Error-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

Network traffic statistics show that, because shorter packets predominate in many applications, headers impose a considerable overhead. But the header content is largely repetitive, thus in the past 15 years many packet header compression techniques have been proposed and studied. Header compression is based on differential methods, which introduces error propagation and leads to problems, e.g., in wireless links. This work presents error-resilient header compression by using forward error correction (FEC), which can significantly improve the throughput of header compression in both uni-directional and bi-directional links. The proposed methods are versatile and can work with pre-existing header compression schemes. Simulations demonstrate the advantages of the proposed system in terms of packet loss rates, throughput and delay.

Index Terms

Header compression, ROHC, error resilience

I. INTRODUCTION

Wireless bandwidth is precious. Every extra bit sent over the air not only uses scarce bandwidth, but also interferes with other users and depletes the mobile battery. One must be frugal with wireless bits.

Unfortunately TCP/UDP/IP packet transmission is anything but frugal. For routing and control purposes, packets have a typical header of 40 bytes in IPv4 and 60 bytes in IPv6. Actual traces of Internet traffic show [1] that a significant number of packets are only 40 bytes long, i.e. they have no payload, and a majority of packets are less than 300 bytes long. Based on the statistics derived from typical traces, headers impose a heavy overhead in terms of bitrate.

Thankfully the header data are redundant, so it is possible to compress them. Many header compression algorithms have been proposed and studied over the past 15 years (Figure 1). Van Jacobson [2] proposed a method that reduces packet overhead to an average of 4-5 bytes per packet. Subsequently, other methods were proposed that reduce the overhead further down to 2-3 bytes per packet. These methods depend on the similarity between successive packet headers, in a manner similar to DPCM. Thus, packet header compression shares the strengths and weaknesses of DPCM: excellent compression is achieved when headers are strongly correlated (as they usually are), but any errors will propagate and contaminate future packets.

To confront the error propagation problem in uni-directional links, existing schemes perform periodic refresh with uncompressed headers, thus periodically restoring synchronization between the transmitter and receiver. In bi-directional links, variations of ARQ with partial retransmission of the compressed information have been proposed. An overview of header compression methodologies in uni-directional and bi-directional links is presented in Section II.

The refresh method for the uni-directional link clearly leaves much to be desired. At high refresh rates, compression ratio is quickly eroded; at lower refresh rates, packet loss ensues. Motivated by the bursty nature of the losses in this application, and by the excellent burst-error correcting capability of Reed-Solomon (RS) codes, we propose to use systematic RS codes for the packet header compression problem in the uni-directional links. The distribution of parity bits over the packets is carefully designed to comply with existing protocols and to reduce the overall error rate. Significant improvements over previous solutions are obtained using this method.

We also design a reduced-complexity solution for the same channel using an interleaved convolutional code. The complexity advantage obtained at the cost of a higher residual packet loss rate, compared with the RS code.

We then turn to the bi-directional link, where feedback can be used to acknowledge the receipt of packets. Again we propose to use FEC methods, where (in principle) multiple packet headers will contribute to each parity bit. Therefore each of the (compressed) packet headers correspond to several symbols in our codewords, and the reverse link provides feedback several times *within* each codeword. This presents an unusual situation for code design, presenting a challenge how to use the feedback link effectively without cannibalizing the coding gain. We construct a predictive ARQ with multiple retransmission within a convolutional codeword. We also design and use a delay-limited interleaver. Using this structure, very attractive throughput-delay tradeoff is obtained in the presence of noise in both forward and reverse links. In fact, both the throughput and delay of the proposed system is superior to previous solutions under a large set of channel conditions.

To summarize, this work constructs a framework such that well-known channel codes can be gainfully and effectively applied to a new network-based application. The contribution of this work consists of the adaptation of Reed-Solomon and convolutional codes through the design of bit allocation and interleaving strategies for the header compression application, and thus obtaining much improved results that were previously unavailable.

The paper is organized as follows. In Section II we review header compression methodology, and present information needed in the remainder of the paper. Sections III and V present code design for the uni-directional and bi-directional links, respectively. Section IV discusses the design of interleavers.

II. Header Compression Methodology

Header fields of consecutive packets are not unrelated [2]. For example, consecutive packets in a session share the same source and destination address, so one may economize by sending them only once. The fields whose values are constant throughout a session are called *static* fields. Some other parts of the header are more or less unrelated from one packet to the next, e.g., the Time To Live (TTL), and are known as *random* fields. Finally, there are some parts, such as the packet number, that are not the same, but can be determined from past values. These are known as *inferred* fields.

Existing header compression methods use variations of (nonlinear) DPCM. Random fields are transmitted intact, while static and inferred fields are compressed by reference to the previous packet header(s). The entire process is a reversible (lossless) compression that is characterized by a finite state machine, whose state must be maintained and updated regularly at the compressor and decompressor alike. This state is known as CONTEXT.

It is well known that DPCM is a simple and powerful technique that can compress highly correlated sequences. In fact the best header compression techniques can compress a 40-byte IPv4 header down to 2–3 bytes (see Figure 1). However the basic DPCM assumes an error-free link; any errors in DPCM transmission will de-synchronize the compressor and decompressor states, thus future received values will be interpreted incorrectly. This problem is known as *error propagation*.

The problem of error propagation was recognized in the earliest works on header compression. The method of Van Jacobson [2] proposed periodic refresh to limit the damage caused by channel errors (Figure 2). Nevertheless, this method concedes lost packets between two refreshes. Perkins and Mutka [3] propose a more robust method where CONTEXT is stationary and does not change, i.e., all compression is done with respect to the last uncompressed packet header. Eventually, as differences between the current packet and last uncompressed packet grow, sending the differential may be unprofitable. Thus Calveras *et al* [4], [5] optimized the frequency of transmission of uncompressed headers in the method of [3]. Unfortunately these methods, while removing dependence among compressed headers, lose a significant part of the compression ratio.

Degermark *et al.* [6] observed that the statistics of the compressed headers are not random, but that some residual redundancy remains in them. Often times, the compressed headers are the same from one packet to the next. Thus, whenever a compressed header is lost, for the next packet, the CONTEXT is repaired by applying the received delta value twice, thus leading to the TWICE algorithm. The basic assumption is that the lost compressed header was the same as the next one received. This is a similar idea to what is known as *error concealment* in the image and video compression literature. The TWICE update is verified by the checksum of the decompressed header at the transport layer (TCP or UDP).

The basic ideas arising from these methods were adopted by the Internet Engineering Task Force (IETF) into various standard documents (RFC's). Compression of RTP packets via the basic Van Jacobson algorithm with the TWICE decoder is incorporated into RFC 2508 [7]. The Enhanced Compressed RTP (ECRTP) algorithm allows for multiple transmission of compressed information based on channel conditions, and is incorporated into RFC 3545 [8].

Perhaps the most effective header compression standard to date is known as RObust Header Compression (ROHC) RFC 3095 [9]. The fundamentals of ROHC are similar to previous header compression algorithms, but ROHC, through attention to detail, achieves superior compression. Since ROHC is widely considered to be the state of the art, we briefly describe some of its key aspects. Due to space constraints the concepts are presented in a simplified form; the interested reader is invited to consult [9] for a more detailed treatment. The reader who is familiar with ROHC may safely bypass the remainder of this section.

ROHC is in fact not one compression algorithm, but several compression algorithms bundled into one package. There are four *compression profiles* in ROHC (Uncompressed profile, RTP, UDP, and ESP profiles) each of them applies to a particular type of packet stream.

Also, based on the existence of feedback and/or reliability of the channel, ROHC operates in one of several *modes*. In unidirectional links, we have the unidirectional mode or U-mode, also known as ROHC-U. In the presence of feedback there are two modes, the reliable mode (ROHC-R) which is a conservative compression, and optimistic mode (ROHC-O), which has higher compression ratio.

Profiles and modes determine the details of the compression algorithm, so in principle, we can think of ROHC as a family of algorithms. The *profiles* adapt ROHC to the properties of the source, while the *modes* adapt ROHC to the properties of the channel.

ROHC senses the quality of the channel and moves between the different modes via an algorithm described by a finite state machine. Specifically, ROHC starts in the U-mode, and progresses to R-mode and O-mode. Whenever there are too many errors, it falls back down through the mode hierarchy. Furthermore, in each of the modes, the operation of ROHC compressor and decompressor is characterized by a finite state machine. To summarize, there is one small-scale FSM describing the operation of the compression algorithm, while there is (another) large-scale FSM driving the movement between different modes. The interested reader is referred to [9], [10] for a more detailed treatment of these subjects.

Among the important innovations in ROHC is the concept of Window-based LSB encoding (W-LSB). The basic idea behind W-LSB is as follows: instead of encoding the difference with respect to the last sample, one can encode the difference with respect to a "neighborhood" of the past few samples. If the neighborhood is such that it is uniquely identified by *any* of the past few samples, then one can decode (decompress) despite the loss of one or more samples, thus reducing the error propagation problem.

Let us assume that we wish to encode a sample value v and denote the maximum and minimum values in a prescribed window as v_{max} and v_{min} respectively. To obtain the description of "neighborhood", the compressor first calculates a range r that describes the size of the so-called neighborhood.

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

The number of bits, k that need to describe the position within this neighborhood are:

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

The k least significant bits (LSB) of v are thus its representative values. At the decompressor, these LSB bits replace the LSB bits of V_{ref}, the last reference value that is received correctly, to give v. Then the window, V_{ref}, v_{max} and v_{min} are all updated.

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 recommends all compressed headers carry a CRC to verify correct decompression, thus adding two bytes to the compressed packet header.

III. Case I: Uni-directional Links

In a unidirectional link, since there is no feedback, a compressed-header packet must be discarded if any of the preceding packets are lost. With forward error correction (FEC), many of these otherwise lost packets can be recovered. In this section we present FEC techniques based on Reed-Solomon codes and convolutional codes for the header compression in a uni-directional link.

A. Reed-Solomon Codes

There are two aspects of the header compression problem that give rise to bursts of errors. First, each packet header, even when compressed, consists of multiple bytes, thus the loss of each compressed header will result in the loss of 2-5 consecutive bytes. Second, the bursty nature of many channels of interest, e.g., the fading wireless channel, result in consecutive losses. These two factors together lead to a bursty error (equivalent) channel in our problem.

Motivated by the excellent burst-error correction properties of Reed-Solomon (RS) codes, we propose to use them for the packet header compression problem. A systematic RS encoder accepts K data symbols and generates $N - K$ parity symbols [11]. A symbol is made up of m bits and the maximum codeword length, $N \leq N_{max} = 2^m - 1$. In this paper we consider $m = 8$, i.e., each symbol is one byte and therefore $N_{max} = 255$. At the receiver, a RS decoder can recover these K data symbols if *any* K from N transmitted symbols are received correctly.

In a packet header compression system, periodically an uncompressed header is transmitted that is around 40 bytes long, and following that compressed packets are transmitted that are much shorter. Our encoder will combine the uncompressed and the compressed headers into one group of symbols, and calculates the parity bits for this group of symbols (see Figure 3). Since we do not wish to manipulate the protocols already in place, we propose to use a systematic code, so that the existing (compressed) packet headers will be transmitted as they are. Additional parity bits will be generated and loaded onto the packets in a manner to be described below.

In order to achieve a configuration of parity bits that will achieve good error performance, careful attention must be paid to how symbol losses are generated through packet losses, since the distribution of symbols and errors in this application is different from many of the channels where RS codes have been used. In particular, note that our symbol errors are highly correlated, even with i.i.d. packet losses, because each packet loss will entail the loss of several systematic bits (compressed header) as well as parity bits that may be loaded onto the packet.

Due to the nature of RS codes, i.e., decoding is affected by the total number of lost symbols (systematic + parity), the best course of action is to equalize the total number of systematic plus parity bits on each packet.[1] Therefore, since uncompressed headers are many times larger than compressed headers, parity bits must be loaded onto the compressed-header packets (see Figure 4) in a manner such that the total number of header bytes plus parity bytes for all packets are equal (as close as possible).

The details of the coding strategy are as follows. Consider an uncompressed header of u bytes and α compressed headers each consisting of c bytes. The RS encoder accepts these $u + \alpha c$ bytes and generates some parity symbols as shown in Figure 3. The parity symbols so generated are distributed equally among the α packets with compressed headers shown in Figure 4. The number of parity symbols generated is $x\alpha$, where x denotes the number of parity symbols per transmitted compressed header.

We now calculate the packet discard rate of the coded system under i.i.d. errors. Let s_a be the average number of symbols per transmitted packet, then $s_a = \frac{u+(x+c)\alpha}{1+\alpha}$. At the decoder we need $u+x\alpha$ symbols for successfully decoding the codeword; $K_p = \frac{u+x\alpha}{s_a}$ packets suffice this purpose. The probability of successfully decoding the codeword over an i.i.d. channel with packet loss probability p can be given as:

$$P_{RS} = \sum_{i=K_p}^{\alpha} \binom{\alpha}{i} p^{\alpha-i}(1-p)^i \tag{3}$$

The average packet discard rate of the coded system $\bar{m}_{cod}$ can then be calculated as:

$$\bar{m}_{cod} = \alpha\, p\, P_{RS} + \bar{m}\,(1 - P_{RS}) \tag{4}$$

[1]With an underlying assumption that packet loss probability is equal among different type of packets.

where $\bar{m}$ is the packet discard rate of the uncoded system given as [12]:

$$\bar{m} = \sum_{k=0}^{\alpha+1} k(1-p)^{(\alpha+1-k)}p = (\alpha+1) - \frac{1-p}{p}\left[1-(1-p)^{\alpha+1}\right] \tag{5}$$

It is possible to perform similar performance analysis under correlated errors using recursion relationships [13], which we omit in the interest of brevity.

Figure 5 shows the performance of various (N, K) RS codes where $u = 40$ bytes and $c = 2$ bytes for the i.i.d. channel. The proposed coded scheme performs well compared to the uncoded system. Decreasing the code rate provides better protection and hence better performance. Extensive experiments were also undertaken in bursty error channels, which are omitted due to editorial constraints on the number of figures. The interested reader is referred for further experimental results to [12].

B. Convolutional Codes

Although RS codes are powerful and give excellent performance, there are two disadvantages associated with them: delay and complexity.

The delay of the RS decoding is due to the fact that, if some symbols are lost, they cannot be recovered until after a prescribed number of systematic plus parity bits have been correctly received. To demonstrate, we calculate the delay for the simple case of i.i.d. errors. Assume a (N, K) RS code, take n_f to be the position of first channel error, and Δ to be the number of symbols in error prior to successful decoding. We must wait for decoding until exactly K correct symbols have been received, the location at which this happens is denoted by S. Note that symbols before n_f have zero decoding delay and symbols after S have no effect on decoding. Packets between n_f and S have delay $S - n_f$. The average delay can be calculated as:

$$\bar{D} = \frac{1}{N}\sum_{n=n_f}^{S}(S-n_f) \tag{6}$$

After substituting $S = K + \Delta$ the expected overall delay per packet $E[\bar{D}]$ is:

$$\begin{aligned} E[\bar{D}] &= E\left[\frac{(K+\Delta-n_f)^2}{2N}\right] \\ &\geq E\left[\frac{K(K+\Delta-n_f)}{2N}\right] \\ &= \frac{r}{2}\left(N(r+p)-\frac{1}{p}\right) \end{aligned} \tag{7}$$

where p is the probability of packet loss and r is the code rate. To calculate this bound, we have used the fact that Δ is less than the total number of errors in the codeword, which is binomially distributed, and n_f is geometrically distributed. Thus it is seen that the average delay of the Reed-Solomon decoding is on the order of the codeword length, i.e., it increases linearly with codeword length. In delay-sensitive situations, we may need to seek alternative solutions. Convolutional codes can be decoded with delay that is roughly on the order of the constraint length of the code, which is independent (and often much smaller than) the length of the codeword.

Another motivation for looking beyond RS codes is complexity. Reed-Solomon codes have excellent burst error correcting properties, but their decoding complexity is at least quadratic ($O(N^2)$) in length of the codeword. Convolutional codes have complexity that increases linearly with codeword length.

Convolutional codes perform well when errors in the codeword are random, but sometimes the losses are bursty. To improve the performance of convolutional codes when losses are bursty, interleaving must be employed. Interleaving does create additional delay, a question that we will address in the sequel.

As shown in Figure 6 at the transmitter a total of $u + \alpha c$ bits are fed to the interleaver. The output bits from the interleaver are fed to a Recursive Systematic Convolutional (RSC) encoder. Parity bits coming out of this encoder are fed to the interleaver and the output interleaved parity bits are equally divided among α compressed packet headers. Interleaving data bits ($u + \alpha c$ of them) before encoding ensures that the parity bits generated belong to data bits which are at least some samples apart. Correct reception of one such parity bit can contribute in reconstruction of many packets. The parity bits are also interleaved so that when a packet is lost, the missing parity bits are separated as much as possible in the codeword. 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). Similarly decoding is done as shown in Figure 6 at the receiver side.

In our simulations $u = 40$ bytes and $c = 2$ bytes. We compare our scheme with ECRTP and ROHC operating in uni-directional mode over an i.i.d. channel. The RSC encoder given in [14] with rate 1/2, $K = 6$ and generator matrix $g^0 = [101011]$ and $g^1 = [111101]$ is used. The results are shown in Figure 7. Extensive experiments were also undertaken in the bursty error channels, the results of which cannot appear here due to editorial constraints on the number of figures. The interested reader is referred for further experimental results to [15].

Interleaving always incurs delay. In general the maximum delay can be as large as the codeword length. In the next section we provide a delay-limited interleaver design algorithm for delay sensitive applications.

IV. Delay-Limited Interleaver Design

An interleaver π is characterized by a pair of spreading factors, indicating the distance properties of symbols at the input and output. The symmetric expression of spreading factor is the *s-parameter*, which is the largest number s such that symbols within distance s at the input are s samples apart in the output stream [16]. The delay of an interleaver is defined as:

$$\delta = \max_i \quad (\pi(i) - i) \tag{8}$$

Let $\mathcal{D}_{max}$ be the maximum allowable delay of the system (in bits). From 8 it can be seen that if an interleaver needs to be designed with $\mathcal{D}_{max}$ in view, then

$$\pi(i) \leq \mathcal{D}_{max} + i \tag{9}$$

The delay-limited interleaver design algorithm can be described as follows. First, an initial guess for the s-parameter is made. A necessary and sufficient condition for an interleaver with spread s and period P to exist is that $P \geq s^2$ [17] therefore our initial guess for s is $\lfloor \sqrt{P} \rfloor$. The algorithm tries to design the interleaver with the specified delay constraint and s parameter. If at any step of the algorithm the interleaver cannot satisfy the delay constraint for a given s, the s parameter is decreased by 1 and the algorithm starts all over again. The interleaver is designed similarly to the classic block interleaver but adhering to Equation (9).

As in the case of a block interleaver, we start by setting $J_{11} = 1$, i.e., the element on the upper left corner of the matrix is one. Recall that a column represents the input and a row represents the output of the interleaver, so every time a 1 is placed, the other elements in the corresponding row and column are set to zero. The next 1 is placed in the second row and $1+(s+1)^{th}$ column. Similarly for the third output bit of the interleaver, a 1 is placed in the third row and $2+(s+1)^{th}$ column. At some point, we will need to wrap back once we exceed the delay limit or when the column value exceeds the period P of the interleaver. In this case we place a 1 in the next least valued and unoccupied column. We continue to fill the matrix in a similar fashion.

The resulting interleaver matrix will have a block interleaver structure but some of the permutation indices are not allowed. The non-permissible indices are represented by $\mathbf{x}$. Thus the interleaver matrix is filled as in the case of a block interleaver but will have delay less than $\mathcal{D}_{max}$. Equation 10 shows the interleaver matrix $\mathbf{J}_{con}$ with spread $s = 1$ and maximum delay $\mathcal{D}_{max} = 7$ bits. The resulting $\mathbf{J}_{con}$ will have a trapezoidal appearance due to the delay constraint.

$$\mathbf{J_{con}} = \begin{bmatrix}
1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \mathrm{x} & \mathrm{x} & \mathrm{x} \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \mathrm{x} & \mathrm{x} \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \mathrm{x} \\
\cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1
\end{bmatrix} \tag{10}$$

V. CASE II: BI-DIRECTIONAL LINKS

In this section we present a coded ARQ technique for resilient packet header compression in the bi-directional links. Because the network ARQ function is available on a per-packet basis, while the codewords consist of multiple packets, it follows that in our system, multiple opportunities exist within a codeword to acknowledge partial receipt of codeword contents. Based on this partial information, a decision must be made on transmission of additional data and parity, in a way that is not exactly similar to any existing system known to the authors. We present an adaptive hybrid ARQ technique to solve this problem.

A. *Hybrid ARQ*

Classic ARQ achieves reliability by retransmissions of lost data [11], [18]. The ability to retransmit means that there are no packet losses in a delay-unlimited system, however, when the channel deteriorates, multiple retransmissions reduce the throughput of ARQ. To reduce the number of retransmissions the forward link can be encoded, a method known as hybrid ARQ. If the error pattern is not correctable, the receiver can then request retransmission of the erroneous block, thus maintaining the reliability of the system [19].

In the existing hybrid ARQ schemes [11] the decoder waits until a codeword is fully received, then tries to decode the codeword and see if retransmission is necessary. In our search for partial

ARQ methods we found [20], where the transmitted packet includes additional information (partial checksums) so that the rough location of error(s) can be determined at the decoder. Then, the acknowledgment will ask only for the corrupted data to be retransmitted. In this scheme the transmitter and receiver exchange more information that the traditional ARQ, however, still only one ACK/NACK is available per encoded block.

In contrast, we have multiple ACK/NACK per codeword. The question is how to use this feedback information in an efficient manner. The basic idea behind our method is as follows: feedback makes available to the encoder some information about the status of the decoder. Whenever a NACK is received, the encoder then makes an estimate of whether the decoder can continue to decode despite the error. Only if the (estimated) answer is in the negative, additional parity bits will be transmitted.

The block diagram of the system is shown in Figure 8. The encoder and decoder block are shown in Figure 6. Each time a packet is lost, the decoder transmits a NACK to the encoder. Once a NACK is received, the transmitter will mimic the decoding operation and will determine if this packet loss will result in a decoding failure. If so, it will retransmit the packet. Due to non-zero round-trip time, a buffer is provided at the encoder to allow for potential retransmissions.

For example, consider encoding of 10 packet headers as shown in Figure 6 at the transmitter side where each packet is denoted by P_i, $i \in \{1, 2, ..., 10\}$. Assume that packets P_1, P_4 and P_9 are lost in transition. The decoder, once P_1 is not received, sends a NACK corresponding to P_1's sequence number. Upon receiving this NACK, the transmitter mimics the receiver's Viterbi decoding and finds out that even with the loss of P_1 the decoding can still be successful, so no action takes place. After some time, a NACK is received for P_4. With *both* P_1 and P_4 missing, the transmitter determines that decoding will fail, so P_4 is re-transmitted until it is successfully received. Finally, a NACK for P_9 is received. From the viewpoint of the transmitter at this time, the receiver is missing only P_1 (since P_4 was successfully re-transmitted). So transmitter will mimic the receiver's decoding to see if it will fail. If so, P_9 will be retransmitted.

We compared, through simulations, the throughput and delay performance of the proposed scheme against the O-mode of ROHC. The packet headers have $u = 40$ bytes and compressed headers have $c = 2$ bytes. We use the RSC encoder given in [14] with rate 1/2, $K = 3$ and generator matrix $g^0 = [101]$ and $g^1 = [111]$. In all experiments the decoder timeouts after a period equivalent to round trip-time of a packet and retransmits the NACK. The feedback channel is assumed i.i.d. with a packet loss rate of 0.1. We use delay-constrained interleavers with $D_{max} = 100, 200, 600$ and 900 bits, designed with the algorithm of Section IV.

Figure 9 demonstrate the performance of the proposed predictive hybrid ARQ technique over a correlated (bursty error) channel. As expected, the average delay increases with D_{max} but is lower than ROHC-O mode for most channel conditions. In the extreme low packet loss regime the average delay of the proposed scheme is above ROHC-O, but overall the delay profile remains consistent even when the channel deteriorates. The spreading factor s of the interleaver increases as D_{max} increases, allowing for better error correction. This is evident from Figure 9 where the throughput improves as D_{max} increases. Experiments were also performed on the i.i.d. channel, whose results can be found in [21]. Overall the proposed scheme comfortably outperforms ROHC-O mode. The advantages are especially evident for throughput in the low to medium loss scenarios, and for delay in medium to high packet loss scenarios.

VI. CONCLUSION

This paper presents error-resilient methods for packet header compression, for both uni-directional and bi-directional links. For uni-directional links, we propose a system involving Reed-Solomon codes, and also a reduced-complexity reduced-delay system with convolutional codes. To maintain performance in bursty channels and yet maintain acceptable delay, a delay-limited interleaver was designed. In bi-directional links, a system involving convolutional codes combined with a new adaptive hybrid ARQ system is proposed. We show that under a wide spectrum of channel conditions the proposed system can significantly improve the performance of the packet header compression systems. Future work includes the use of turbo codes for the packet header compression application.

REFERENCES

[1] NLANR, “Wan packet size distribution,” HTTP Link, January 1997, http://www.nlanr.net/NA/Learn/packetsizes.html.

[2] 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.

[3] S. J. Perkins and M. W. Mutka, “Dependency removal for transport protocol header compression over noisy channels,” in *Proc. of IEEE International Conference on Communications (ICC)*, vol. 2, June 1997, pp. 1025–1029.

[4] A. Calveras, M. Arnau, and J. Paradells, “A controlled overhead for TCP/IP header compression algorithm over wireless links,” in *Proc. of 11th International Conference on Wireless Communications (Wireless’99)*, Calgary, Canada. 1999.

[5] ——, “An improvement of TCP/IP header compression algorithm for wireless links,” in *Proc. of Third World Multiconference on Systemics, Cybernetics and Informatics (SCI 99) and the Fifth International Conference on Information Systems Analysis and Synthesis (ISAS 99)*, vol. 4, July/August. Orlando, FL. 1999, pp. 39–46.

[6] 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.

[7] 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.

[8] T. Koren and et al, “Rfc 3545 - enhanced compressed rtp (ECRTP) for links with high delay, packet loss and reordering,” RFC 3545 IETF Network Working Group, July 2003, http://www.ietf.org/rfc/rfc3545.txt.

[9] C. Bormann and et al, “RFC 3095-RObust Header Compression (ROHC): Framework and four profiles: RTP,UDP,ESP, and uncompressed,” RFC 3095 IETF Network Working Group, July 2001, http://www.ietf.org/rfc/rfc3095.txt.

[10] E. Ertekin and C. Christou, “Internet protocol header compression, robust header compression, and their applicability in the global information grid,” *IEEE Communications Magazine*, vol. 42, pp. 106–116, November 2004.

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

[12] V. Suryavanshi, A. Nosratinia, and R. Vedantham, “Resilient packet header compression through coding,” in *Proc. IEEE GLOBECOM*, vol. 3, Dallas, TX USA, 29 Nov.-3 Dec. 2004, pp. 1635–1639.

[13] E. O. Elliot, “A model of the switched telephone network for data communications,” *Bell Syst. Tech. J.*, vol. 44, no. 1, pp. 89–109, January 1965.

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

[15] V. Suryavanshi and A. Nosratinia, “Convolutional coding for resilient packet header compression,” in *Proc. IEEE GLOBECOM*, November 2005, accepted, manuscript available at http://www.utdallas.edu/$\sim$vas021000/globecom05.pdf.

[16] C. Heegard and S. B. Wicker, *Turbo Coding*. Kluwer International Series in Engineering and Computer Science, 1999.

[17] V. Tarokh and B. Hochwald, “Existence and construction of block interleavers,” in *Proc. of IEEE International Conference on Communications (ICC)*, vol. 25, no. 1, April 2002, pp. 1855–1857.

[18] S. Lin, D. J. Costello, and M. Miller, “Automatic repeat request error control schemes,” *IEEE Communications Magazine*, vol. 22, no. 12, pp. 15–17, December 1984.

[19] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*. Pearson– Prentice-Hall., 2004.

[20] H. S. Cheng, G. Fairhurst, and N. Samaraweera, “Efficient partial retransmission arq strategy with error detection codes by feedback channel,” in *IEE Proceedings - Communications*, vol. 147, no. 5, October 2000, pp. 263–268.

[21] V. Suryavanshi and A. Nosratinia, “A hybrid ARQ scheme for resilient packet header compression,” in *Proc. Asilomar Conference on Signals, Systems and Computers*, October 2005, submitted, manuscritp available at http://www.utdallas.edu/∼vas021000/asilomar05.pdf.

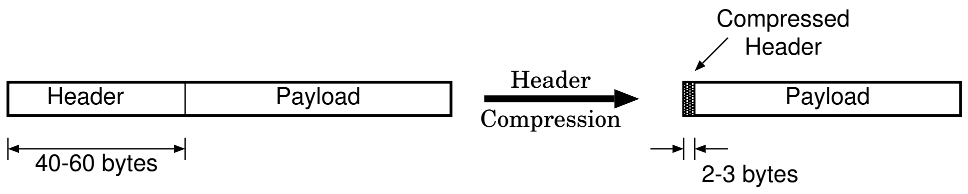


Fig. 1. Reducing overhead through header compression

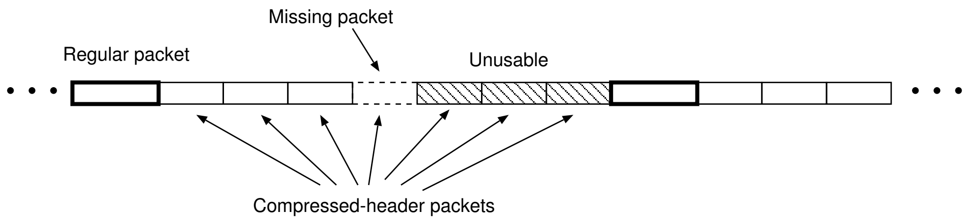


Fig. 2. Error Propagation due to single packet loss

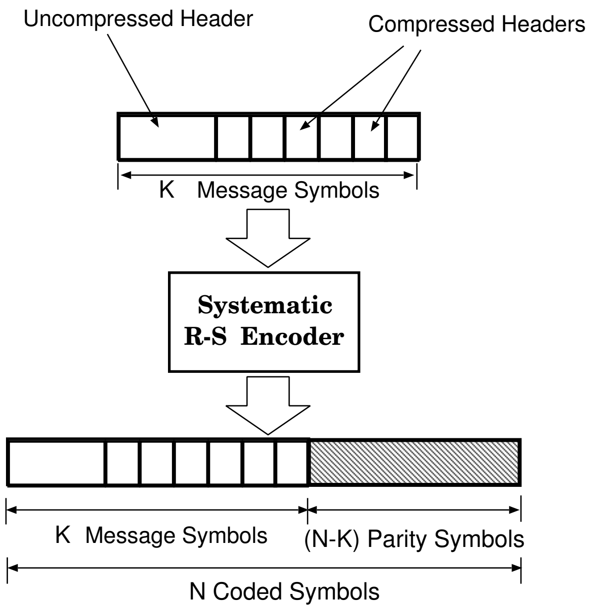


Fig. 3. Generation of parity symbols from packet headers

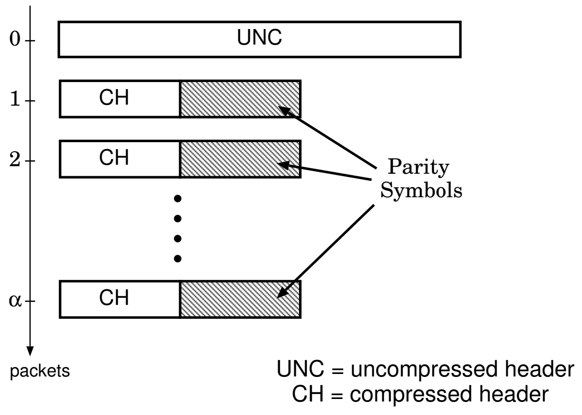


Fig. 4. Equal distribution of parity symbols among compressed packet headers. There are α compressed packet headers

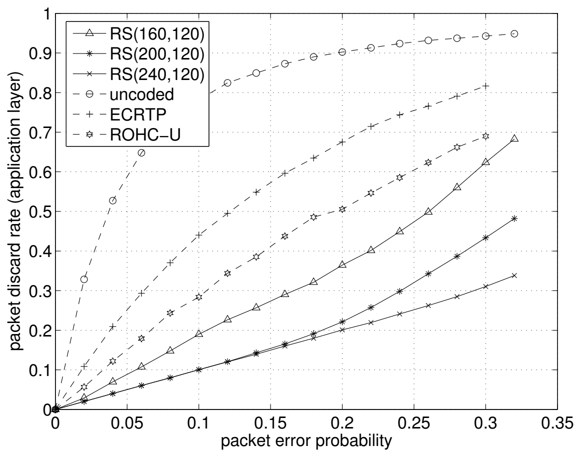


Fig. 5. Performance of RS coded system on iid unidirectional channel

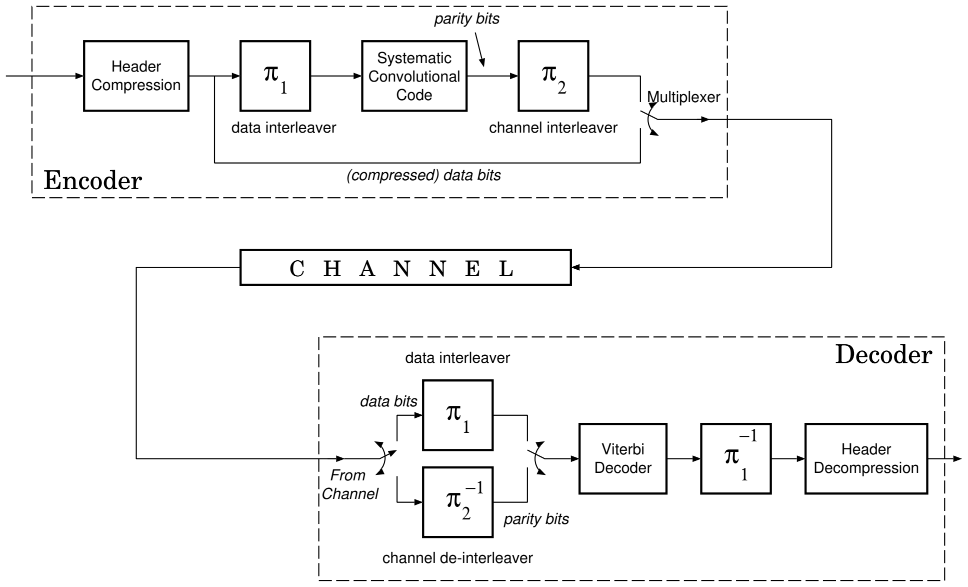


Fig. 6. The header compression system with convolutional codes

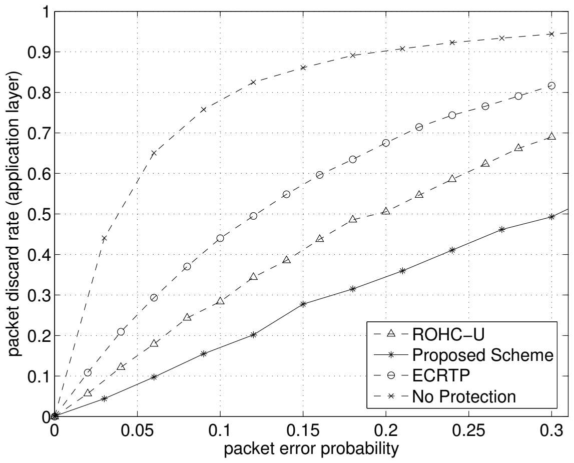


Fig. 7. Performance of interleaved and convolutionally coded system over iid unidirectional channel

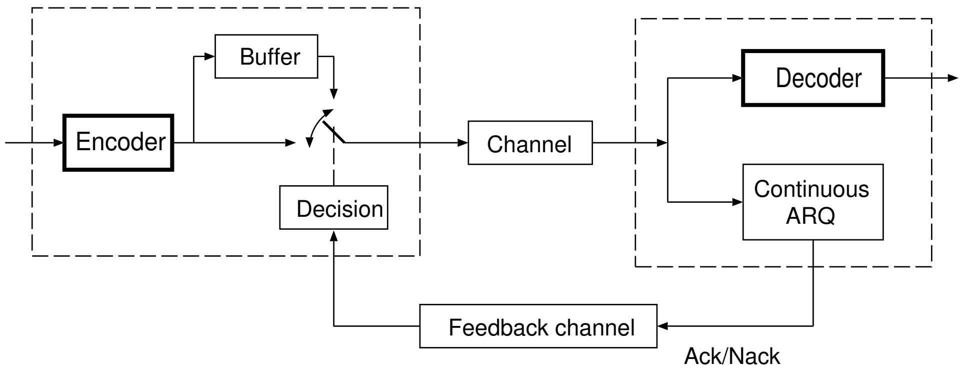
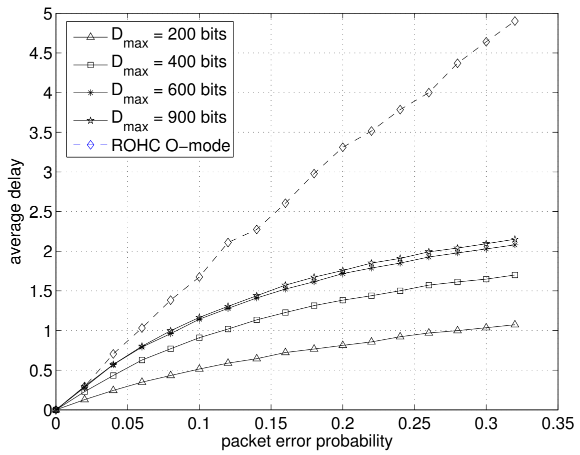


Fig. 8. Predictive Hybrid ARQ – the blocks entitled "encoder" and "decoder" are delineated in Figure 6



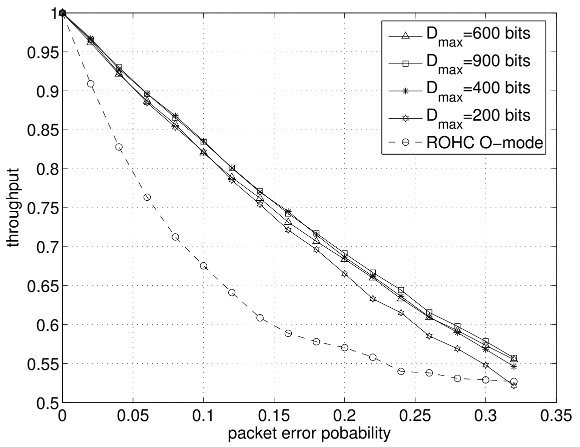


Fig. 9. Delay and throughput, correlated bidirectional channel with burst length, $B_L = 5$