Convolutional Coding for Resilient Packet Header Compression

Vijay A Suryavanshi and Aria Nosratinia

Multimedia Communications Laboratory, The University of Texas at Dallas
Richardson, TX 75083-0688, USA
E-mail: {vas021000, aria}@utdallas.edu
Phone: (972) 883-2894

Keywords: Header compression, forward error correction, convolutional codes,

Abstract:
This paper proposes a system using convolutional codes to mitigate error propagation in packet header compression. Convolutional codes are a class of Forward Error Correction (FEC) codes, and their use is motivated because on uni-directional links loss of even one packet can render subsequent packets useless. A combination of two interleavers is used to address channel memory and increase the power of the code, and the optimum yet computationally efficient Viterbi algorithm is used for decoding at the receiver. Simulation results demonstrate the advantages of the proposed scheme.

Convolutional Coding for Resilient Packet Header Compression

Vijay A Suryavanshi and Aria Nosratinia
Multimedia Communications Laboratory, The University of Texas at Dallas
Richardson, TX 75083-0688, USA
E-mail: {vas021000, aria}@utdallas.edu

***Abstract*— This paper proposes a system using convolutional codes to mitigate error propagation in packet header compression. Convolutional codes are a class of Forward Error Correction (FEC) codes, and their use is motivated because on uni-directional links loss of even one packet can render subsequent packets useless. A combination of two interleavers is used to address channel memory and increase the power of the code, and the optimum yet computationally efficient Viterbi algorithm is used for decoding at the receiver. Simulation results demonstrate the advantages of the proposed scheme.**

I. Introduction

Recent investigations [1] indicate that 55% of packets have length less than 200 bytes. The current overhead of 40 bytes per RTP/UDP/IPv4 packet is thus inefficient. IPv6 introduces a higher overhead of 60 bytes, which further motivates packet header compression. In any header compression scheme, the header fields are differentially encoded and transmitted as a compressed packet header. Due to differential nature of the system, a single packet loss can render subsequent packets useless. On uni-directional links the receiver has to discard compressed headers unless it receives an uncompressed packet. Several schemes have been proposed that try to improve error resilience in header compression schemes.

We propose to use convolutional codes to improve the error resilience of header compression scheme. Packet headers are convolutionally encoded and the resulting parity bits are equally divided among the compressed packet headers. At the receiver side, the optimum yet computationally efficient Viterbi algorithm [2] is used for decoding the received bits. Simulation results indicate that our proposed scheme shows improvement over other existing schemes.

The present work extends and continues the work in [3], where a block coding scheme using Reed-Solomon codes was proposed for resilient packet header compression. That work demonstrated that encoding for packet header compression is a good idea, however, Reed-Solomon decoding can be computationally complex ($O(N^2)$ in the length of the code word) and does not provide a means of trading complexity for performance and vice versa. Convolutional codes are linear complexity in the length of the code word, and their performance can be adjusted by changing the memory of the code (number of states in the Viterbi trellis) without interfering with other system parameters. Unfortunately the block code design is not suitable for convolutional codes, therefore this paper is dedicated to the new design that uses convolutional codes, and evaluating its performance.

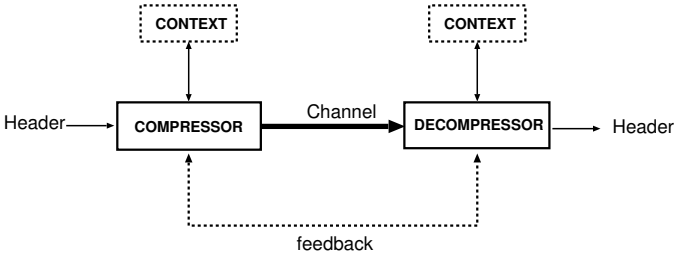

Fig. 1. Basic Header Compression Scheme: CONTEXT buffer stores the current uncompressed packet header

This paper is organized as follows. In the next section we describe the basic operations involved in any header compression scheme. In Section II we briefly discuss the existing schemes and techniques to mitigate error propagation. Section III gives a brief overview of convolutional codes and the Viterbi algorithm along with the interleaver to be used. Our scheme is discussed in Section IV. Finally we present simulation results in Section V and conclude in Section VI

A. Header Compression Basics

Packet header compression techniques can be understood by a quick look at the method of Van Jacobson [4], which successfully compresses a 40 byte TCP/IPv4 header to 4 bytes. The basic ideas behind this technique are:

- Do not re-transmit fully static fields, e.g. address fields
- Transmit dynamic fields, e.g., checksum fields
- For slowly varying fields, e.g., sequence number, transmit only the difference from last known value.

The same principles can be applied to compress a RTP/UDP/IP packet [5]. In fact RTP/UDP/IP packets are more compressible due to the fact that differences among header fields often remain constant or change by small amount. This leads to a compressed header of 2-4 bytes. For a brief overview of existing header compression schemes the reader is referred to [3], [6].

A basic header compression system has a CONTEXT buffer on each side of the link as shown in Figure 1. At the transmitter

and receiver, the last available header (uncompressed) is kept in the CONTEXT buffer as a reference. Compression or decompression takes place with respect to that reference, and the CONTEXT is updated regularly.

Some header fields stay *static* throughout a session whereas some header fields can be *inferred* from header fields belonging to a previous packet. Other fields such as the checksum are almost *random* in the sense that they cannot be inferred easily from other available information. At the start of a session the transmitter sends an uncompressed header which initializes the decompressor. *Static* fields are not included in the compressed header whereas *random* fields are included verbatim. The remaining fields are differentially encoded or not transmitted at all (*e.g., inferred* fields). On an average a 40 byte IPv4 (RTP/UDP or TCP) header can be compressed down to 3 bytes.

As long as the compressor and decompressor are synchronized, i.e., both CONTEXT buffers are identical in contents, packet headers can be decompressed correctly. The differential nature of the scheme, however, makes the system particularly vulnerable to packet losses. A lost packet desynchronizes the decompressor and subsequent packets are rendered useless, as shown in Figure 2. An uncompressed packet needs to be transmitted so that both sides are synchronized with each other again.

If a feedback link exists (bi-directional link) then the compressor can be informed about the status of the receiver, and upkeep of the CONTEXT is easier. On uni-directional links, however, the only thing that can be done is to periodically transmit uncompressed headers. Our contributions are especially targeted to help the vulnerable uni-directional links.

II. Existing Error Resilient Methods

As mentioned in previous section, feedback from the decompressor can force the compressor to send refresh packets. Different schemes have been proposed which address the problem of error propagation [7], [8], [9]. These schemes work in conjunction with information provided by the feedback path. On unidirectional links, periodic packets with uncompressed headers are transmitted to reduce error propagation. A longer refresh period gives better efficiency in terms of overhead, but on average it increases the packet discard rate when losses occur [3]. Even if feedback is possible, a long RTT can reduce the benefits of having feedback in first place.

When feedback is not possible, the decompressor can try to repair the CONTEXT locally. This involves some form of smart processing on part of decompressor to try and establish synchronization with the compressor. In [10] a mechanism is proposed wherein the decompressor tries to locally correct the desynchronized CONTEXT. The decompressor adds delta values of the last correctly received header TWICE to a correctly received compressed header if a packet is loss in between. The sanctity of the update after applying TWICE is verified by computing the checksum of the decompressed header at the transport layer. This algorithm is very effective in reducing error propagation only when the differences between the compressed headers is fairly constant and is thus heavily dependent on characteristic of the stream. Next we discuss two existing methods which try to achieve robustness against packet losses their own way.

A. W-LSB Encoding: ROHC U-mode

In LSB encoding of the header fields, only one reference value V_{ref} is stored in the CONTEXT. Incoming packet headers are compressed with respect to this V_{ref}. Therefore a single packet loss desynchronizes the decompressor. Instead of caching only one V_{ref} value, in RFC 3095 the compressor CONTEXT maintains a sliding window of previously compressed packet headers. Let us denote the maximum among these values in the window as v_{max} and similarly v_{min} is the minimum. When the compressor receives v (an uncompressed header which needs to be compressed), it calculates the range r as follows:

$$r = \max(|v - v_{max}|, |v - v_{min}|) \tag{1}$$

The number of LSB bits, k that need to be conveyed are then calculated as:

$$k = \lceil (\log_2(2r + 1)) \rceil \tag{2}$$

Next, the compressor adds this v to the existing window and updates v_{max} and v_{min} accordingly. The window is moved upwards by removing older V_{ref}'s and adding newer v's to the window.

The decompressor chooses the last correctly decompressed header as its reference value (say) v_{ref}. When the decompressor receives k LSB bits for a particular v, it chooses one as the decompressed value which is closest to v_{ref} and has the same k LSB bits. Obviously higher the value of k is, even if some packets are lost in between, with high probability the next correct packet is decompressed correctly. Hence, there is a trade-off involved in terms of value of k i.e., the window length, and resilience against error propagation. A longer window ensures better performance against packet losses but increases the number of bits k that need to be transmitted. Also on uni-directional links RFC 3095 recommend all compressed headers carry a CRC to verify correct decompression.

B. Multiple Updates: RFC 3545-ECRTP

Enhanced Compressed RTP (ECRTP), as explained in RFC 3545, proposes sending changes in header fields multiple times to maintain synchronization between the compressor and decompressor. The basic idea is to send updates in (say) N packets so that at least receiving one correct packet will maintain synchronization. This scheme heavily relies on TWICE in case packets are lost. Again use of TWICE entails inclusion of the UDP checksum field to verify correct decompression.

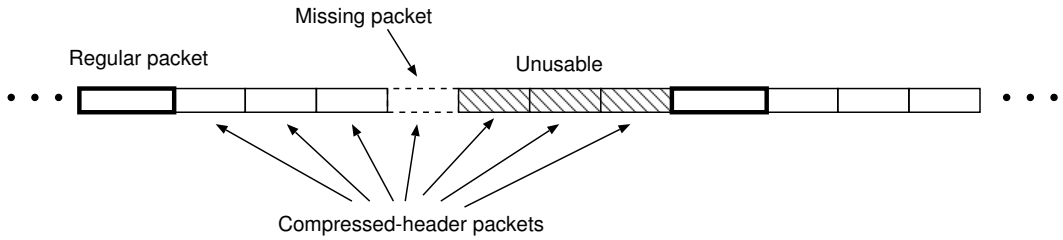

Fig. 2. Error Propagation due to single packet loss

We propose use of FEC to prevent error propagation on lossy uni-directional links. In the next section we discuss briefly our previous work in this area and explain the scheme proposed in this paper.

III. Convolutional Coding for Error Resilience

In [3] the authors introduced an FEC based scheme to prevent error propagation over uni-directional links involving header compression. Systematic Reed-Solomon (RS) codes are used to encode packet headers and the resulting parity bytes are equally divided among the compressed headers. Depending upon the rate of the code, varying performance is achieved. The decoding at the receiver side is accomplished using standard techniques such as the Berlekamp-Massey algorithm or Peterson-Gorenstein-Zierler algorithm [11]. The decoding complexity of RS codes is at least quadratic in block length of the code. Thus a very large block length requires high computational complexity at the decoder side.

Convolutional codes on the other hand have decoding complexity that is only linear in block length using the Viterbi algorithm. This motivates the idea of using convolutional codes for improving error resilience in header compression schemes. The Viterbi algorithm [2] is a maximum likelihood decoding algorithm and has close links with problems involving dynamic programming. In the next few sections we will discuss our proposed scheme.

A. Convolutional Codes and Viterbi Algorithm

Convolutional codes use a finite state machine for the calculation of the codeword. The output at each time depends on the input as well as past inputs [11]. The memory of the code K determines both the power of the code, as well as the computational complexity needed for decoding. A rate $\frac{k}{n}$ convolutional encoder accepts k bits at a time and outputs n bits at a time. In this paper we only consider rate-$1/2$ convolutional codes. The popularity of convolutional codes as error correcting codes is due to the optimal yet computationally efficient Viterbi decoding algorithm [2]. The Viterbi algorithm tries to maximize the probability of decoding a received sequence to a correct codeword by searching through the trellis of the code.

Error correcting performance of a convolutional code depends on the memory K and rate. Higher value of K and a lower rate imparts stronger error correcting capability to the code. Increasing K also increases the number of operations to be performed while decoding. The number of states in a trellis is exponential in K and the Viterbi algorithm has to calculate likelihood at each state. Specifically, for a code of block length N and memory K, the Viterbi algorithm requires $N2^{2(K+1)}$ operations.

With the loss of each packet a number of consecutive bits are lost. Convolutional codes are susceptible to the loss of consecutive bits, therefore interleaving must be used to avoid burst erasures to be "seen" by the code. An interleaver reorders the bits so that successive bit errors will appear to the code as randomly dispersed bit errors. The best interleavers are those that maximally separate nearby bits, an exmple of which follows.

B. S-Random Interleaving

An S-Random interleaver is based on the idea of semi-random permutations [12]. The basic idea is this:

Generate integers i, $1 \leq i \leq N$, and randomly select one integer at a time. If the current integer is equal to any of the previous S selections within a distance of $\pm S$, the current selection is placed back in the pool and another random integer is drawn.

This design procedure ensures that if two input bits to the interleaver π are within distance S then they are at least S distance apart at the output. Therefore considering two indices i, j such that

$$0 < |i - j| \leq S \tag{3}$$

then the design conditions ensures that

$$|\pi(i) - \pi(j)| > S \tag{4}$$

The design parameter S can be adjusted to obtain varying error correction performance. Although the searching time for this algorithm increases with S, for $S < \sqrt{\frac{N}{2}}$ an S-Random interleaver can be designed in reasonable time [12]. In the next section we will describe the basic idea of using convolutional codes along with an S-Random interleaver to achieve error resilience in header compression systems on uni-directional links.

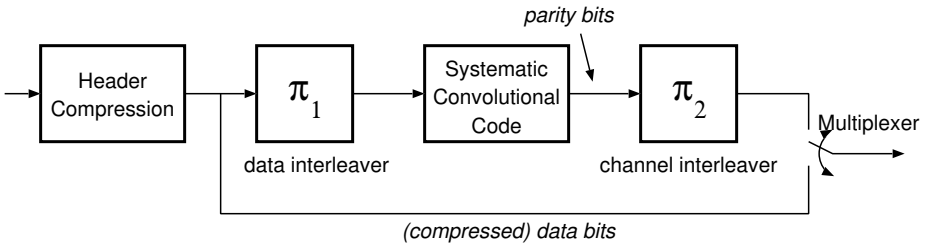

Fig. 3. Interleaving and Convolutional Encoding of header bits

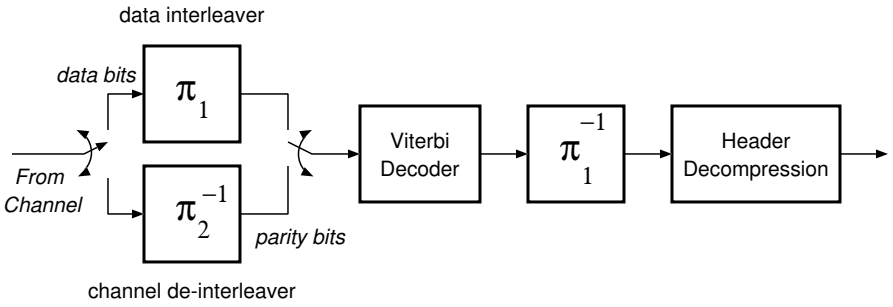

Fig. 4. Deinterleaving and Viterbi Decoding of header bits

IV. Proposed Scheme

We assume that the compressor outputs compressed headers of 2-3 bytes. Let α be the refresh period, i.e., the number of compressed headers between two uncompressed headers. The S-random interleaver is fed bits from α compressed headers and the uncompressed header. If u is the number of bits in the uncompressed header and c the number of bits in a compressed header, then a total of $u + \alpha c$ bits are given to the interleaver. The output bits from the interleaver are fed to a Recursive Systematic Convolutional (RSC) encoder. The parity bits coming out of this encoder are fed to an S-Random interleaver and the output interleaved parity bits are equally divided among the α compressed packet headers.

Interleaving the data bits ($u + \alpha c$ of them) before encoding ensures that the parity bits generated belong to data bits which are at least S distance apart. Correct reception of one such parity bit can contribute in reconstruction of many packets. The parity bits are also interleaved which ensures that even if a packet is lost, the missing parity bits are as far as possible at the time of decoding. Also the parity bits are equally distributed among the α compressed packets instead of transporting them in one single packet. This excludes the possibility of losing all the parity bits except when all the packets are lost (a rare event). In the next section we present simulation results and compare the performance of our scheme with existing schemes.

V. Simulations

We compare our scheme with ECRTP and ROHC operating in uni-directional mode. We us the RSC encoder given in [13] with rate 1/2, $K = 6$ and generator matrix $g^0 = [101011]$ and

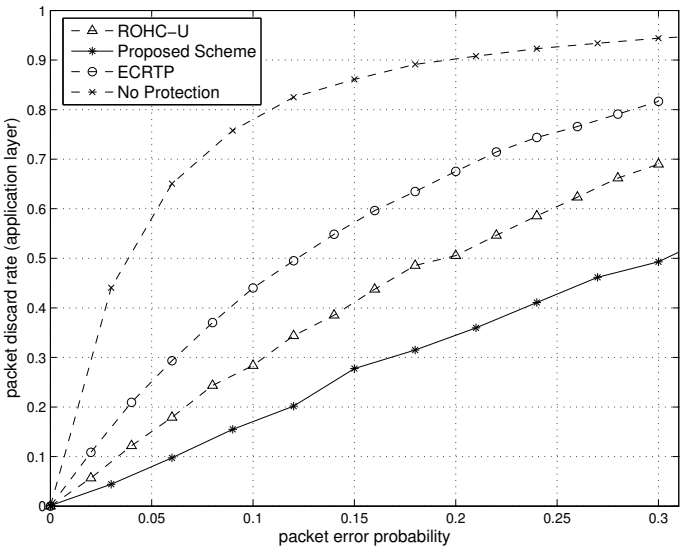

Fig. 5. IID Packet Loss: Comparison of Packet Discard Rates at the Application Layer

$g^1 = [111101]$. The number of bits in uncompressed header is $u = 320$ and length of compressed header is $c = 16$ bits. We set the refresh rate $\alpha = 40$. For the S-Random interleaver the design parameter $S = 16$. We consider two cases: i.i.d. packet losses and correlated packet losses. For a correlated packet loss model, a two-state Markov channel accurately describes the Internet packet loss characteristics [14]. For fair comparison, the end-to-end rate of the system is kept constant across comparisons.

It can be seen from Figure 5 and Figure 6 that the proposed

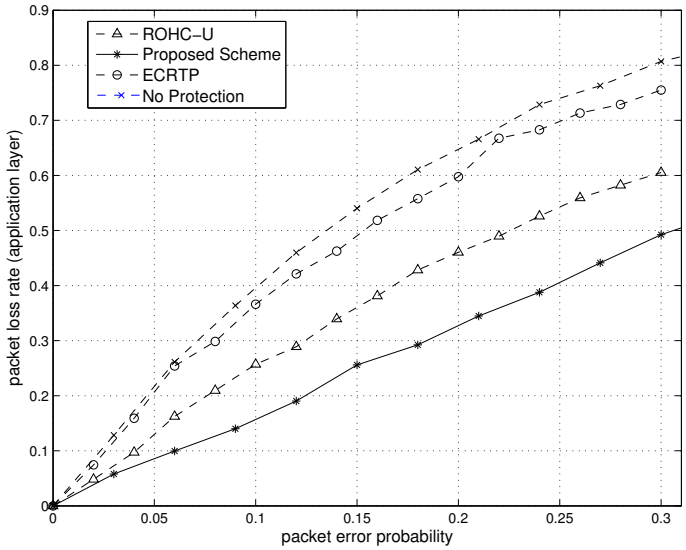

Fig. 6. Correlated Packet Loss: Comparison of Packet Discard Rates at the Application Layer. Burst length, $B_L = 5$

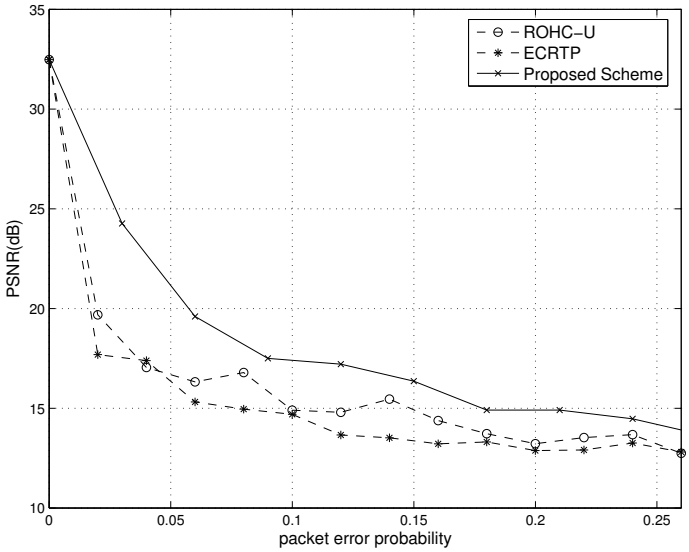

Fig. 7. Video Quality Evaluation

scheme performs very well compared to existing schemes. Note that the packet loss rate is the one observed at the application layer. Packets discarded by the decompressor never reach the application layer. We have not assumed any protection on the payload.

Bits saved due to compressing headers at the lower layers can be utilized to achieve a better presentation of the source data at the application layer. In [3] it was shown that a gain up to 1.5 dB can be achieved. Here we compare the performance of our scheme in video transport applications with the existing schemes. The *Foreman* QCIF sequence is encoded at 65 kbps with a H.263+ encoder and transmitted over an i.i.d. channel. Each packet carries only one GOB to improve error resilience at the video decoder. In case a GOB is lost, to reduce visual artifacts the GOB in the same spatial location is copied in the current location. The PSNR metric is used to compare the video quality in the three cases. Figure 7 also indicates that on average a 2 dB improvement in PSNR is obtained by using our proposed method.

VI. Conclusion

We present a scheme to improve error resilience in header compression schemes via use of convolutional codes. Compared to existing schemes the proposed scheme performs very well. Decoding convolutional codes introduces moderate complexity since the Viterbi algorithm is only linear in code length. Video simulations also indicate a better performance achieved by our scheme.

References

[1] CAIDA, "Packet Length Distributions," HTTP Link, August 2004, http://www.caida.org/analysis/AIX/.

[2] A. J. Viterbi, "Error bounds for convolutional codes and an asyptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. 13, pp. 260–269, April 1967.

[3] V. Suryavanshi, A. Nosratinia, and R. Vedantham, "Resilient Packet Header Compression through Coding," *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, vol. 3, pp. 1635–1639, 29 Nov.-3 Dec., 2004 2004.

[4] V. Jacobson, "TCP/IP Compression for Low-Speed Serial Links," RFC 1144 IETF Network Working Group, Feb 1990, http://www.ietf.org/rfc/rfc1144.txt.

[5] S. Casner and V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links," RFC 2508 IETF Network Working Group, Feb. 1999, http://www.ietf.org/rfc/rfc2508.txt.

[6] I. Joseph, "Survey of Header Compression techniques," NASA/TM - 2001-211154, Sept 2001, National Aeronautics and Space Administration (NASA).

[7] S. J. Perkins and M. W. Mutka, "Dependency Removal for Transport Protocol Header compression over noisy channels," *Proc. of IEEE International Conference on Communications (ICC)*, vol. 2, pp. 1025–1029, June 1997.

[8] M. Rossi, A. Giovanardi, M. Zorzi, and G. Mazzini, "Improved Header Compression for TCP/IP over Wireless Links," *Electronics Letters*, vol. 36, no. 23, pp. 1958–1960, November 2000.

[9] ——, "TCP/IP Header Compression: Proposal and Performance Investigation on a WCDMA Air Interface," *Proc. of the 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, vol. 1, pp. A 78–A 82, Sept 2001.

[10] M. Degermak, M. Engan, B. Nordgren, and S. Pink, "Low loss TCP/IP header compression for Wireless Networks," in *mobicam96*, New York, NY, October 1997.

[11] S. B. Wicker, *Error Control Systems for Digital Communcation and Storage*. Englewood Cliffs, NJ: Prentice Hall, 1995.

[12] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *The Telecommunications and Data Acquisition Progress Report 42-122*, pp. 56–65, 15 August 1995.

[13] P. Frenger, P. Orten, and T. Ottosson, "Convolutional codes with optimum distance spectrum," *IEEE Communications Letters*, vol. 3, pp. 317 –319, November 1999.

[14] B. Girod, K. Stuhlmueller, M. Link, and U. Horn, "Packet Loss Resilient Internet Video Streaming," *Proc. SPIE Visual Communications and Image processing 99*, vol. Vol.3653, pp. 833–844, Jan 1999.