parameters (i.e., ADU size, levels, and number of bits to be encoded) dictate the number of ADUs that will be produced from a particular phase. For the test image we used in our experiments (see Section 3.2) an ADU size of 512 bytes was chosen. With this particular ADU size, phase 0 produced only one ADU, and phase 1 produced four ADUs. All remaining ADUs were produced in phase 2.

[2]The maximum number of wavelet decompositions depends on the image's dimensions. In our implementation it was set to $log_2(dimension)-2$.

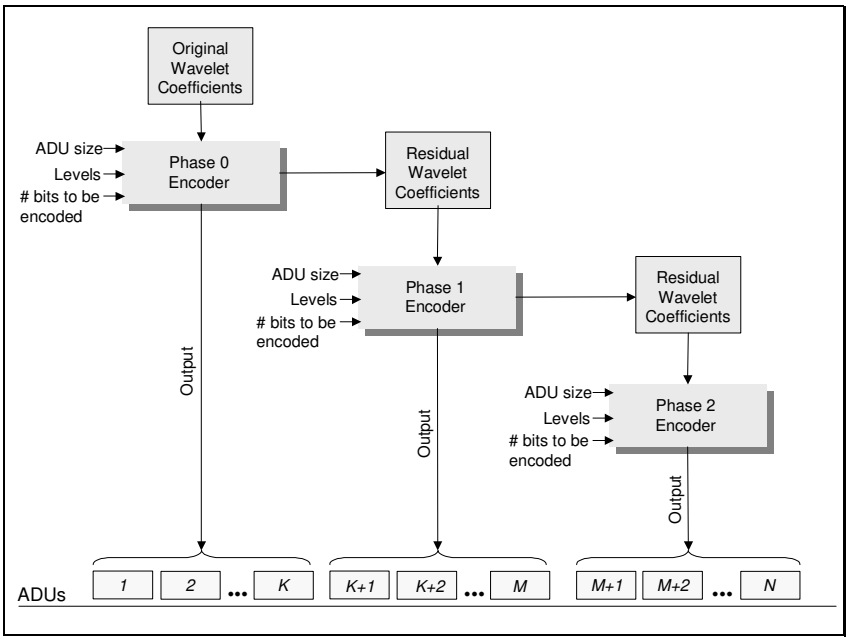

Figure 1. *Phases of SPIHT-NC Encoding Process*
Figure 1 *Les phases du processus d'encodage SPIHT-NC*

Typically, as the phase number increases, the wavelet decomposition level decreases. Using fewer levels of wavelet decomposition in later phases is desirable because the coefficients in the coarser scales are refined in earlier phases beyond their most significant bit and most of the parent-child dependencies at these scales are already exploited [SHE 99].

Each phase uses the uninterleaved SPIHT algorithm to pre-calculate the sizes of trees. Within each ADU, the encodings of a certain number of trees are interleaved. The starting tree number and the number of trees encoded are specified in each ADU's descriptor so that the receiver knows exactly what trees are available in the ADU. Depending on the tree sizes, an ADU might contain one or more trees. If trees were encoded in an uninterleaved fashion, one would have to specify the size of each tree, so that the receiver would know where one tree ends and another one starts. Interleaving the trees in each ADU eliminates this overhead.

The trees can be ordered for transmission in several ways. The simplest approach is to encode the trees in a raster scan order. Another method is to select the trees from spatially diverse locations so that any missing trees can be interpolated from the ones that are available. A third method is to encode the trees that improve the visual

quality the most in early ADUs. In our experiments we used the simplest method. We encoded trees at each phase in a raster scan order.

Figure 2 shows the file structures of SPIHT and SPIHT-NC. Both files start with a signature [3]. In SPIHT, an image descriptor follows that provides information about the image and some encoding parameters. The rest of the file contains the encoded data which is highly state dependent. In SPIHT-NC, after the signature, we have a sequence of self-contained ADUs. Each ADU has an image descriptor which carries image related parameters (such as the image dimensions), an ADU descriptor which carries information specific to that particular ADU (such as phase number, starting tree number, number of trees in the ADU, etc.), and encoded data.

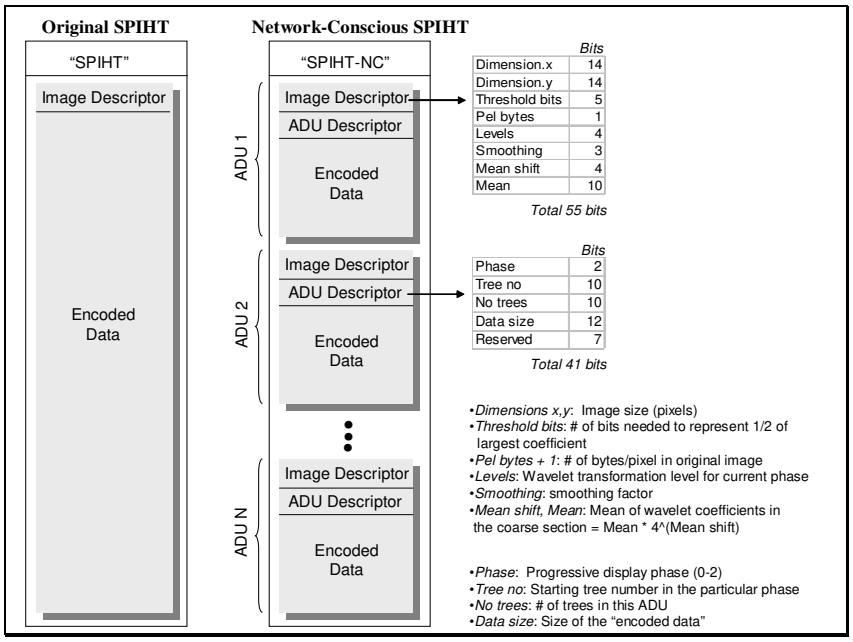

Figure 2. *File Structures of SPIHT and SPIHT-NC*
Figure 2. *Structures des fichiers SPIHT et SPIHT-NC*

[3]Said and Pearlman's original SPIHT code uses a one-byte signature (0x6E). For clarity we will use "SPIHT" and "SPIHT-NC", respectively.

3. SPIHT-NC vs. SPIHT Empirical Performance Evaluation

In this section we present a comparison of SPIHT and SPIHT-NC in terms of absolute compression and progressive display over lossy packet-switched networks.

When measuring the image quality, there are two types of image quality assessment tools we can use. The first type, the raw error measure, gives simple mathematical deviations between the original and the reconstructed image. The Mean Squared Error (MSE) and Peak Signal to Noise Ratio (PSNR) are two most commonly used examples of this type. The second type is based on human perception. It gives a numeric value that will quantify a viewer's satisfaction (or dissatisfaction) in observing the reconstructed image vs the orignal image. For our experiments, we picked the first type (particularly PSNR), because (1) they are easy to calculate and have low computational complexity and (2) they are independent of viewing conditions and individual observers.

3.1 Disadvantages of SPIHT-NC

The original SPIHT algorithm uses embedded coding. Therefore SPIHT allows an encoder to terminate the encoding at any point allowing a target rate or target distortion metric to be met exactly. Similarly a decoder can stop decoding at any point in a received bit stream and produce exactly the same image that would have been encoded at the bit rate corresponding to the truncated bit stream. With SPIHT-NC we had to sacrifice the valuable property of embedded coding to be able to produce independent ADUs. A SPIHT-NC encoder must know the target bit rate ahead of time.

A second disadvantage is that SPIHT-NC's compression performance is slightly worse than SPIHT. The obvious penalty comes from the extra header information that SPIHT-NC has to include. For each ADU, the size of additional header information is 12 bytes. Another overhead comes from the fact that we use a three phase process and each phase has its own compression overhead. For example, at each phase if a coefficient is found significant, the sign of the coefficient has to be encoded. However, since the coefficient is residual of a previous phase, it might well be the case that the sign of the coefficient might also have been encoded in the previous phase (if the coefficient is found significant in the previous phase). This multiple encoding of the same sign bit results in lower compression ratio.

Table 1 shows a typical example of SPIHT-NC's compression penalty with the Lena image. As the PSNR (and therefore the file size) increases, the compression penalty decreases. While at a PSNR of 33.05dB the compression penalty is 15.7%, the penalty is only 5.5% at a PSNR of 46.16dB. The overhead of having three phases remains fixed as the PSNR changes. Therefore this overhead is more obvious with a smaller file size.

PSNR	File Size (Bytes)		Compression Penalty (%)
	SPIHT	*SPIHT-NC*	
33.05	6126	7091	15.7
36.75	12266	13507	10.1
41.06	23697	25339	6.9
46.16	40697	42973	5.5

Table 1. *SPIHT-NC's Compression Penalty with Lena Image*
Table 1. *Pénalité de compression de SPIHT-NC pour l'image de Lena*

3.2 SPIHT-NC's Progressive Display Advantage

To illustrate the progressive display advantage of SPIHT-NC over SPIHT when images are transmitted over lossy, packet-switched networks, we ran a set of experiments comparing SPIHT-NC over a reliable, unordered transport protocol vs. SPIHT over a reliable, ordered transport protocol. Our aim is to investigate SPIHT-NC's performance vs. SPIHT under various network loss rates and bandwidths, and transport window sizes.

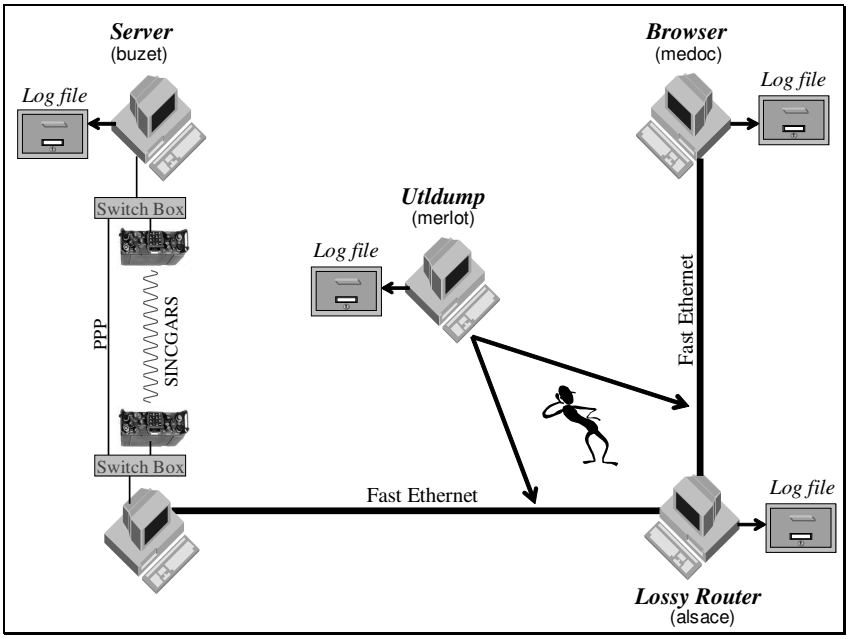

Figure 3. *NETCICATS : Experimental Setup*
Figure 3. *NETCICATS : Environnement expérimental*

The architecture for the experiments is shown in Figure 3. Each experiment downloads a compressed image from a server (on `buzet`) to a client (on `medoc`) using an interface similar to familiar web browsers. Packets are routed through a lossy router [4](on `alsace`) and either a PPP link or a SINCGARS radio link. A *utldump* [5]process (on `merlot`) observes the traffic on the network, and helps verify the correctness of our experimental operation. A log file for each process (i.e., server, browser, utldump, and lossy router) is created to later verify and analyze the results.

3.2.1 Transport Protocol

Turning to the measurement of progressive display, one faces a dilemma. If one uses the Internet's most advanced transport protocol available, namely TCP, one is forced to accept TCP's ordered, reliable delivery. But when used over ordered service, SPIHT-NC cannot possibly offer an advantage over SPIHT. On the other hand, there is no commonly accepted *un*ordered reliable protocol that features TCP-compatible congestion avoidance. As such, a direct comparison of an experimental unordered reliable protocol (without congestion control) vs. TCP (with its congestion control) would unfairly favor SPIHT-NC. Our approach is to compare SPIHT vs. SPIHT-NC over experimental transport protocols that differ only in that one is *ordered*, and the other is *unordered*. To cover cases with and without TCP-compatible congestion control, we used two experimental transport protocols: X2E and S2E developed within the Protocol Engineering Laboratory at the University of Delaware. Details of these protocols are available in [CON 00]. For our purposes, it suffices to say that X2E is an unordered, no-loss, maybe-duplicates transport protocol with window-based flow control and no slow start. S2E is the same as X2E except that it provides ordered, no-duplicates service. To compare SPIHT vs. SPIHT-NC in an environment with flow control but no slow start (no congestion control), we transmit SPIHT over S2E, and SPIHT-NC over X2E. Since we would like to observe the effects of various transport and network quality of service parameters on SPIHT vs. SPIHT-NC, we ran hundreds of experiments investigating loss rate, sending transport window size, bandwidth, and transmission medium (e.g., PPP, SINCGARS, Internet). Details and results of these experiments can be found in [IRE 99]. Because of space limitations we focus our results on experiments testing SPIHT-NC vs. SPIHT performance at various loss rates and transport window sizes.

---

[4]Lossy Router is an IP gateway that randomly drops certain packets according to a specified loss model and loss rate.

[5]*utldump*is a modified version of the *tcpdump* program written by Jacobson, Leres, and McCanne [STE 94]. tcpdump puts the network interface card into promiscuous mode so it captures every packet going across the wire. Then by applying a user-specified filter (e.g., host names, port number, etc.) packets of interest to the user are displayed. utldump performs the same job for UTL (Universal Transport Library) [CON 98] packets.

3.2.2 Effects of Loss Rate

This experiment involves downloading SPIHT-NC and SPIHT versions of a space shuttle image over a 9.6Kbps PPP link. Flow control between transport sender and receiver is performed via a sending window size of 16 packets. The lossy router simulates 0%, 5%, 10%, 20%, and 30% one-way IP packet loss. Graphs presented in this paper represent averages of at least 30 runs for each combination of the tested parameters. For example, Figure 4 contains graphs showing SPIHT-NC (illustrated in gray) vs. SPIHT (illustrated in black) at 0%, 10%, and 20% one-way IP packet loss rates. Graphs on the left show average number of bytes delivered to the application as time progresses. Graphs on the right show average PSNR values for the images that are displayed during the same time interval.

In these experiments, the relation between bytes and PSNR is non-linear. Since SPIHT's wavelet transformation provides layering of information, the effect of x data bytes from one layer on PSNR will not be the same as the effect of x data bytes from another layer x bytes from an upper layer (coarse scale) will result in a larger PSNR increase than x bytes from a lower layer (finer scale).

In the wavelet domain, upper layers contain significant coefficients (i.e., larger in absolute value). With SPIHT bits from these significant coefficients get transmitted first. Therefore, a sharp increase in PSNR occurs in the early moments of the transmission, and a steady increase occurs in the later part of the transmission. This sharp increase is most visible with the arrival of the first packet.

With 0% network packet loss, the arrival of the first SPIHT packet results in a PSNR jump from 0 to 23.27dB. Similarly, with the arrival of the first SPIHT-NC packet, the PSNR jumps from 0 to 23.24dB. The small PSNR difference comes from the fact that SPIHT-NC has a header 5-bytes larger than SPIHT's first packet. Therefore, the first SPIHT-NC packet (ADU) carries 5-bytes less *information*. Packets 2--5 yield more modest increases in PSNR (24.96dB, 26.06dB, 26.71dB, 27.34dB with SPIHT, and 23.97dB, 24.73dB, 25.85dB, 27.15dB with SPIHT-NC). Because of the large jump in PSNR with the first packet, it is hard to analyze the early moments of the graph. If the first packet is lost, the receiver has no image, i.e., a PSNR of 0dB. If the first packet is delivered to the application (i.e., not lost), the receiver sees an image with a PSNR of more than 23dB. Considering the graphs in Figure 4 represent averages of multiple runs, the early moments of the graph will have the averages of 0's (when the first packet is lost) and 23's (when the first packet is not lost). The average of these numbers will be between 0dB and 23dB depending on the loss rate.

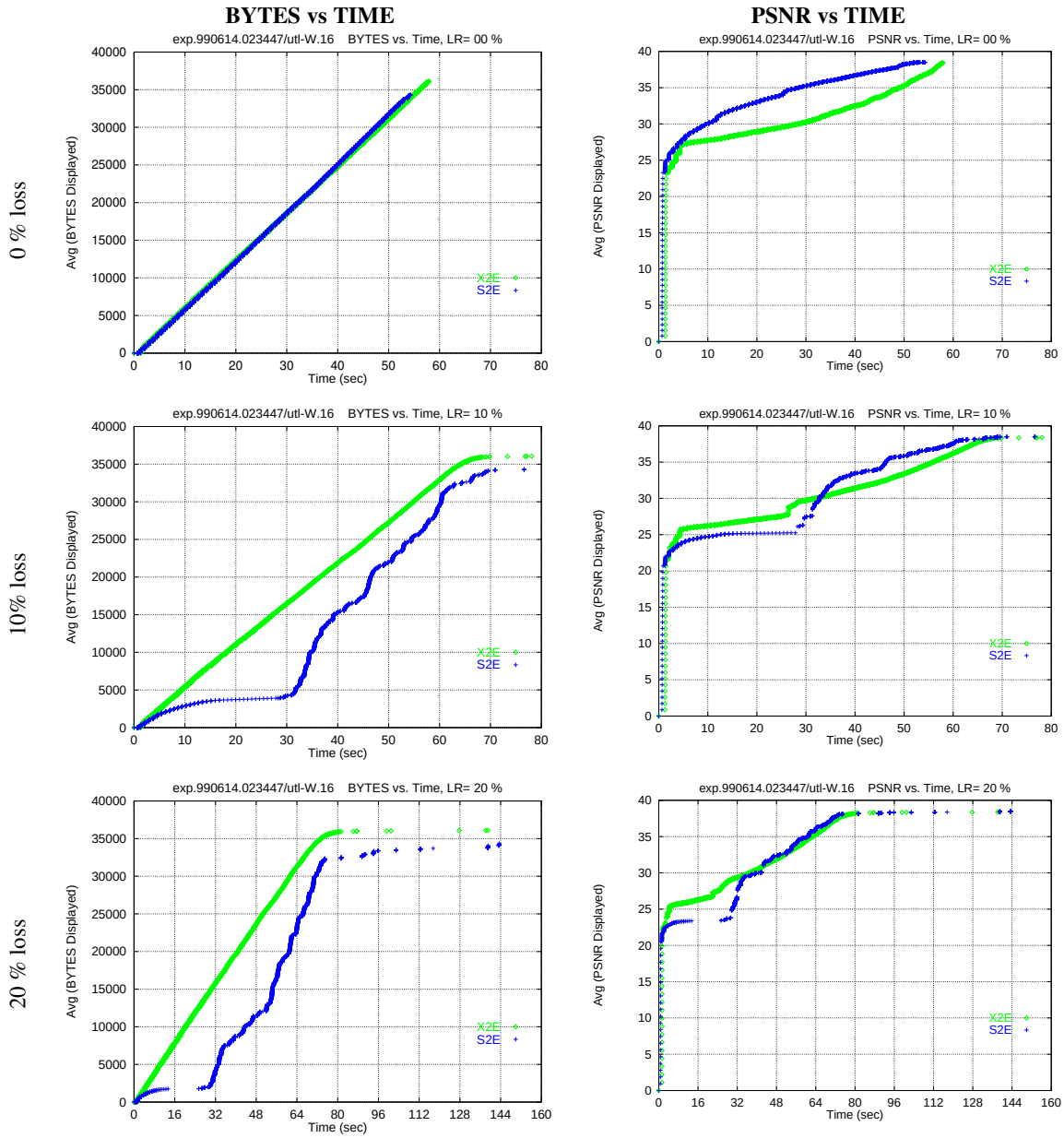

Figure 4. *SPIHT-NC/X2E vs SPIHT/S2E at 0%, 10%, and 20% Loss*
Figure 4. *SPIHT-NC/X2E vs SPIHT/S2E - taux de perte de 0 %, 10 %, 20 %*

Consider the graphs when the loss rate is 20%. Of 100 runs we expect 20 runs with 0dB and 80 runs with 23dB. The average of these numbers is roughly 18dB. In real life, however, the client never sees an image with 18dB with this particular implementation as seen in the graphs for both SPIHT and SPIHT-NC. Therefore, instead of producing an image that shows the average case, in Figure 5 we present the two possible real life cases. With a 20% loss rate, on the average 20% of the time the client will initially see an (blank) image with a PSNR of 0dB, and 80% of the time we will see an image with a PSNR of roughly 23dB.

The above statements apply to both SPIHT-NC and SPIHT. The interesting point is what happens after the first packet. With SPIHT, if the first packet is lost, any subsequent packets will be buffered at the transport receiver. No image will appear until the missing first packet is retransmitted and successfully received. With SPIHT-NC, on the other hand, even when the first packet is lost, packets that follow will be delivered, decoded, and displayed. Since the first packet contains the coarse scale coefficients, the quality of the image produced with the subsequent packets alone will be limited. Nevertheless, with SPIHT-NC, at least some image will be displayed on the screen during the time that the missing first packet is retransmitted and received. In real-time applications over low-bandwidth networks, this time can be significant (i.e., life or death).

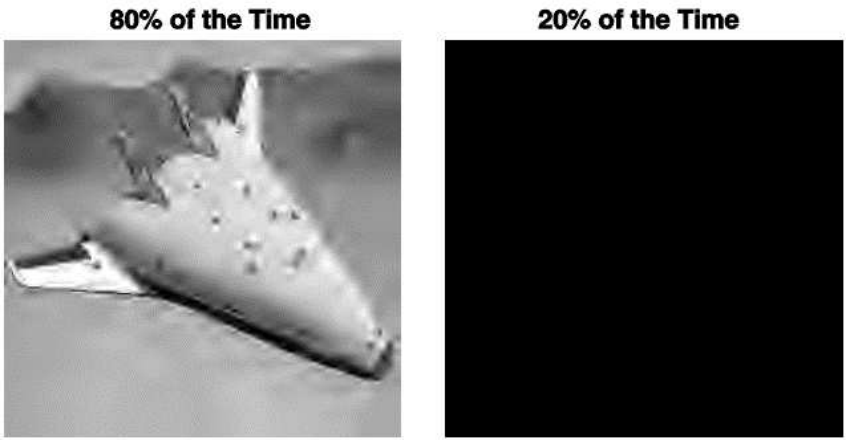

Figure 5. *Reconstructed image with or without first packet at 20% loss (independent of algorithm)*
Figure 5. *Image reconstruite sans premier paquet vs avec le premier paquet (indépendamment de l'algorithme)*

Figure 6 illustrates the image's quality when only packet 1 of the first five packets is lost. Even without packet 1's coarse scale coefficients, the coefficients in the next

four packets present an image which is recognizable. In a time-critical application, SPIHT-NC will provide some information sooner than SPIHT.

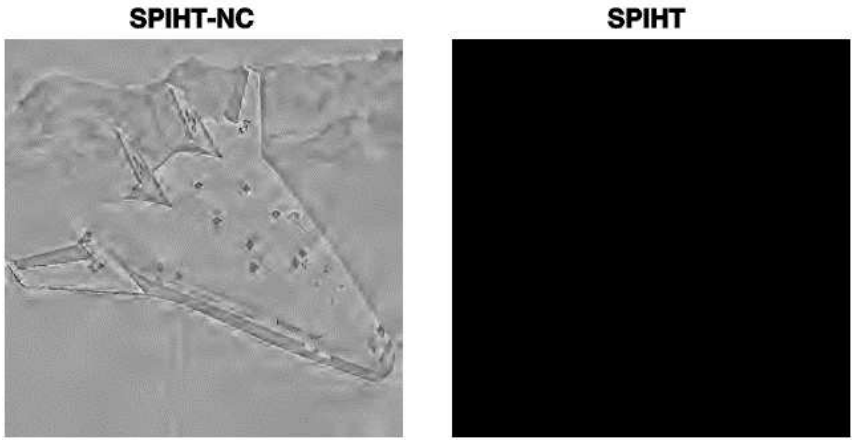

Figure 6. *Reconstructed Image after 5 Packets When the First Packet is Lost*
Figure 6. *Image reconstruite après 5 paquets lorsque le premier est perdu*

Figure 4's graphs on the lower left show that unordered service delivers more data than ordered service when there is loss. At 0% loss, SPIHT provides a better progressive display than SPIHT-NC from start to finish. Considering the excellent progressive display capability of SPIHT, this result is not surprising. One observation to note is that SPIHT-NC performs closely to SPIHT in the early moments of the transmission. Similar performance occurs because the first two phases of SPIHT-NC were optimized to perform better (i.e., close to SPIHT) at low bit rates. The performance difference is larger in the later parts of the transmission where ADUs of the third phase are being transmitted.

Starting with 10% loss, SPIHT-NC begins to show better progressive display than SPIHT early in the transmission. At 10% loss, SPIHT-NC performs better in the first 33 seconds. The largest gain for SPIHT-NC occurs around 28 seconds where SPIHT-NC provides a PSNR of 28dB while SPIHT provides a PSNR of only 25dB. As the loss rate increases so does the gain of SPIHT-NC over SPIHT. At 20% loss, SPIHT-NC performs better in the first 40 seconds. The largest gain for SPIHT-NC occurs around 30 seconds where SPIHT-NC provides a PSNR of 28dB while SPIHT provides a PSNR of 23dB. To illustrate this performance gain visually, Figure 7 provides both SPIHT-NC and SPIHT images at 30 seconds under 20% packet loss. The SPIHT-NC image shows a sharper shuttle and focused mountains in the background. With SPIHT, the background is ambiguous.

Figure 8 presents the same results as Figure 4, but with the graphs organized by algorithm rather than by loss rate. The graph on the right illustrates how increasing the loss rate significantly affects the progressive display performance of SPIHT. During a transmission, if the loss rate suddenly changes from 0% to 10%, the progressive display performance will degrade by a large amount.

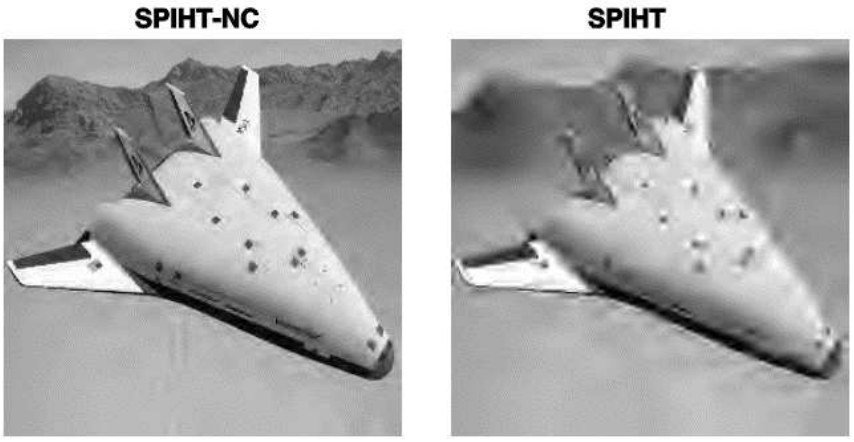

Figure 7. *SPIHT-NC vs SPIHT at 20% Loss*
Figure 7. *SPIHT-NC vs SPIHT à 20 % de perte*

With SPIHT-NC (the graph on the left), however, as the loss rate increases, progressive display performance degrades gracefully and by smaller amounts. As the loss rate increases, the network throughput decreases. Therefore, as the loss rate increases, we see a shift in the graphs towards the right. As expected, it takes more time to transmit the same image at higher loss rates than lower loss rates. The shifts on the SPIHT graph occur early on resulting in poorer average progressive display. The shifts on the SPIHT-NC graph are more noticeable towards the end.

Figure 9 shows the standard deviations for SPIHT-NC and SPIHT at various loss rates. Each point in these graphs represents the average PSNR value from all runs with an error bar representing (+/-) one standard deviation. In both SPIHT-NC and SPIHT graphs, the variance in early moments of the transmission is larger than at the end. This difference is due to the fact that the loss or delivery of the first packet makes a huge impact on the resulting PSNR. Once packet 1 arrives, the PSNR variance diminishes. Overall, SPIHT's variance is larger than SPIHT-NC's. This shows the graceful degradation advantage of SPIHT-NC over SPIHT.

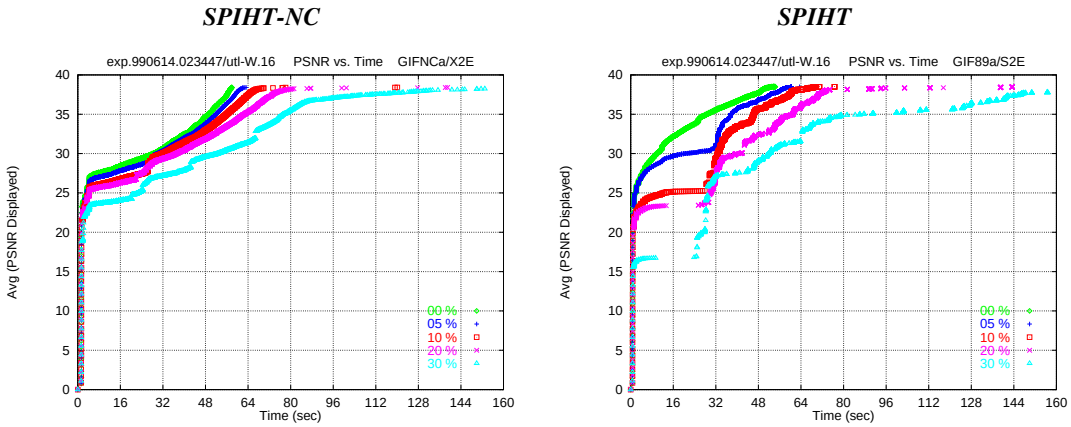

Figure 8. *Performance of SPIHT-NC and SPIHT at 0%, 5%, 10%, 20%, and 30% Loss*
Figure 8. *Performances de SPIHT-NC et SPIHT à 0, 5, 10, 20 et 30 % de perte*

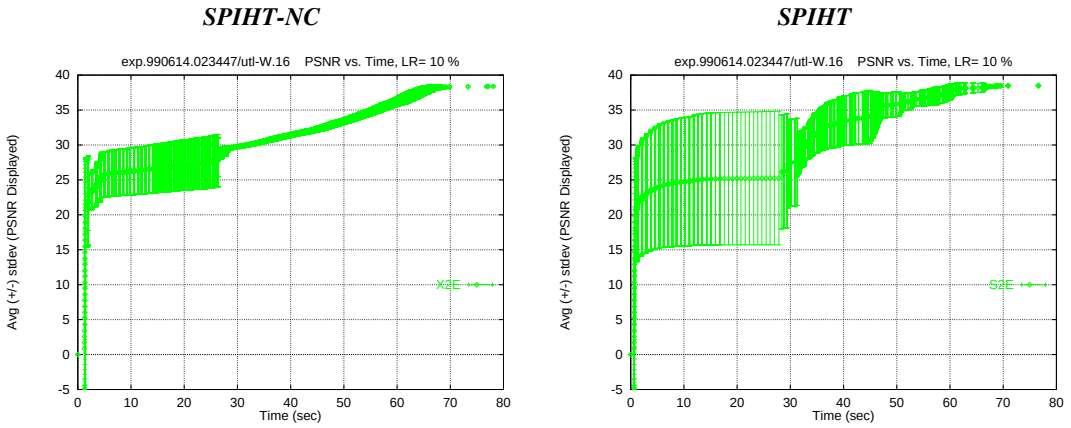

Figure 9. *SPIHT-NC and SPIHT with Error Bars at 10% Loss*
Figure 9. *Performance de SPIHT-NC et SPIHT avec des intervalles d'erreur (à 10 % de perte)*

3.2.3 Effects of Transport Window Size

In this experiment we evaluate the effect of sending transport window size on the progressive display advantage of SPIHT-NC over SPIHT. Studying window size is

important because most often applications do not have control over what window size is used at the transport or link layers. Hence it is desirable to have applications perform robustly without regard to the window size.

The SPIHT-NC and SPIHT versions of the space shuttle image are downloaded over a 9.6Kbps PPP link. The one way IP packet loss rate is fixed at 10%.

Figure 10 compares the progressive display of SPIHT-NC vs. SPIHT at window sizes of 8, 16, and 32 packets. Graphs on the left (bytes vs. time) show that SPIHT-NC over reliable unordered transport service (illustrated in grey) always delivers more bytes than SPIHT over reliable ordered transport service (illustrated in black) regardless of the transport sender's window size.

In terms of PSNR, at window size 8, SPIHT-NC does not perform better than SPIHT. Initially, the two approaches perform similarly, and then SPIHT starts performing better roughly after 17 seconds. The 17 second point represents when SPIHT-NC enters its third phase.

SPIHT-NC shows periods of definite improvement over SPIHT for window sizes 16 and 32. At size 16, SPIHT-NC performs better during the first 33 seconds, on average providing 3 to 5 more dB. At size 32, SPIHT-NC always outperforms SPIHT. SPIHT-NC's peak performance gain occurs around 54 seconds, when SPIHT-NC provides an image with a PSNR of 34dB while SPIHT provides an image with a PSNR of only 25dB. Figure 11 illustrates these two images. Again, notice the more focused background and shuttle with SPIHT-NC

Figure 12 displays performance according to algorithms rather than window size. From Figure 12, we see that as the window size increases, the progressive display performance of SPIHT decreases both in terms of bytes delivered and more importantly in terms of PSNR. This result is counterintuitive. Why does performance degrades as more buffer resources are provided? This degradation comes from the fact that window size is directly proportional to the queue length at PPP buffers, and larger window size results in larger queuing delay at PPP buffers. As mentioned previously, the loss or delivery of the first packet dominates the PSNR value. If the first packet is lost, its retransmission time is dictated by the queuing delay at PPP buffers. The larger the window size (hence larger the queuing delay) the more time it takes for the first packet's retransmission to arrive at the receiver. Therefore, as the window size increases, the performance of SPIHT decreases.

The bottom two graphs show how SPIHT-NC's performance is essentially unaffected by window size. The PSNR only slightly decreases as the window size increases.

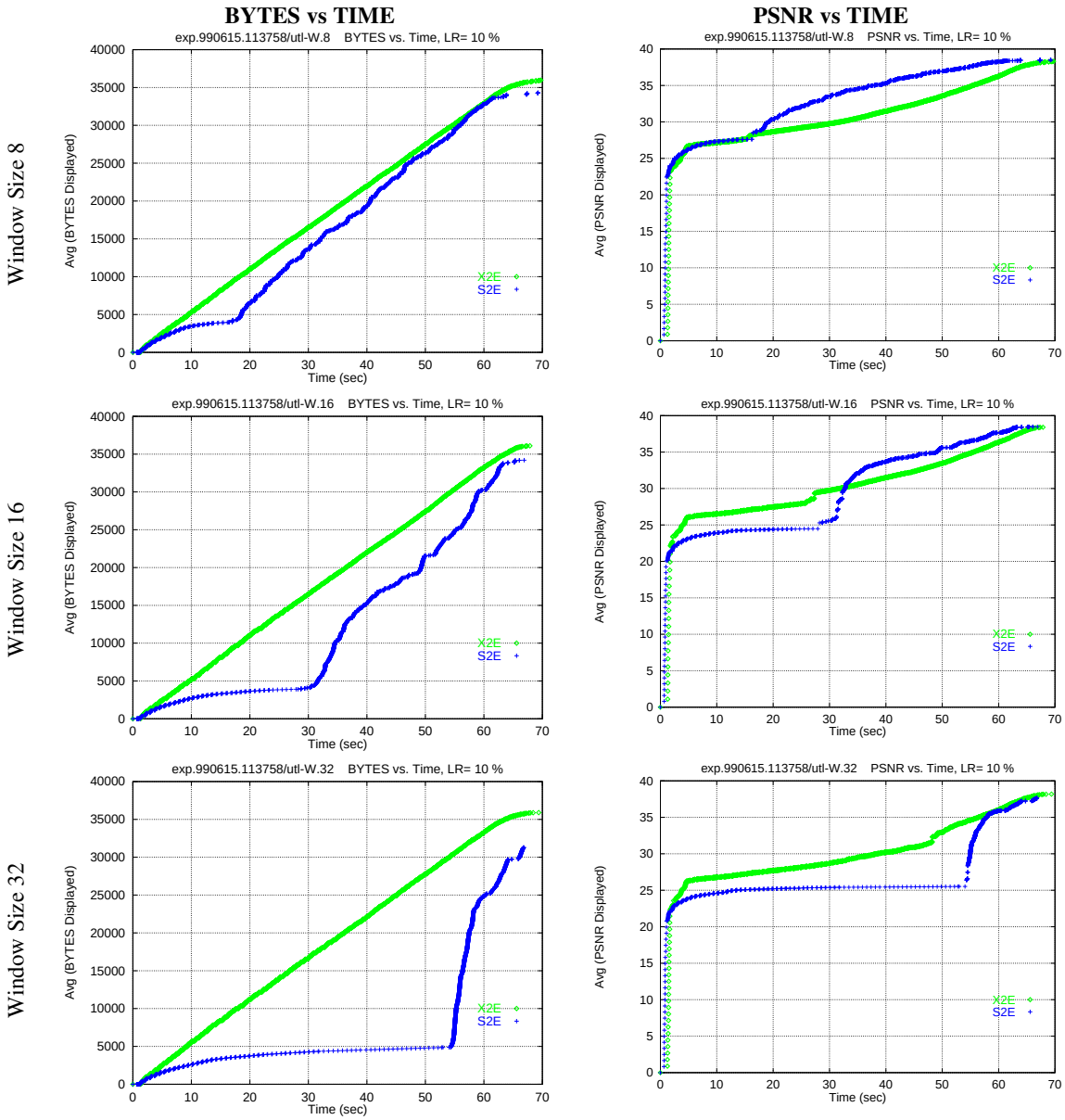

Figure 10. *SPIHT-NC/X2E vs SPIHT/S2E at Sending Window Sizes 8, 16, and 32*
Figure 10. *SPIHT-NC/X2E vs SPIHT/S2E - tailles de fenêtre de 8, 16, et 32*

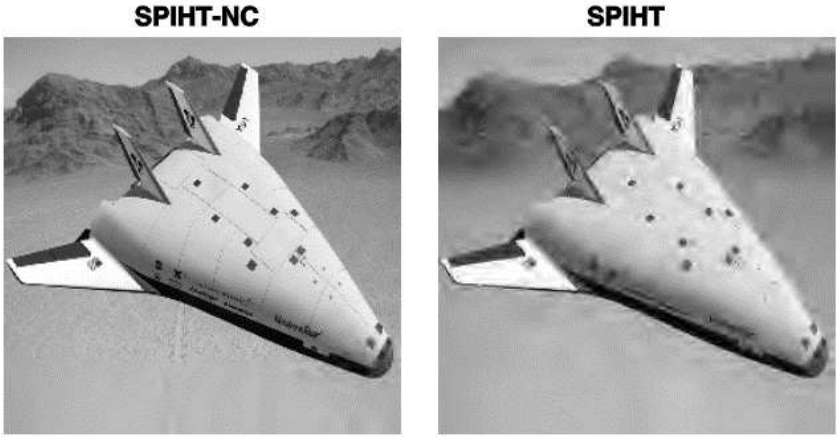

Figure 11. *SPIHT-NC (34dB) vs SPIHT (25dB) at 10% Loss, Window Size 32*
Figure 11. *SPIHT-NC (34dB) vs SPIHT (25dB) - à 10 % de perte, taille de fenêtre de 32*

It is important to observe the significant difference in how much window size affects the two algorithms. SPIHT-NC is virtually unaffected, while SPIHT is most certainly affected. These results indicate SPIHT-NC as being more robust to lower layer windowing issues. Given applications rarely have control over link and transport layer windowing sizes, it is desirable to have a robust approach such as SPIHT-NC.

4. Summary and Conclusions

SPIHT is an embedded zerotree encoding algorithm that provides excellent progressive display and performs well at low bit rates. However, it is highly state dependent and susceptible to bit errors. SPIHT's performance over packet switched networks degrades quickly when the packet loss rate increases.

SPIHT-NC, a network-conscious version of SPIHT, is designed to improve performance under lossy conditions. SPIHT-NC uses an uninterleaved version of SPIHT encoding to produce ADUs that are independent of each other. A multiphase (or multistage) encoding provides progressive display. Since ADUs are independent, a packet lost within the network affects only a single ADU, not the entire image. This feature makes SPIHT-NC more robust.

The gain of SPIHT-NC--- and we extrapolate--- of any network-conscious compression technique occurs when images need to be progressively displayed at the receiver as soon as possible. In military applications, seconds may be a matter of life and death. In less critical, yet still 'timely' applications such as browsing the Web, faster display improves user perception and acceptance. Faster display is certainly appealing to Web advertisers who want their logos to appear on the screen before the user moves on to another page.

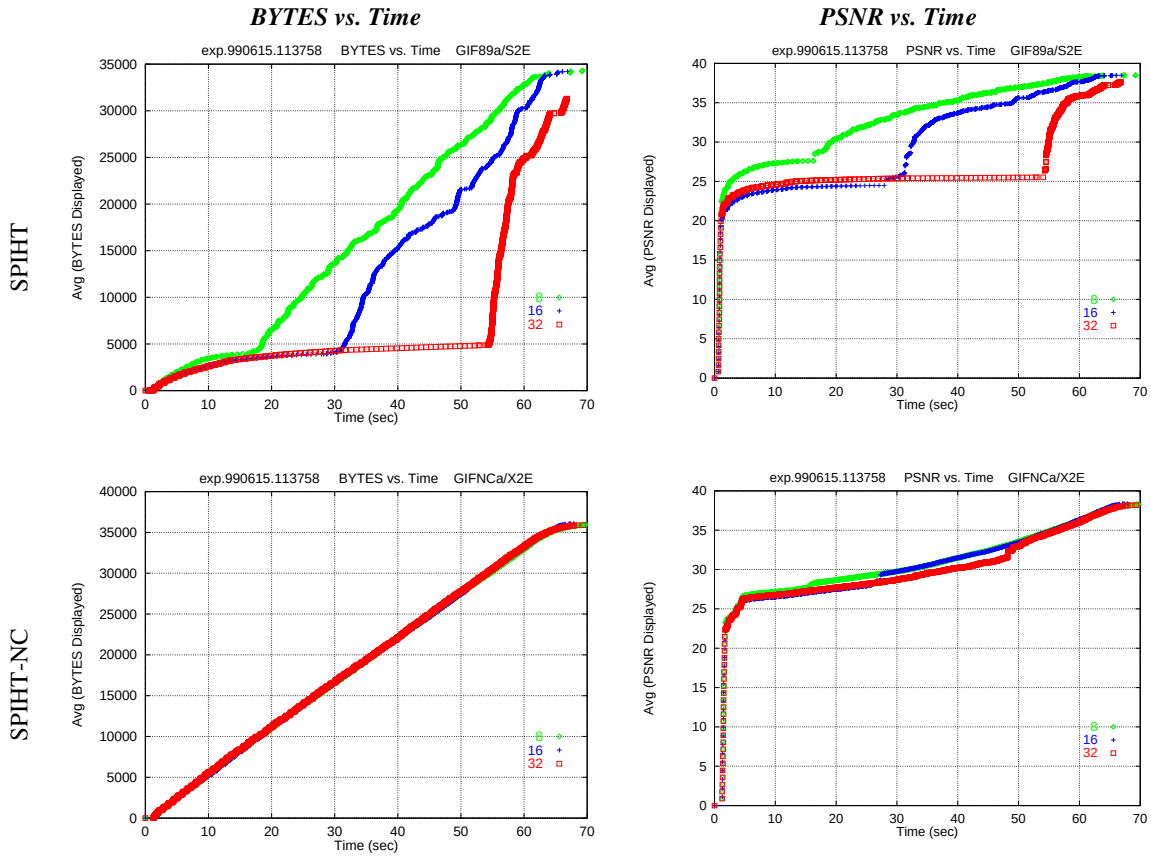

Figure 12. *Performance of SPIHT and SPIHT-NC at Window Sizes 8, 16, and 32*
Figure 12. *Performance de SPIHT et SPIHT-NC – tailles de fenêtre de 8, 16, 32*

A primary motivation of this research is to argue that future image compression standards take into consideration whether or not the images are likely to be transmitted over the Internet (or other lossy, packet-switched networks), and displayed in either real-time or interactive environments where progressive display efficiency is a major consideration. Network-conscious image compression focuses not simply on maximizing compression; it focuses on optimizing overall progressive display performance. Experimental results show that, starting at a 10% packet loss rate, SPIHT-NC outperforms SPIHT in the early moments of the progressive display where better progressive display is desirable. As the loss rate increases, the performance gain of SPIHT-NC over SPIHT increases due to the fact that SPIHT-NC degrades gracefully under lossy conditions. As the time it takes for a

retransmission to arrive at the transport receiver increases, so does the performance gain of a network-conscious approach over a traditional approach.

Acknowledment

This research is supported, in part, by the ATIRP Consortium sponsored by the US Army Research Lab under the Federal Laboratory Program (DAAL01-96-2-0002).

References

[AME 99] AMER P., IREN S., SEZEN G., CONRAD P., TAUBE M., Caro A., "Network-conscious GIF image transmission over the Internet", *Computer Networks*, vol. 31, n° 7, April 1999, p. 693-708.

[CLA 90] CLARK D., TENNENHOUSE D., "Architectural considerations for a new generation of protocols", *ACM SIGCOMM Conference*, Philadelphia, Pennsylvania, USA, September 1990, p. 200-208.

[CON 98] CONRAD P., AMER P., TAUBE M., SEZEN G., IREN S., CARO A., "Testing environment for innovative transport protocols", *MILCOM Conference*, Bedford, MA, USA, October 1998.

[CON 00] CONRAD P., "Order, reliability, and synchronization in transport layer protocols for multimedia document retrieval", *PhD Dissertation*, CIS Department, University of Delaware, USA, 2000.

[CRE 96A] CREUSERE C., "A family of image compression algorithms which are robust to transmission error", *Proceedings IS&T/SPIE Wavelet Applications in Signal and Image Processing IV*, vol. 2825, August 1996, p. 890-900.

[CRE 96B] CREUSERE C., "Image coding using parallel implementation of the embedded zerotree wavelet algorithm", *Proceedings IS&T/SPIE Symposium on Electronic Imaging,* vol. 2668, San Jose, California, USA, 1996.

[CRU 97] CRUMP V., FISCHER T., "Intraframe low bitrate video coding robust to packet erasure", *Data Compression Conference*, Snowbird, Utah, USA, 1997.

[IRE 98A] IREN S., AMER P., CONRAD P., "NETCICATS : Network-conscious image compression and transmission system", *Springer-Verlag Lecture Notes in Computer Science : Advances in Multimedia Information Systems*, vol. 1508, September 1998, p. 57-68.

[IRE 98B] IREN S., AMER P., CONRAD P., "Network-conscious image compression over wireless networks", *Springer-Verlag Lecture Notes in Computer Science : Interactive Distributed Multimedia Systems*, vol. 1483, September 1998, p. 149-158.

[IRE 99] IREN S., "Network-conscious image compression", *PhD Dissertations*, CIS Department, University of Delaware, USA, 1999.

[IRE 00] IREN S., AMER P., "SPIHT-NC : Network-conscious zerotree encoding", *Data Compression Conference*, Snowbird, Utah, USA, March 2000, p. 313-322.

[MAN 97] MAN S., KOSSENTINI F., "Robust EZW image coding for noisy channels", *IEEE Signal Processing Letters,* vol. 4, n° 8, August 1997, p. 227-229.

[ROG 98] ROGERS J., COSMAN P., "Robust wavelet zerotree image compression with fixed-length packetization", *Data Compression Conference*, Snowbird, Utah, USA, March 1998.

[SAI 96] SAID A., PEARLMAN W., "A new, fast, and efficient image codec based on set partitioning in hierarchical trees", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, n° 3, June 1996.

[SHA 93] SHAPIRO J., "Embedded image coding using zerotrees of wavelet coefficients", *IEEE Transactions on Image Processing*, vol. 41, n° 12, December 1993, p. 3445-3462.

[SHE 99] SHERWOOD P., ZEGER K., "Macroscopic multistage image compression for robust transmission over noisy channels", In *SPIE,* vol. 3653, 1999.

[STE 94] STEVENS W., "TCP/IP Illustrated- The Protocols, volume 1", Addison-Wesley, MA, 1994.

IREN (S.)

Sami Iren received the BS degree in Computer Engineering from Middle East Technical University, Ankara, Turkey in 1992, and the MS and PhD degrees in Computer and Information Science in 1996 and 1999 respectively, from the University of Delaware, USA. After working at GTE Research Laboratories for a short period, he joined Marconi Plc. where he currently works as a Lead Software Engineer. His current research focus is on Traffic Management, Quality of Service, and Traffic Engineering. He is also interested in transport layer protocols, compression algorithms, and multimedia communication systems.

AMER (P.D.)

Paul D. Amer received the BS degree summa cum laude in Mathematics from SUNY at Albany in 1974, and the MS and PhD degrees in Computer and Information Science in 1976 and 1979, respectively, from The Ohio State University. Since 1979, he has been at the University of Delaware where currently he is professor of computer science. From 1978 to 1987, he was concurrently employed part-time as a Research Computer Scientist for the National Bureau of Standards in Wash, DC. In 1985, he spent a sabbatical year in Paris teaching at the Ecole Nationale Superieure des Telecommunications, and researching the formal description technique Estelle as part of the ESPRIT Project SEDOS at the Agence de l'Informatique. In 1992, Professor Amer spent his second sabbatical year in Toulouse at the Laboratoire d'Automatique et d'Analyse des Systemes (LAAS du CNRS) investigating a partial order transport protocol to support multimedia applications (see RFC1693). In 1996, he received a one year University Research Fellowship from the Center for Advanced Studies at the University of Delaware. In 1999, Professor Amer returned to Toulouse for his third sabbatical year sharing his time between LAAS and the Ecole Nationale Superieure d'Ingenieurs de Constructions Aeronautiques (ENSICA). Currently Professor Amer is investigating data compression in multimedia, and innovative transport layer services and protocols.