

CODING TECHNIQUES FOR RESILIENT PACKET HEADER COMPRESSION

APPROVED BY SUPERVISORY COMMITTEE:

Dr. Aria Nosratinia, Chair

Dr. Andrea Fumagalli

Dr. Hlaing Minn

Copyright 2005

Vijay A Suryavanshi

All Rights Reserved

To my parents for believing in me

CODING TECHNIQUES FOR RESILIENT PACKET HEADER COMPRESSION

by

VIJAY A SURYAVANSHI, B.E.

THESIS

Presented to the Faculty of
The University of Texas at Dallas
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT DALLAS

August 2005

ACKNOWLEDGEMENTS

I express my sincere gratitude and appreciation to my advisor Dr. Aria Nosratinia. Without his unwavering patience and confidence in me, this thesis would not have been possible. For the past two years being a part of his research group has been an exciting challenge and privilege.

I also thank Dr. Andrea Fumagalli and Dr. Hlaing Minn to agree on serving my supervisory committee.

Doing research at MCL also gave me the opportunity to make many friends. Todd, Ahmad, Mohammad, Ramakrishna, Shahab, Harsh, Name, Ramy, Negar and Ali, thank you all for your help. In particular, I would like to thank Ramakrishna and Harsh for providing me with software for my experiments. It has been an absolute pleasure working with all of you guys.

I cannot express in words how deeply I feel about the support and confidence my parents have shown in me over the years. Everything seems insignificant compared to the sacrifices they have made to educate their children. Thank you, thank you very much. Also I express my heartfelt thanks to my girlfriend Mangala for her love, support and understanding. Thank you for enduring my absence.

Finally, I would like to thank my friends, cricket buddies and especially my roommates. Living away from home and family is never a joy; thank you guys for the good times.

July 2005.

CODING TECHNIQUES FOR RESILIENT PACKET HEADER COMPRESSION

Vijay A Suryavanshi, M.S.E.E.

The University of Texas at Dallas, 2005

Supervisor: Dr. Aria Nosratinia

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. Well-known error correcting codes such as Reed-Solomon and convolutional codes are utilized in this direction. For delay-sensitive applications we also propose a delay-limited interleaver design. The proposed methods are versatile and can work with pre-existing header compression schemes. Simulations demonstrate advantages of the proposed system in terms of packet loss rates, throughput and delay.

TABLE OF CONTENTS

Acknowledgements	v
Abstract	vi
List of Figures	ix
List of Tables	xi
Chapter 1. Introduction	1
1.1 Header Compression: State-of-the-art	2
1.2 Synopsis of this Thesis	3
Chapter 2. Header Compression	5
2.1 Basics of Header Compression	5
2.2 Error Propagation	7
2.3 Existing Schemes	8
2.4 IETF Standards	11
2.4.1 RObust Header Compression (ROHC)–RFC 3095	11
2.4.1.1 WLSB encoding of compressed header fields	16
2.4.2 Enhanced Compressed RTP (ECRTP)–RFC 3545	17
2.5 Summary	17
Chapter 3. Uni-directional Links	18
3.1 Channel Models	18
3.2 Performance of Header Compression on Uni-directional Links	20
3.3 Motivation for Coding Packet Headers	21
3.4 Reed-Solomon Codes	22
3.4.1 Performance analysis of RS coded system	24
3.4.2 Simulation results	26
3.5 Convolutional Codes	30

3.5.1	Delay-limited interleaver design	33
3.5.2	Simulation results	37
3.6	Summary	40
Chapter 4.	Bi-directional Links	41
4.1	Hybrid ARQ Techniques	41
4.2	Predictive Hybrid ARQ	42
4.3	Simulation Results	44
4.4	Summary	48
Chapter 5.	Conclusion and Future Work	53
	Bibliography	55
	VITA	

LIST OF FIGURES

1.1	Packet length statistics (from [1])	2
1.2	Reducing overhead through header compression	3
2.1	TCP/IPv4 packet header	6
2.2	Header compression architecture.	7
2.3	Error Propagation due to single packet loss	8
2.4	Van Jacobson method. Each packet header is compressed with respect to its previous packet header	9
2.5	A scheme by Perkins <i>et.al.</i> Each packet header is compressed with respect to a fixed uncompressed header	10
2.6	The TWICE algorithm. Differences are applied <i>twice</i> to a packet if a previous packet is lost	11
2.7	State transition among compressor states	12
2.8	State transition among decompressor states	13
2.9	Mode transition in ROHC	15
3.1	A binary erasure channel	19
3.2	A channel modeled on a two-state Markov process	20
3.3	Generation of parity symbols from packet headers	23
3.4	Equal distribution of parity symbols among compressed packet headers. There are α compressed packet headers	24
3.5	Performance of RS codes over the uni-directional i.i.d. channel	26
3.6	Performance comparison of RS codes with ECRTP and ROHC on i.i.d. uni-directional channel	27
3.7	Performance of RS codes over the uni-directional bursty channel with $B_L = 5$	28
3.8	Performance of RS codes over the uni-directional bursty channel with $B_L = 10$	29
3.9	Performance of RS codes over the uni-directional bursty channel with $B_L = 10$	29
3.10	Comparison of average location of first error event.	30
3.11	Illustration of decoding delay in RS coded systems.	30
3.12	Interleaving and convolutional encoding	32
3.13	Deinterleaving and decoding convolutional codes via Viterbi algorithm	33

3.14	Block interleaver with $s = 3$ and $\mathcal{D} = 9$ bits.	34
3.15	Delay constrained interleaver with $s = 1$ and $\mathcal{D}_{max} = 7$ bits.	35
3.16	Performance of the interleaved convolutional code for the uni-directional i.i.d. channel	37
3.17	Performance of the interleaved convolutional code for the uni-directional correlated channel	38
3.18	Performance of various constraint length convolutional encoders for the uni-directional i.i.d. channel	39
3.19	Performance of various constraint length convolutional encoders for the uni-directional correlated channel	39
4.1	Predictive Hybrid ARQ – the blocks entitled “encoder” and “decoder” are delineated in Figure 3.12 and 3.13 respectively	43
4.2	Performance of the predictive hybrid ARQ method over an i.i.d. channel with encoder constraint length $K = 3$	46
4.3	Performance of the predictive hybrid ARQ method over a correlated chan- nel ($B_L = 5$) with encoder constraint length $K = 3$	47
4.4	Delay and throughput performance of predictive hybrid ARQ over an i.i.d. channel. Constraint length, $K = 4$	49
4.5	Delay and throughput performance of predictive hybrid ARQ over an i.i.d. channel. Constraint length, $K = 6$	50
4.6	Delay and throughput performance of predictive hybrid ARQ over a corre- lated channel ($B_L = 5$). Constraint length, $K = 4$	51
4.7	Delay and throughput performance of predictive hybrid ARQ over a corre- lated channel ($B_L = 5$). Constraint length, $K = 6$	52

LIST OF TABLES

2.1	IPv4 Header Field Classification	6
3.1	Generator functions for various rate 1/2 convolutional codes from [2] . . .	38

CHAPTER 1

INTRODUCTION

Providing IP-based wireless multimedia services is one of the most exciting and challenging aspects of next generation wireless networks. Typically, multimedia applications require timely and accurate delivery of packets to maintain acceptable quality. In these situations protocols such as RTP (Real-time Transport Protocol) [3] along with UDP/IP (User datagram Protocol/Internet Protocol) are used. Specifically, RTP oversees latency requirements whereas UDP/IP provides network functionalities. Each protocol layer appends its own control information in form of an header. For example, data generated at the application layer forms the payload part of an RTP/UDP/IP packet with 40 bytes of header information attached. This represents a moderate overhead if the payload is equal to few thousand bytes. This is far from the truth.

Actual traces of Internet traffic show [1] (Figure 1.1) 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. In another study it was found that over 55% of packets have length less than 200 bytes [4]. Thankfully the header fields in successive packets are redundant, so it is possible to compress them. Many header compression algorithms have been proposed and studied over the past 15 years. Several IETF (Internet Engineering Task Force) standards (RFCs- Request For Comments) have been proposed to converge these methods to define a common platform for header compression.

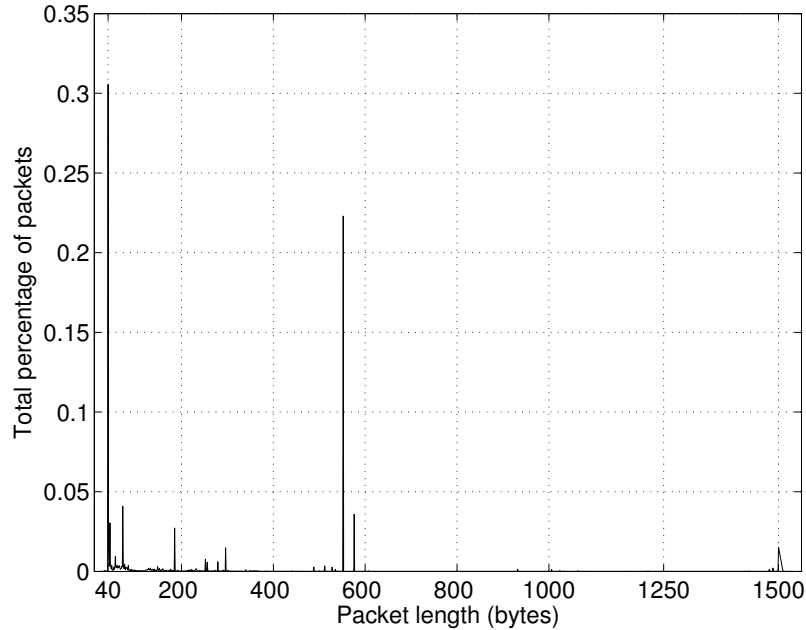


Figure 1.1. Packet length statistics (from [1])

1.1 Header Compression: State-of-the-art

RFC 1144 [5](see Figure 1.2) proposes a method that reduces packet overhead to an average of 4-5 bytes per packet. Subsequently, other header compression algorithms 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.

Loss of a single packet thus means that subsequent packets have to be discarded. 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. RFC 3545 [6]

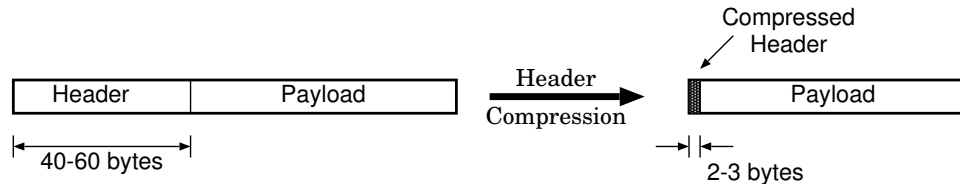


Figure 1.2. Reducing overhead through header compression

proposes sending multiple update so that correct receipt of one such update can stop error propagation. The number of times an update is transmitted depends upon implementation and channel conditions. In RFC 3095 [7] header compression is presented as a finite-state machine with a specialized encoding method to compress header fields. This improves compression efficiency at the same time maintains robustness.

1.2 Synopsis of this Thesis

In this work we propose to use error correcting codes to limit error propagation in header compression schemes. We present coding techniques based on well known error correcting codes such as Reed-Solomon and convolutional codes. On uni-directional links, the refresh approach clearly leaves much to be desired. At high refresh rates, compression ratio is quickly eroded; at lower refresh rates, packet loss ensues. Motivated by 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 packet header compression problem in the uni-directional links.

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 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 trade off 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.

The outline of this thesis is as follows. Chapter 2 introduces the basic concepts and motivation behind header compression. Error propagation and various existing schemes that try to overcome it are also presented. Two widely deployed IETF standards, ROHC and ECRTP are also reviewed briefly. In Chapter 3 we introduce the idea of coding for header compression over uni-directional links. Coding techniques based on Reed-Solomon and convolutional codes is explained. We also provide performance analysis of the uncoded as well as coded system. For delay sensitive applications a delay-limited interleaver design algorithm is described as well. Chapter 4 provides a new predictive hybrid ARQ technique for bi-directional header compression. Finally, we conclude in Chapter 5 and discuss possible future research in this area.

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.

CHAPTER 2

HEADER COMPRESSION

2.1 Basics of Header Compression

A TCP/ IPv4 packet header is shown in Figure 2.1. In a given session ¹ the address of the sender (*source address*) and receiver (*destination address*) stay same. Thus transmission of 8 bytes occupied by the address fields per packet is redundant. This static nature is observed in other header fields as well. For example, the *protocol* field as well as the *vers* field stay the same packet-to-packet. A basic header compression algorithm applies this observation by transmitting the *static* fields only once instead of repeating them in every packet. A still closer look at the packet header reveals that some header fields can be *inferred* from other fields. The *packet length* can be calculated from the total packet length at the link layer. Further more it is observed that the difference between some fields stay constant or differ by very small amount. Thus, only the differences need to be conveyed. Table 2.1 shows classification of the header fields.

To summarize:

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

¹By a session we mean flow of packets between two nodes

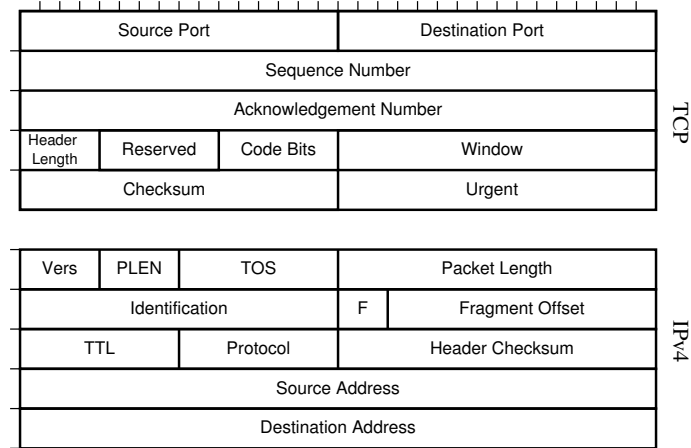


Figure 2.1. TCP/IPv4 packet header

Table 2.1. IPv4 Header Field Classification

Header Field	Size (bits)	Type
Version	4	<i>static</i>
Header Length	4	<i>static</i>
Type of Service	8	<i>random</i>
Packet length	16	<i>inferred</i>
Identification	16	<i>random</i>
Fragment Offset	13	<i>static</i>
TTL	8	<i>random</i>
Protocol	8	<i>static</i>
Checksum	16	<i>inferred</i>
Address	32	<i>static</i>

Based on above discussion, a 40 byte TCP/IPV4 packet header can be compressed down to 3–4 bytes [5]. The common approach in any header compression algorithm is to transmit a regular packet initially—known as uncompressed header—and then transmitting only the differences between the current packet and previous packet.

A typical header compression system is shown in Figure 2.2. At the transmitter, the header compression module compresses a packet header passed on by the upper layer (IP layer). Compression takes place with respect to a previous uncompressed header stored in

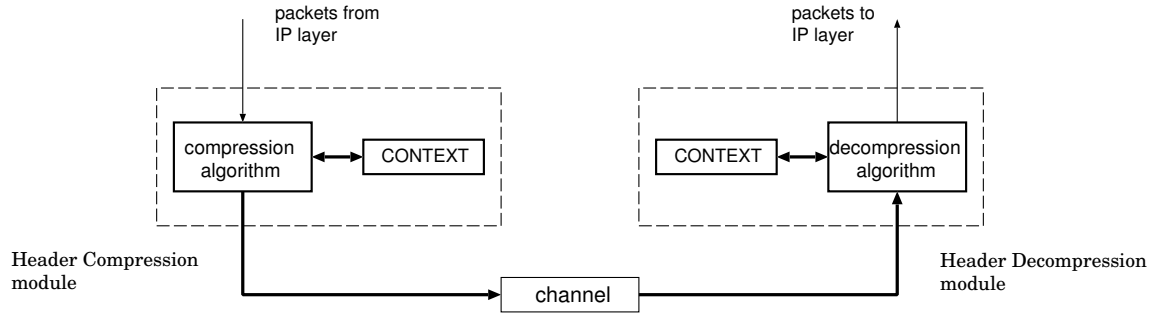


Figure 2.2. Header compression architecture.

a buffer known as the CONTEXT buffer. The compressed header is transmitted over the channel and the current contents of the CONTEXT is replaced by uncompressed version of a recent packet header. Similarly, at the receiver side header decompression module decompresses a compressed header with respect to the CONTEXT at its side. After decompression the CONTEXT is updated by the decompressed version of the compressed header.

Thus a header compression algorithm has a DPCM like structure. An incoming packet header is compressed by transmitting only the differences with respect to a previous packet header and so on. Header compression schemes enjoy all the benefits of DPCM but also share a drawback, error propagation.

2.2 Error Propagation

For successful decompression at the receiver side, the CONTEXT should contain the information of the previous packet header. If a packet is lost or corrupted, CONTEXT is corrupted and decompression of subsequent packets is erroneous. This is known as error propagation. Figure 2.3 shows the effect of a single packet loss on subsequent packets. Note that even though packets after the first loss event are received, after decompression they will fail checksum at upper layers and thus discarded. Therefore at each step in

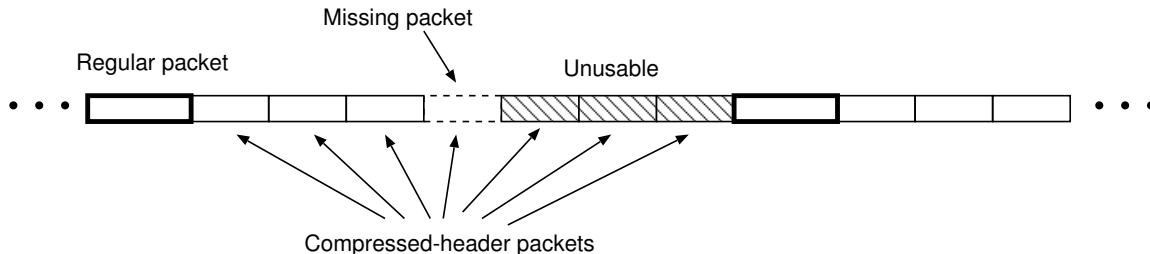


Figure 2.3. Error Propagation due to single packet loss

compressing/decompressing packet headers, the CONTEXT on each side should be in synchronization.

A simple solution on uni-directional links is to transmit an uncompressed packet at regular intervals. This will periodically refresh the CONTEXT at the decompressor side and erase any effects of previous losses. The refresh approach offers a trade-off between compression and efficiency. A higher refresh rate will reduce error propagation, but compression efficiency suffers due to frequent transmission of an uncompressed header.

When feedback is present, the receiver can notify the transmitter when a packet is lost. The transmitter responds by transmitting an uncompressed version of the lost packet. This resynchronizes the CONTEXT on both sides and further decompression of headers is error-free. Performance in the feedback case heavily relies on round-trip time of the system.

2.3 Existing Schemes

The idea of compressing packet headers dates back at least to 1990 when Van Jacobson proposed a header compression technique to compress TCP/IP_{v4} headers over bandwidth-limited links [5]. The key concepts behind his algorithm are identical to the one described in Section 2.1. Figure 2.4 depicts the Van Jacobson algorithm where headers are compressed with respect to a previous packet header. As noted in Section 2.2 the

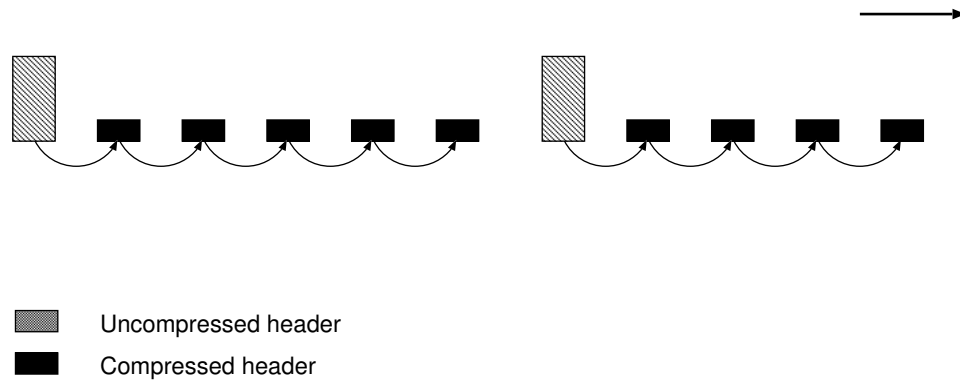


Figure 2.4. Van Jacobson method. Each packet header is compressed with respect to its previous packet header

differential nature of encoding introduces error propagation. To combat error-propagation many schemes have been proposed. These techniques modify the original Van Jacobson idea to improve its resilience on lossy channels.

Instead of compressing with respect to a previous packet header, Perkins and Mutka [8] proposed to use a *fixed* reference header at both CONTEXT. The method of Perkins *et al* is shown in Figure 2.5. This technique clearly avoids error propagation since a lost packet does not affect the compression of subsequent packets. The scheme has one drawback, however: the length of a compressed header grows as its distance from the reference increases. This is due to the fact that values in the header fields increase from packet-to-packet. Thus compression efficiency suffers.

Calveras *et al.* [9, 10] optimized the method by Perkins *et al.* as to when to transmit an uncompressed header. The basic idea is to transmit an update header based on the information provided by the decompressor. This checks the growth of compressed header length that are far away from the reference header. The decision as to when to transmit the update is made upon the feedback. The optimization is a function of this feedback information. When the reference is erroneous neither [8] nor [9, 10] nor [5] will work.

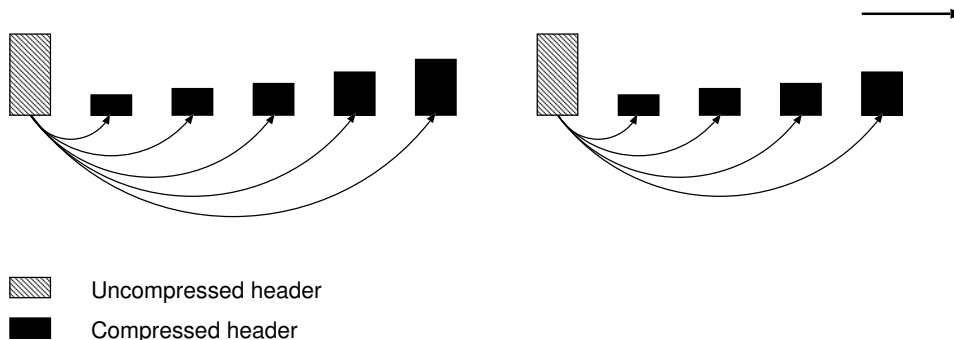


Figure 2.5. A scheme by Perkins *et al.* Each packet header is compressed with respect to a fixed uncompressed header

Rossi *et al.* [11, 12] proposed a different mechanism for the update. Similar to the method in [9, 10], the update is a compressed header compressed with reference to the current CONTEXT. In addition to this, Rossi *et al.* propose to send a *request flag* in the update packet. The decompressor receives this update but does not use it as its current reference. Instead it is stored in a buffer and the decompressor sends an ACK indicating correct receipt of the update. The compressor responds by compressing subsequent packets with respect to this reference. Likewise, the decompressor can now use this *flagged* update as its new CONTEXT. This method requires feedback and performance degrades if the ACKs are corrupted on their way.

A rather localized approach, introduced by Degermark *et al.*, is known as the TWICE algorithm [13]. Degermark's idea was to apply the delta values (i.e., packet header differences) *twice* to the CONTEXT whenever an intermediate packet is lost. The underlying assumption is that most of the delta values are the same from packet to packet. Thus if a packet is lost, the differences in the previous compressed packet is same as that of the lost packet. When the next packet arrives, differences from the previous packet is applied *twice* and thus error propagation is avoided (see Figure 2.6). The sanctity of the update (after applying TWICE) is verified by computing checksum of the decompressed header at the upper layers (TCP layer). This method does not require feedback but

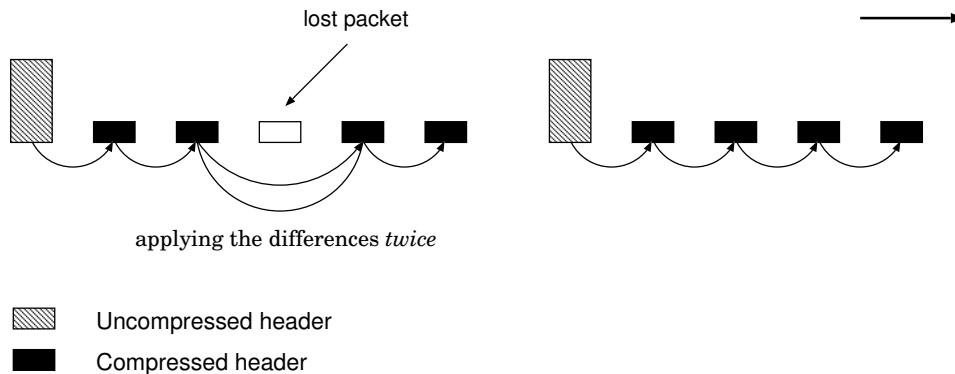


Figure 2.6. The TWICE algorithm. Differences are applied *twice* to a packet if a previous packet is lost

performance depends heavily upon the characteristic of the packet streams.

2.4 IETF Standards

RFC 1144 is based on Van Jacobson's method of compressing TCP/IPv4 headers. Based on these very principles, RTP/UDP/IP packets can be compressed as well (RFC 2508). The aforementioned methods are suitable when the channel is relatively error free. For wireless channel a bit-error rate of 10^{-3} is not uncommon, therefore, in spite of excellent compression efficiency these methods are not suitable over wireless channels.

Precisely with this in mind, IETF developed a header compression framework for wireless channels, known as Robust Header Compression (ROHC). ROHC [7] defined in RFC 3095, requires sequential delivery of packets and misordering is assumed to be taken care of prior to compression/decompression. RFC 3545 [6] also known as ECRTTP (Enhanced Compressed RTP) relaxes this requirement. We next describe these two standards in detail.

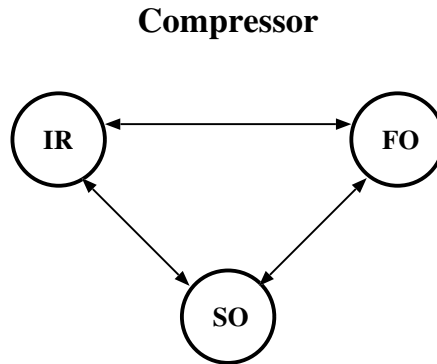


Figure 2.7. State transition among compressor states

2.4.1 RObust Header Compression (ROHC)–RFC 3095

ROHC utilizes the well-known principles of header compression along with some innovative techniques. Central to the framework is the idea of compression *profiles*. ROHC defines four compression *profiles*: RTP/UDP/IP, UDP/IP, ESP/IP and one uncompressed profile. These profiles designate a mapping function between the sequence number SN and predictable fields. Thus knowledge of SN along with the unpredictable fields leads to successful decompression. Protocols such as UDP do not have a SN . In this case ROHC appends an “external” SN to these packets. Protocols that are not supported by ROHC utilize the uncompressed profile. When the channel contains multiple streams, a CONTEXT identifier (CID) is used to distinguish between those streams. Thus, a packet containing a particular CID is compressed/decompressed according to a CONTEXT identifier.

In conjunction with compression *profiles*, ROHC also introduced the concept of states and modes.

The operation of the compressor and/or decompressor is defined by a finite state machine. The states specify what compression/decompression processes take place, and thus determine the instantaneous compression efficiency of ROHC. The compressor state

depends upon variation in header fields of successive packets. The decompressor state relies on the compressor state as well the channel conditions.

The three compressor states are:

1. Initialization and Refresh (IR)
2. First-Order (FO)
3. Second-Order (SO)

The states and their relationship are shown in Figure 2.7. At the start of a session the compressor always starts in IR state. All the necessary information is conveyed to build a reliable CONTEXT at the decompressor. When the compressor is confident enough about the CONTEXT at the decompressor, it transits to FO state. In FO state irregularities in the header fields are efficiently communicated. This leads to a well defined CONTEXT at the decompressor. The RTP *SN* and the changing fields are transmitted in a compressed format and transmitted along with other relevant information in a compressed packet. Finally, when in the SO state the compressor only needs to transmit the RTP *SN*. Thus it can be seen that compression efficiency increases as the compressor ascends to higher states. In the SO state the compressed header can be as small as one byte. Variations in header fields causes the compressor to transit back to lower states.

The decompressor also operates in three different modes (see Figure 2.8):

1. No Context (NC)
2. Static Context (SC)
3. Full Context (FC)

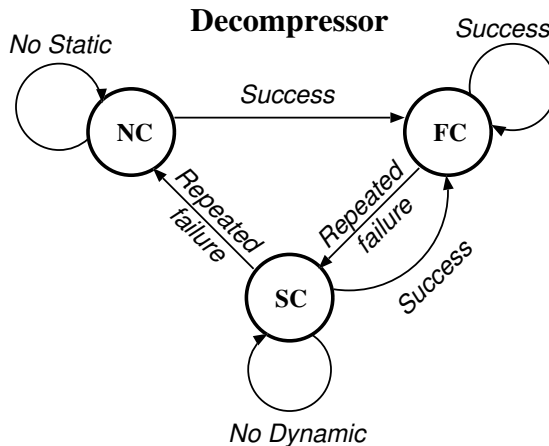


Figure 2.8. State transition among decompressor states

The decompressor lacks any information about the static and dynamic parts of a header when in NC state. Thus the decompressor can only handle uncompressed packets and is reminiscent of IR state of the compressor. After it (decompressor) receives information about the static fields, the decompressor transits to the SC state. In this state the decompressor has information about the static fields but still not sure about the dynamic fields. When the dynamic fields are available, the decompressor moves to the FC state. Now the decompressor has the entire CONTEXT so as to decompress any packet. Decompression failures means that the decompressor descends to lower states. When in FC state, a decompression failure results in transition to SC. Repeated failures finally results in the NC state. The decompressor can ascend back to a higher state as soon as enough information is available to do so.

The ability of ROHC to operate under different link requirements is reflected in its *modes*. Figure 2.9 shows the mode transition depending upon the feedback information received. Note that ROHC *always* starts on the U-mode and graduates to other modes depending upon channel conditions and source characteristics. We describe these three modes in brief; a detail description can be found in RFC 3095 [7].

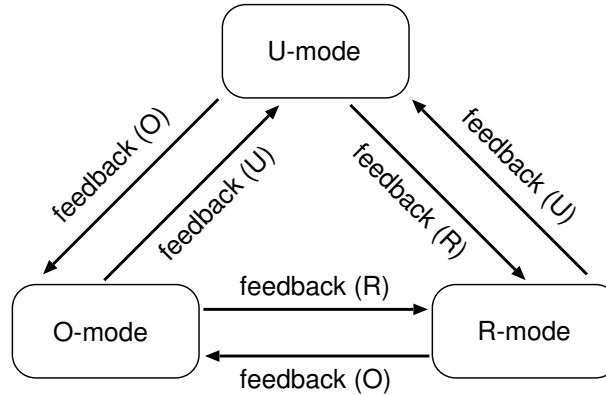


Figure 2.9. Mode transition in ROHC

Uni-Directional mode–(U-mode): ROHC always starts its operation from this mode. In absence of feedback ROHC continues to work in this mode only. Initially uncompressed packets are sent multiple times. Only with enough confidence in decompressor, the compressor transits to higher states. The compressor fall backs to the IR and FO state periodically to ensure correct decompression at the decompressor. Since the compressor mostly operates in the IR or FO state, U-mode has the lowest compression efficiency among all modes.

Bidirectional Optimistic mode–(O-mode): In the bidirectional mode of ROHC we have two classifications, Optimistic O-mode and Reliable R-mode. In the O-mode, the decompressor keeps the compressor informed about the CONTEXT at its side. Whenever decompression is erroneous, a NACK is transmitted. The compressor responds by sending update information to re-synchronize the decompressor CONTEXT. The decompressor can optionally transmit ACKs corresponding to successful update of CONTEXT. This facilitates transition of compressor to the FC state achieving compression efficiency. The updates from the compressor are protected by a 3-bit CRC to prevent incorrect updation of CONTEXT. Still there is chance that this CRC might fail. The R-mode uses a stronger 7-bit CRC and therefore is more *reliable*. The key advantage of operating in O-mode is that the feedback channel is used sparsely. Also high compression is achieved although

with some sacrifice in reliability.

Bidirectional Reliable mode—(R-mode): A 7-bit CRC is transmitted along with the updates to ensure reliability. In contrast to the O-mode, when in the R-mode the compressor can only transit to higher states (SC or FC) when a ACK is received. This ACK is to be conveyed by the decompressor whenever it receives the update correctly. The error-recovery mechanism, i.e. transmission of NACK when CONTEXT is incorrect, is similar to the O-mode. Thus reliability is achieved at the expense of compression efficiency and frequent feedback requests.

2.4.1.1 WLSB encoding of compressed header fields

In addition to states and modes, ROHC also defines an innovative technique to encode compress header fields. This is known as Window-based Least Significant Bit (WLSB) encoding. The basic idea behind WLSB is as follows: instead of encoding the difference with respect to the last header, one can encode the difference with respect to a “neighborhood” of the past few headers. If the neighborhood is such that it is uniquely identified by *any* of the past few headers, then one can decode (decompress) despite the loss of one or more headers, 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.

$$r = \max(|v - v_{max}|, |v - v_{min}|) \quad (2.1)$$

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

$$k = \lceil (\log_2(2r + 1)) \rceil \quad (2.2)$$

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.

2.4.2 Enhanced Compressed RTP (ECRTP)–RFC 3545

RFC 3545 [6] improves upon CRTP (RFC 2508) header compression by adding features to deal with large delays and high error rates. One advantage of ECRTP over ROHC is that ROHC assumes misordering to be a non-issue. ECRTP on other hand accepts packets in any order and processes them sequentially. ECRTP proposes sending updates 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.

2.5 Summary

In this chapter we reviewed key concepts behind header compression. Although correlation among successive packet headers can reduce the packet overhead, error propagation is still a problem. We discussed some existing schemes that attempt to improve performance of header compression schemes. The IETF adopted some of these methods and developed common standards (RFCs). ROHC (RFC 3095) and ECRTP (RFC 3545)

are two such prominent standards by IETF. In the coming chapters we will discuss our proposed schemes for both: uni-directional and bidirectional links.

CHAPTER 3

UNI-DIRECTIONAL LINKS

This chapter introduces the concept of coding packet headers. Error propagation can severely degrade system performance, but this can be countered through proper application of error correcting codes and recovery of lost packet headers. In this chapter we discuss design techniques based on Reed-Solomon and convolutional codes. We analyze the performance of uncoded header compression on uni-directional links for i.i.d. and correlated channel. Also performance of the Reed-Solomon coded system is analyzed as well as its average delay calculated. Finally we conclude by presenting another coded system based on convolutional codes and delay-limited interleaving.

3.1 Channel Models

We consider two channel models for simulations: an independent and identically distributed (i.i.d.) channel and a two-state Markov channel. In an i.i.d. channel the packets (bits) are lost independent of each other with some probability. A two-state Markov process on the other hand models correlated losses on a channel. Wired channels can be modeled as an i.i.d. process due to fewer burst errors. In contrast due to bursty nature of losses, wireless channels are usually characterized via a Markov process [14].

In packet-switching networks, packets are either received correctly or not received at all. Each packet includes a CRC checksum calculated at the transmitter. At the receiver, the CRC is calculated again and failing which the packet is simply discarded. The upper layers can recognize a missing packet by observing the sequence number SN of

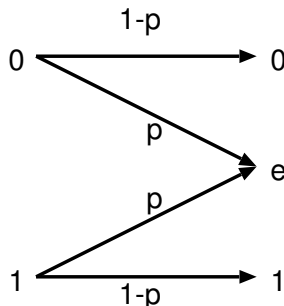


Figure 3.1. A binary erasure channel

received packets. Thus the receiver knows *which* packet is missing; this behavior is akin to an erasure channel.

Figure 3.1 shows a binary erasure channel (BEC). Each bit is received correctly with probability $1 - p$ and erased with probability p . Thus this model is suitable for packet based communications over a wired channel. Internet for example can be modeled as an i.i.d. channel. Studies [15] corroborate this view with an observation that bursty errors are possible but not prominent.

Wireless channels have bursty errors due to interference, multipath fading and noise. A finite-state, finite-order Markov process can capture this nature of wireless channels very well [14, 16]. A first-order, two-state Markov process often adequately characterizes the wireless media [17] (Figure 3.2).

As shown in Figure 3.2, when in state “0” packets are received correctly. The channel can continue to be in state “0” or transit to state “1”. When in state “1”, the packets transmitted over this channel are corrupted and subsequently dropped. Let us denote by P_{ij} the probability of moving to state j from state i . Thus, for example P_{10} is the state transition probability of moving from state “1” to state “0”. Clearly $P_{11} = 1 - P_{10}$. The transition probability can be represented in a matrix form as follows:

$$\mathbf{P} = \begin{bmatrix} P_{00} & P_{01} \\ P_{11} & P_{10} \end{bmatrix} \quad (3.1)$$

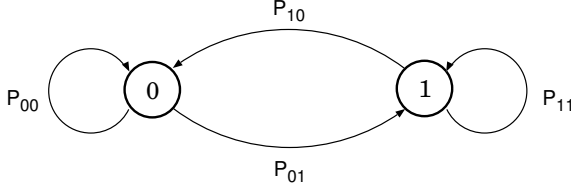


Figure 3.2. A channel modeled on a two-state Markov process

The steady state probability of being in state “0” is $\pi_0 = \frac{P_{10}}{P_{01}+P_{10}}$ and for state “1”, $\pi_1 = \frac{P_{01}}{P_{01}+P_{10}}$. Also the mean sojourn time in a state “0” and “1” is a geometric random variable, $B_0 = \frac{1}{P_{01}}$ and $B_1 = \frac{1}{P_{10}}$ respectively. Note that B_1 denotes the average burst length of errors.

3.2 Performance of Header Compression on Uni-directional Links

Recall that without any feedback in place, a lost packet means that subsequent packets have to be discarded (see Figure 2.3). First we will calculate the packet discard rate for the i.i.d. channel and then the correlated channel.

On an i.i.d. channel each packet is lost independent of other packets. Let p denote the packet loss probability of the i.i.d. channel. An uncompressed packet header is transmitted every (say) α compressed packet headers. Let us consider a window of $\alpha + 1$ such packet (one uncompressed packet header followed by α compressed headers).

If k denotes the number of packets discarded due to the first error event it can be easily seen that k is a random variable according to a *modified* geometric distribution as follows:

$$P(k) = \begin{cases} (1-p)^{(\alpha+1)} & k = 0 \\ p(1-p)^{(\alpha+1-k)} & k = 1, 2, \dots, \alpha + 1 \end{cases}$$

The average number of packets discarded can then be calculated as:

$$\bar{m}_i = \sum_{k=0}^{\alpha+1} k(1-p)^{(\alpha+1-k)}p = (\alpha+1) - \frac{1-p}{p} [1 - (1-p)^{\alpha+1}] \quad (3.2)$$

For a session comprising of T total packets, the overall packet discard rate can be calculated as:

$$\overline{M}_i = \lfloor \frac{T + \alpha}{\alpha + 1} \rfloor \left[(\alpha + 1) - \frac{1 - p}{p} [1 - (1 - p)^{\alpha + 1}] \right] \quad (3.3)$$

As expected the performance can be improved by increasing the refresh rate but compression efficiency suffers.

For the correlated case we have a similar distribution for the number of discarded packets:

$$P(k) = \begin{cases} (1 - P_{01})^{(\alpha + 1)} & k = 0 \\ P_{01}(1 - P_{01})^{(\alpha + 1 - k)} & k = 1, 2, \dots, \alpha + 1 \end{cases}$$

and therefore,

$$\overline{m}_b = (\alpha + 1) - \frac{1 - P_{01}}{P_{01}} [1 - (1 - P_{01})^{\alpha + 1}] \quad (3.4)$$

Similarly,

$$\overline{M}_b = \lfloor \frac{T + \alpha}{\alpha + 1} \rfloor \left[(\alpha + 1) - \frac{1 - P_{01}}{P_{01}} [1 - (1 - P_{01})^{\alpha + 1}] \right] \quad (3.5)$$

Substituting $P_{01} = \frac{\pi_1}{(1 - \pi_1)B_1}$, we get the packet discard rate in terms of burst length and steady state error probability as:

$$\overline{M}_b = \lfloor \frac{T + \alpha}{\alpha + 1} \rfloor \left[(\alpha + 1) - \frac{1 - \frac{\pi_1}{(1 - \pi_1)B_1}}{\frac{\pi_1}{(1 - \pi_1)B_1}} \left[1 - \left(1 - \frac{\pi_1}{(1 - \pi_1)B_1} \right)^{\alpha + 1} \right] \right] \quad (3.6)$$

3.3 Motivation for Coding Packet Headers

Error propagation on uni-directional links is prevented by periodic refreshing. Sending uncompressed headers at regular intervals restores synchronization between the two CONTEXT whenever packets are lost. As noted in Section 3.2, increasing the refresh rate provides better performance in terms of packet discard rate, but compression efficiency shrinks. The refresh method clearly leaves much to be desired.

A compressed packet usually consists of 2–5 bytes. Thus there are two features of the header compression problem that give rise to burst of errors. First, each packet header,

even when compressed, consists of multiple bytes, thus the loss of each compressed header will result in 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 bursty nature of losses and by excellent burst-error correcting capability of Reed-Solomon (RS) codes, a technique is proposed to mitigate error propagation on uni-directional links. In a similar vein, a reduced-complexity solution using an interleaved convolutional code is proposed. Coding alone cannot achieve performance. As we will see, distribution of parity symbols needs careful attention to enhance the effectiveness of these error correcting codes.

3.4 Reed-Solomon Codes

Forward error correction (FEC) comes in two flavors: block codes and convolutional codes [18, 19]. Block codes offer excellent error-correcting capability at the expense of higher decoding complexity. On the other hand, convolutional codes offer reduced computational complexity.

A systematic RS encoder accepts K data symbols and generates $N - K$ parity symbols [18]. 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

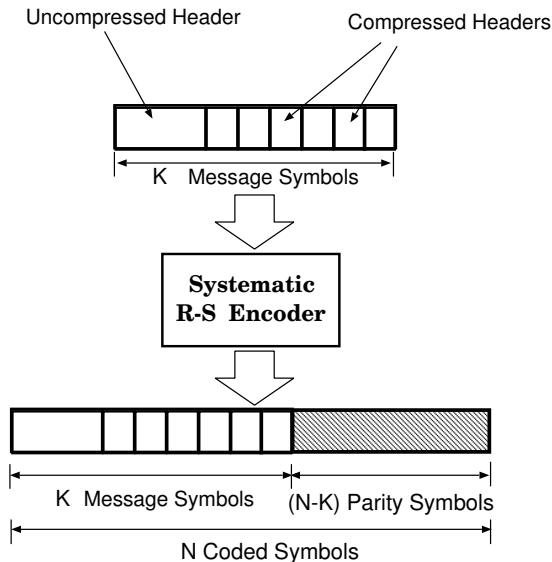


Figure 3.3. Generation of parity symbols from packet headers

group of symbols (see Figure 3.3). 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. This makes it possible to operate over links where nodes lack decoding capability. In this case when the node receives a coded block it can simply choose to discard the parity bits.

In order to achieve a configuration of parity bits that will attain 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

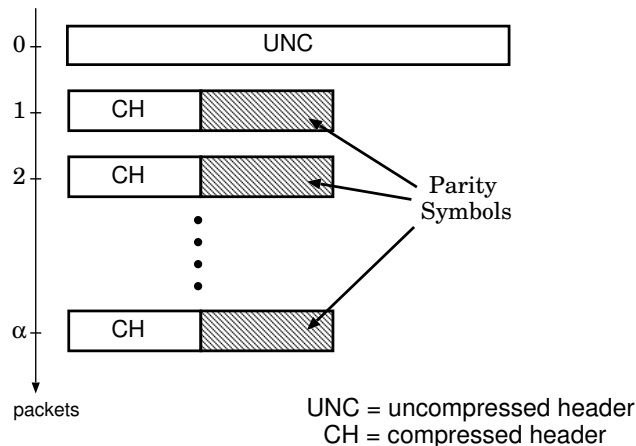


Figure 3.4. Equal distribution of parity symbols among compressed packet headers. There are α compressed packet headers

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 3.4) 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 3.3. The parity symbols so generated are distributed *equally* among the α packets with compressed headers shown in Figure 3.4. The number of parity symbols generated is $x\alpha$, where x denotes the number of parity symbols per transmitted compressed header.

3.4.1 Performance analysis of RS coded system

First we calculate the performance of the coded system when errors are i.i.d.. 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; $K_p = \frac{u+x\alpha}{s_a}$

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

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:

$$P_{RS}^i = \sum_{i=K_p}^{\alpha} \binom{\alpha}{i} p^{\alpha-i} (1-p)^i \quad (3.7)$$

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

$$\bar{m}_{cod} = \alpha p P_{RS}^i + \bar{m}_i (1 - P_{RS}^i) \quad (3.8)$$

where \bar{m}_i is the packet discard rate of the uncoded system to be substituted from Equation 3.2.

For the correlated case we follow the approach given in [20] which is based on the concept of error-free intervals or gaps. Let a gap length ν be the event that after an error there are $\nu - 1$ consecutive error free symbols and an error occurs again. Also $g(\nu) = P(0^{\nu-1}1|1)$ then gives the distribution of ν . Let $G(\nu)$ denote the gap distribution function which gives the probability of a gap length greater than $\nu - 1$, $G(\nu) = P(0^{\nu-1}|1)$. From Figure 3.2 we can derive the following:

$$g(\nu) = \begin{cases} 1 - P_{10}, & \nu = 1 \\ P_{10} (1 - P_{01})^{\nu-2} P_{01}, & \nu > 1 \end{cases} \quad (3.9)$$

$$G(\nu) = \begin{cases} 1, & \nu = 1 \\ P_{10} (1 - P_{01})^{\nu-2}, & \nu > 1 \end{cases} \quad (3.10)$$

Let $L(m, n)$ denote the probability of having $m - 1$ errors within the next $n - 1$ packets following an error. Using recurrence relations it can be shown that:

$$R(m, n) = \begin{cases} G(n), & m = 1 \\ \sum_{\nu=1}^{n-m+1} g(\nu) L(m-1, n-\nu), & 2 \leq m \leq n \end{cases} \quad (3.11)$$

Then the probability of losing m packets within a block of n packets is,

$$P(m, n) = \sum_{\nu=1}^{n-m+1} \pi_1 L(m, n - \nu + 1) \quad , \quad 1 \leq m \leq n \quad (3.12)$$

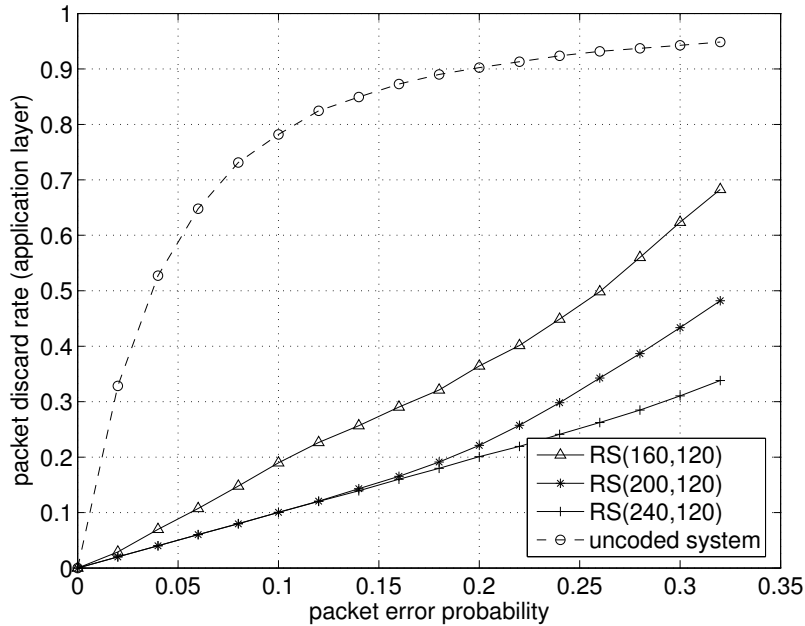


Figure 3.5. Performance of RS codes over the uni-directional i.i.d. channel

The probability of successfully decoding a codeword on the correlated channel can then be calculated as,

$$P_{RS}^b = 1 - \sum_{m=K_p}^{\alpha} P(m, n) \quad (3.13)$$

Similarly, the average packet discard rate of the coded system for the correlated case is given as,

$$\bar{m}_{bcod} = \alpha \pi_1 P_{RS}^b + \bar{m}_b (1 - P_{RS}^b) \quad (3.14)$$

where \bar{m}_b can be substituted from Equation 3.4.

3.4.2 Simulation results

As shown in Figure 3.3 an uncompressed packet header with u bytes and α compressed headers with c bytes is presented to a systematic RS encoder. The parity symbols so generated are distributed (see Figure 3.4) *equally* among the compressed packet headers. For the simulations we consider $u = 40$ and $c = 2$ bytes. Each compressed header

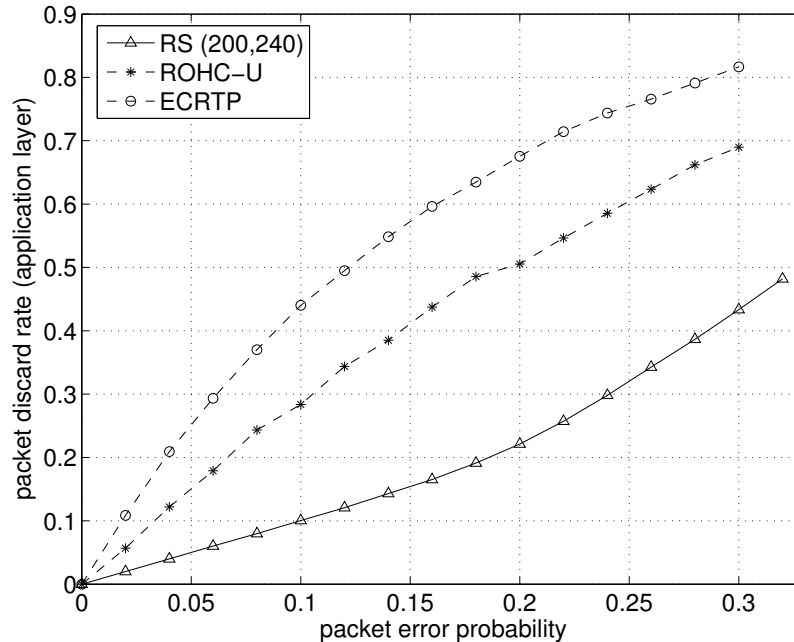


Figure 3.6. Performance comparison of RS codes with ECRTP and ROHC on i.i.d. uni-directional channel

carries x parity symbols, thus there are αx parity symbols. This leads to $K = u + \alpha x$ and $N = u + \alpha(c + x)$ as the parameters for the RS code. We simulate the RS coded system for $x = 1, 2$ and 3 which corresponds to $(160, 120)$, $(200, 120)$ and $(240, 120)$ RS codes.

Figure 3.5 compares the performance of the various coded system with the uncoded case. As expected the performance improves as the code rate is decreased. Note that the packet discard rate is calculated at the application layer. Thus, even though we recover compressed headers the payload is still missing. So any packet header compression system is bounded by the performance of a “perfect” header compression, represented by a line with unit slope. We also compare performance of RS(200,240) code with ECRTP and ROHC in the uni-directional mode. Simulation results in Figure 3.6 illustrate the benefits of coding.

For the correlated case we consider $B_L = 5, 10$. Figures 3.7 and 3.8 compare the

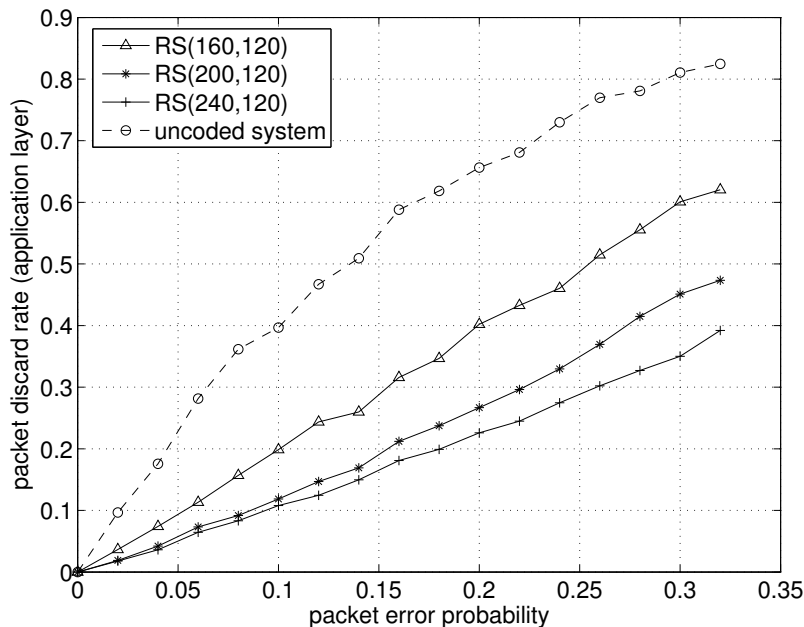


Figure 3.7. Performance of RS codes over the uni-directional bursty channel with $B_L = 5$ performance of the coded system with the uncoded one. Again the proposed scheme performs well over all channel conditions. Similar performance is observed by the proposed system when compared with ECRTP and ROHC in uni-directional mode. The burst length, B_L is set to 5 (Figure 3.9). The 95% confidence interval for all simulations lies within $\pm 0.1\%$ of the results shown.

The performance of uncoded header compression seems better on the correlated channel than on the i.i.d. channel (compare Figures 3.5 and 3.7). This happens because the errors in the correlated case are more likely to be bursty. In other words, assuming the first packet arrives without error, in the uncorrelated case, it is more likely that a longer sequence of packets arrive without error, before the first error is observed. Due to the nature of the header compression problem, it is the location of the first error, rather than average error rate of the channel, that determines the performance of the system. Therefore, between two channels *with the same average error rate*, the one that is bursty will show better performance.

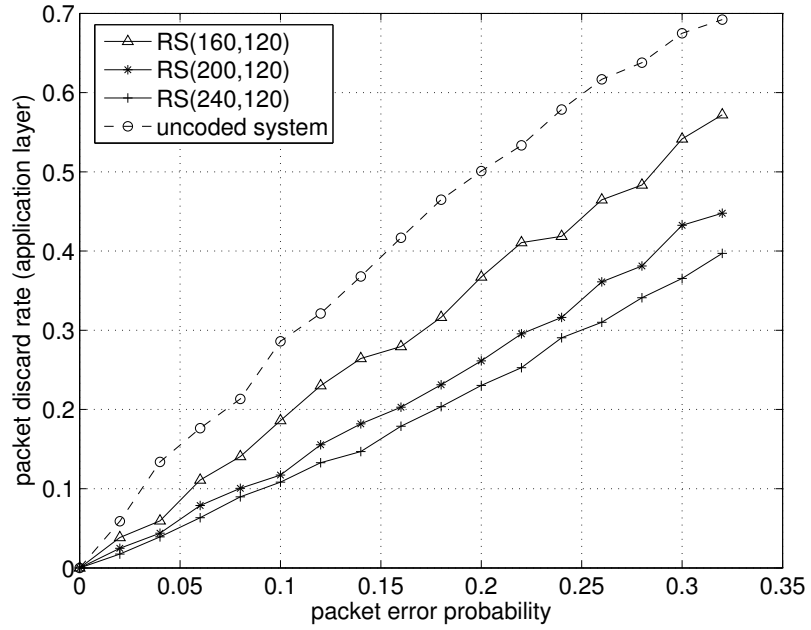


Figure 3.8. Performance of RS codes over the uni-directional bursty channel with $B_L = 10$

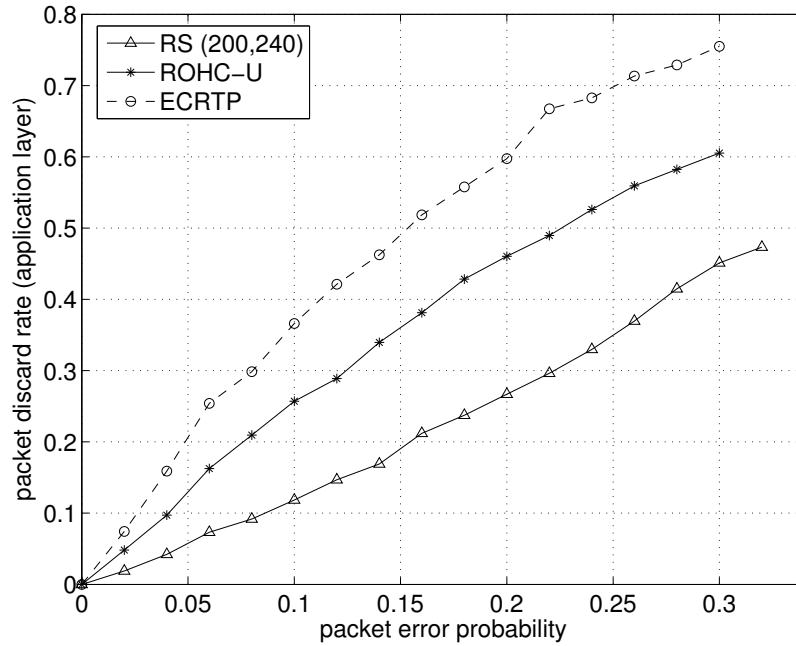


Figure 3.9. Performance of RS codes over the uni-directional bursty channel with $B_L = 10$

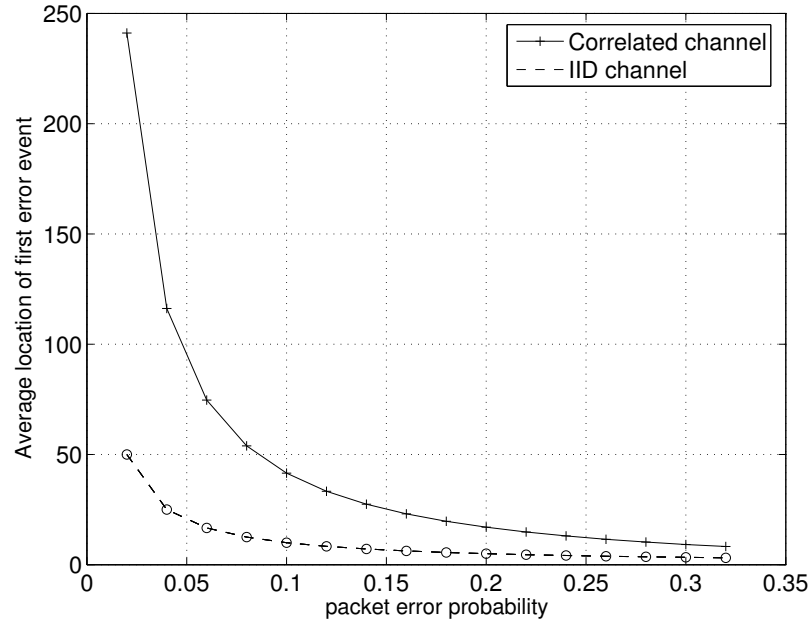


Figure 3.10. Comparison of average location of first error event.

3.5 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 a prescribed number of systematic plus parity bits have been correctly received.

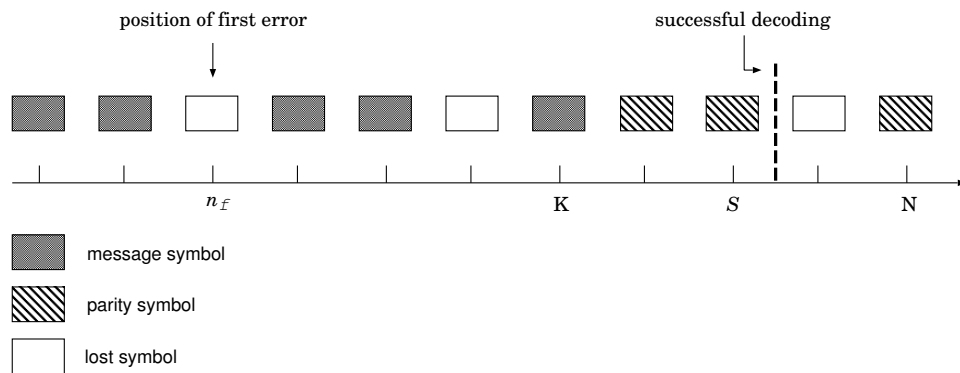


Figure 3.11. Illustration of decoding delay in RS coded systems.

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. This situation is illustrated in Figure 3.11. It can be seen that we must wait for decoding until exactly K correct symbols have been received (in Figure 3.11 we need to wait for two more symbols), 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:

$$\bar{D} = \frac{1}{N} \sum_{n=n_f}^S (S - n_f) \quad (3.15)$$

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

$$\begin{aligned} E[\bar{D}] &= E \left[\frac{(K + \Delta - n_f)^2}{2N} \right] \\ &\geq E \left[\frac{K(K + \Delta - n_f)}{2N} \right] \end{aligned} \quad (3.16)$$

where p is the probability of packet loss and r is the code rate. The position of first error n_f is a geometric random variable with mean $1/p$. Also, to successfully decode a codeword, we need K correct symbols, thus the ‘‘time to K successes,’’ which is equivalent to $K + \Delta$, follows a negative binomial distribution, therefore

$$P[K + \Delta = m] = \binom{n-1}{K-1} (1-p)^K p^{m-K} \quad m = K, K+1, \dots$$

or

$$P[\Delta = \delta] = \binom{n-1}{K-1} (1-p)^K p^\delta \quad \delta = 0, 1, \dots$$

The mean of Δ can be calculated to be $E[\Delta] = \frac{K}{1-p} - K$. Therefore the bound on the delay is:

$$E[\bar{D}] \geq \frac{r}{2} \left(\frac{Nr}{1-p} - \frac{1}{p} \right) \quad (3.17)$$

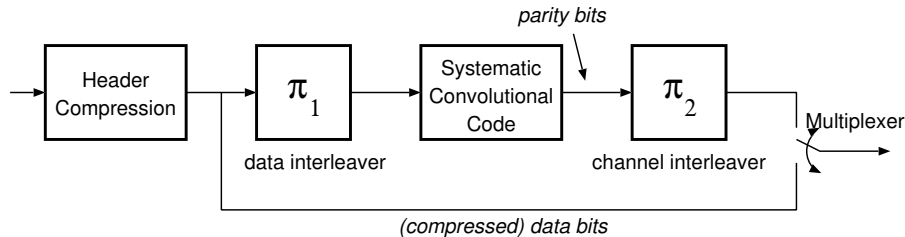


Figure 3.12. Interleaving and convolutional encoding

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 via the Berlekamp-Massey algorithm or Peterson-Gorenstein-Zierler algorithm [18]. Convolutional codes have complexity that increases linearly with codeword length using the optimum yet computationally efficient Viterbi algorithm [21]. Decoding wise, convolutional codes offer an interesting trade-off. Increasing the constraint length of the encoder improves the performance but complexity also increases. This trade-off is not observed in RS codes. Thus as per the system requirements performance can be compromised if lower complexity is desired.

Convolutional codes perform well when errors in the codeword are random. 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 3.12 at the transmitter a total of $u + ac$ bits are fed to the

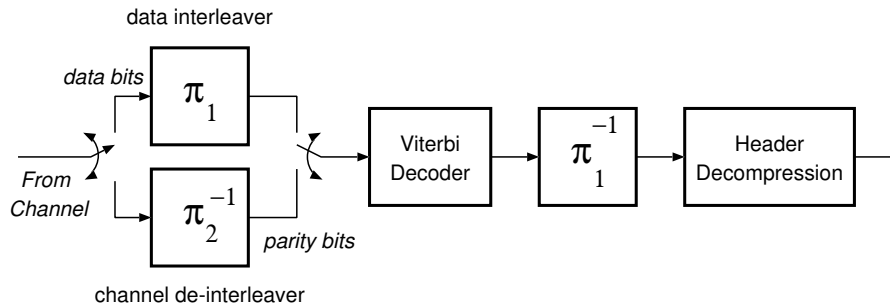


Figure 3.13. Deinterleaving and decoding convolutional codes via Viterbi algorithm

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. This is shown in Figure 3.4. Interleaving data bits before encoding generates parity bits belonging to data bits that are not close to each other. Correct reception of one such parity bit can contribute in reconstruction of many packets.

The parity bits are equally distributed among the α compressed packets instead of transporting them in one single packet. Thus loss of a particular packet would not lead to decoding failure. The decoding operation is shown in Figure 3.13 at the receiver side.

3.5.1 Delay-limited interleaver design

We use two interleavers in this scheme to augment the performance of convolutional codes. Unfortunately interleaving always incurs delay. In general the maximum delay can be as large as the codeword length. 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 distance apart in the output stream [22]. The delay of an interleaver is defined as:

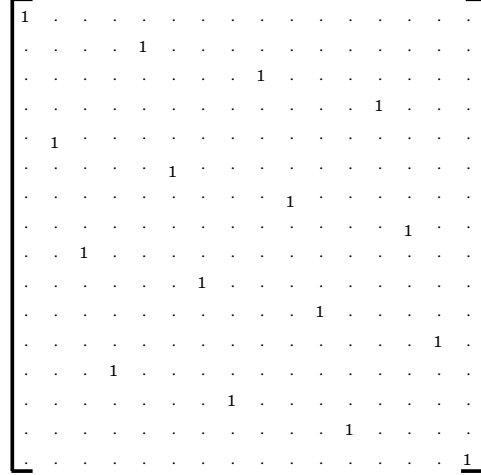


Figure 3.14. Block interleaver with $s = 3$ and $\mathcal{D} = 9$ bits.

$$\delta = \max_i (\pi(i) - i) \quad (3.18)$$

A classic block interleaver fills a matrix row wise with input bits and reads them out column wise. If M and N are number of rows and columns respectively of a block interleaver matrix, then the delay of this interleaver is given as [22]:

$$\mathcal{D} = (M - 1) \times (N - 1) \quad (3.19)$$

We denote by J_{ij} as the i^{th} row and j^{th} column of interleaver matrix \mathbf{J} . The period P of the interleaver is the block length. \mathbf{J} has dimension $P \times P$ where P is the period of the interleaver. Figure 3.14 shows the interleaver matrix of a block interleaver with $M = N = 4$ (and therefore $P = M \times N = 16$). Each element of the matrix is either 1 or 0, with a 1 in a column denoting the position of the input bit at the output of the interleaver. To clearly show the structure of the interleaver matrices in Figure 3.14 and 3.15 the zeros are not shown. As can be seen from the figures, each row and column has only one 1. Thus initially a 1 is placed in the first row and first column and then in the second row

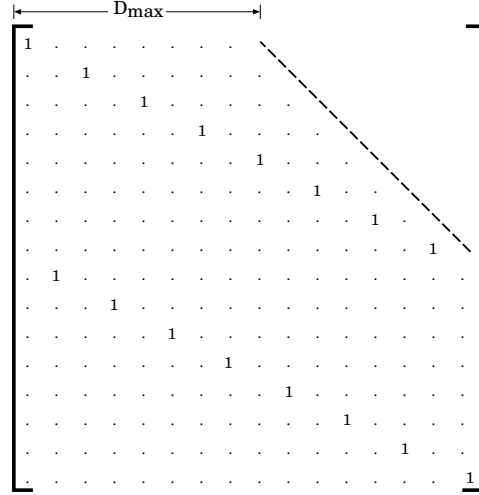


Figure 3.15. Delay constrained interleaver with $s = 1$ and $\mathcal{D}_{max} = 7$ bits.

and $1 + (s + 1)^{th}$ column and so on. Note that the interleaver in Figure 3.14 has delay $\mathcal{D} = 9$ bits with $S = M - 1 = 3$.

Let \mathcal{D}_{max} be the maximum allowable delay of the system (in bits). From Equation 3.18 it can be seen that if an interleaver needs to be designed with \mathcal{D}_{max} in view, then

$$\pi(i) \leq \mathcal{D}_{max} + i \quad (3.20)$$

The delay-limited interleaver design algorithm starts with an initial guess for the s -parameter. A necessary and sufficient condition for an interleaver with spread s and period P to exist is that $P \geq s^2$ [23] 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 3.20. As in the case of a block interleaver, we start by placing a 1 in the first row and first column. Note that a row represents the output bit of the interleaver. The next 1 is placed in the

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 with 1s in a similar fashion.

The resulting interleaver matrix will have block interleaver structure but some of the permutation indices are not allowed. This is indicated by a dashed-line in Figure 3.15. Thus the interleaver matrix is filled as in the case of a block interleaver but will have delay less than \mathcal{D}_{max} . Figure 3.15 shows the interleaver matrix with spread $s = 1$ and maximum delay $\mathcal{D}_{max} = 7$ bits. The resulting matrix will have a trapezoidal appearance due to the delay constraint.

The following pseudo-code describes the steps involved in the algorithm:

START

1. Set $s \rightarrow s = s_{max} = \lfloor \sqrt{P} \rfloor$
2. Initialize

$$\mathbf{J} = 0$$

$$i = j = 0$$

$$J_{ij} = 1$$
3. Increment the row value $\rightarrow i = i + 1$
4. If $i \geq s$ goto 8
5. Set $j = i + s + 1$
6. If $j > \mathcal{D}_{max} + i$, then $s = s - 1$ and goto 2
7. Set $J_{ij} = 1$ and goto 3
8. If $j > \mathcal{D}_{max} + i \rightarrow$ goto 9
9. Wrap around \rightarrow select $j = \min_{unoccupied} column$

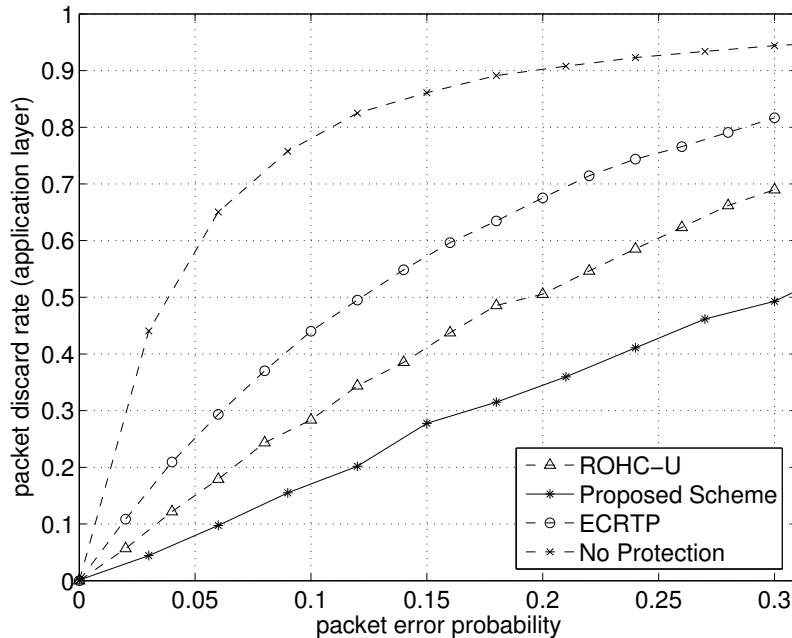


Figure 3.16. Performance of the interleaved convolutional code for the uni-directional i.i.d. channel

10. if $j = \{ \}$, then $s = s - 1$ and goto 2
11. Set $J_{ij} = 1$ and goto 3

END

3.5.2 Simulation results

Performance of the interleaved convolutional system is evaluated with similar settings as in Section 3.4.2. Thus, $u = 40$ and $c = 2$ with $\alpha = 40$. The proposed scheme is compared again with ECRTTP and ROHC in the uni-directional mode. The interleaver used in these experiments is an unconstrained one i.e., with $D_{max} = u + \alpha c = 960$ bits. Decreasing the delay budget will weaken the spreading factor of the interleaver and also the performance of the system. For the uni-directional case we will only focus on achieving maximum efficiency in terms of packet discard rate.

Figure 3.16 shows effectiveness of the proposed scheme over the i.i.d. channel.

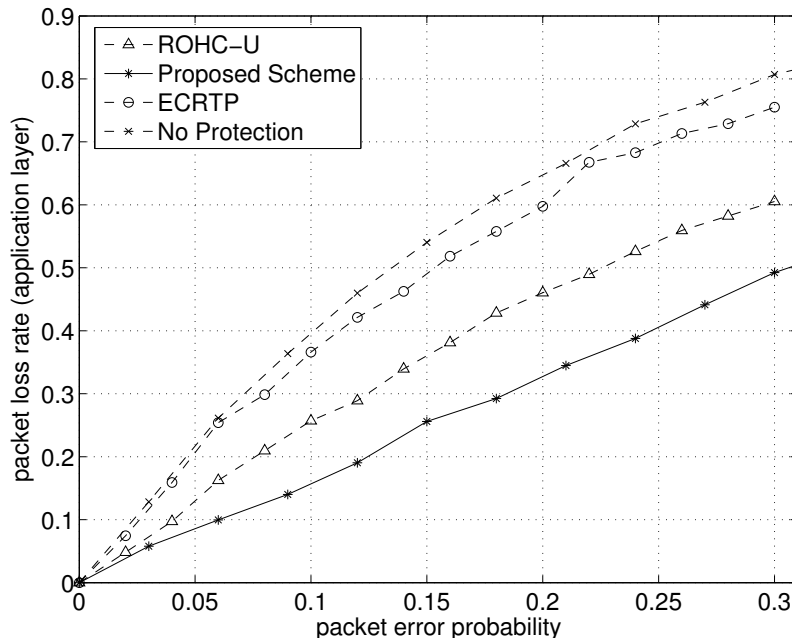


Figure 3.17. Performance of the interleaved convolutional code for the uni-directional correlated channel

Table 3.1. Generator functions for various rate 1/2 convolutional codes from [2]

constraint length K	g^0	g^1
3	[101]	[111]
4	[001 101]	[001 111]
5	[010 011]	[011 101]
6	[101 011]	[111 101]

Compared to ECRTP and ROHC operating in the U-mode, the proposed scheme performs very well over all channel conditions. The RSC encoder given in [2] with rate 1/2, $K = 6$ and generator matrix $g^0 = [101011]$ and $g^1 = [111101]$ is used. Using the similar RSC encoder, results for the correlated channel also show identical performance (see Figure 3.17). The simulations results lie within $\pm 1.44\%$ of the reported results for a 95% confidence interval.

Increasing the constraint length of a convolutional code gives better error correcting

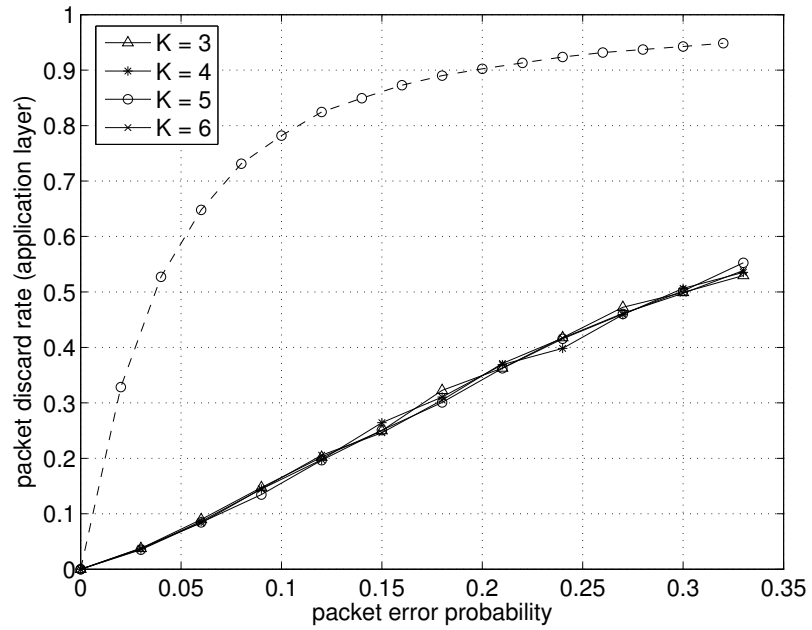


Figure 3.18. Performance of various constraint length convolutional encoders for the uni-directional i.i.d. channel

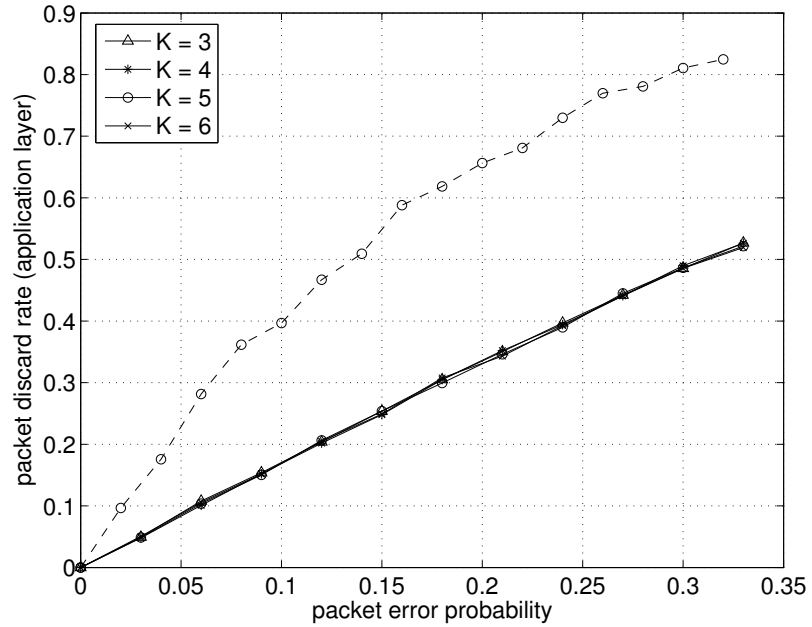


Figure 3.19. Performance of various constraint length convolutional encoders for the uni-directional correlated channel

capability. We utilize different constraint length encoders shown in Table 3.1 [2]. It can be observed from Figures 3.18 and 3.19 that the performance is insensitive to the constraint length. This reinforces the bursty nature of header compression systems. A single packet loss means several bits are erased (the corresponding parity bits as well).

3.6 Summary

We introduced the idea of coding packet headers for uni-directional links. Excellent performance gains are achieved both for the i.i.d. as well as for the correlated channel. Due to decoding complexity of RS codes we proposed a reduced-complexity solution in the form of interleaved convolutional codes. To address the issue of delay introduced by interleaving, we presented a delay-limited interleaver design. Simulation results in both the cases (RS codes as well as interleaved convolutional codes) demonstrate the benefits of coding.

CHAPTER 4

BI-DIRECTIONAL LINKS

We propose a new predictive hybrid ARQ technique and compare the performance of the proposed scheme with ROHC-O, the current state-of-the-art in resilient header compression in bi-directional links.

4.1 Hybrid ARQ Techniques

In classic ARQ, reliability is achieved by retransmission of lost data [18, 24]. FEC on the other hand adds redundancy to facilitate recovery of the lost information. ARQ techniques offer reliability when channel conditions deteriorate, but throughput suffers due to frequent retransmissions. In comparison, FEC techniques offer a constant throughput but reliability depends upon the code rate. The lower the code rate, the better the performance, but throughput is lower as well.

Proper combination of these two techniques can overcome their respective drawbacks. Hybrid ARQ involves an encoded forward link for error correction and detection and a feedback link for possible retransmission. At the transmitter parity bits are added to the data block to detect and correct errors. In case the receiver is not able to correct these errors, the data block is transmitted again. For each received data block the receiver either sends an ACK (data block is received or decoded successfully) or a NACK (data block is undecodable). The transmitter responds to a NACK by retransmitting the information. Based on the retransmission strategy hybrid ARQ can be classified as either type-I or type-II hybrid ARQ.

Type-I hybrid ARQ involves an error correction and detection code over each data block. When errors are detected in a coded block (error detection) the receiver attempts to correct them (error correction). Depending upon the channel conditions and error pattern errors are either correctable or not. The receiver accepts the coded block if the error pattern is correctable and sends an ACK. If not, the receiver rejects the coded block and transmits a NACK. The transmitter then retransmits the coded frame. This continues until success.

Type-I hybrid ARQ is most suitable when the channel stays stationary i.e., constant error rates. Thus a code can be designed intelligently to overcome errors over the channel. If the channel quality fluctuates wildly one is wasting parity bits when the channel is good, while sending many retransmissions when the channel is bad.

4.2 Predictive Hybrid ARQ

In existing hybrid ARQ schemes [18] the decoder waits until a codeword is fully received. It then tries to decode the codeword and decides if retransmission is necessary. In our search for partial ARQ methods we found [25], 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 than 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

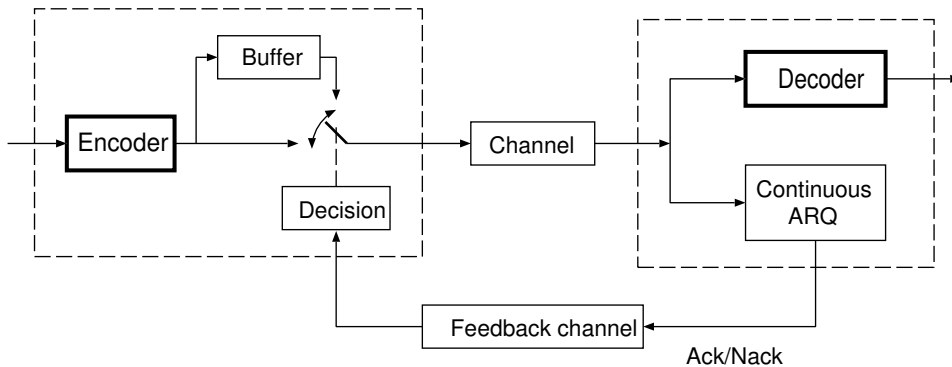


Figure 4.1. Predictive Hybrid ARQ – the blocks entitled “encoder” and “decoder” are delineated in Figure 3.12 and 3.13 respectively

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 4.1. We call our proposed scheme predictive hybrid ARQ. The encoder and decoder block are shown in Figure 3.12 and 3.12 respectively. 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 3.12 at the transmitter side where each packet is denoted by P_i , $i \in \{1, 2, \dots, 10\}$. 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 simulate and compare the performance of the predictive hybrid ARQ system with ROHC-O mode. Recall that when in O-mode, the decompressor only sends NACK following an error. This results in spare usage of the feedback channel. We do not compare ECRTP since simulations in [26] already indicate superior performance of ROHC-O compared to ECRTP.

4.3 Simulation Results

As shown in Figure 3.12, $u = 40$ bytes of an uncompressed header along with $\alpha = 40$ compressed headers are supplied to a RSC convolutional encoder. Each compressed header consists of $c = 2$ bytes giving a total of 480 data bits. Prior to encoding the data bits are fed to an interleaver designed with the algorithm in Chapter 3. Using this delay-limited algorithm we design interleavers with delay $D_{max} = 100, 200, 400$ and 900 bits. The parity bits are interleaved using the same interleaver. Parity bits they are equally distributed among the compressed header packet. We simulate and compare performance of the proposed predictive hybrid ARQ system against ROHC-O over the i.i.d. channel as well as on the correlated channel (see Section 3.1).

We use the RSC encoder given in [2] 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. This assumption is reasonable since header compression usually functions on a per hop basis, introducing negligible propagation delay. Figure 4.2 shows the performance of the proposed system. Compared with ROHC-O, our method performs well under all channel conditions. Par-

ticularly the delay profile remains consistent even when the channel deteriorates, whereas delay of ROHC-O increases linearly. Only at low error rates, due to interleaving, higher delay is incurred. Overall the delay profile of the proposed scheme is better than ROHC-O.

The proposed method also performs significantly better in terms of throughput. Figure 4.2 shows that the throughput improves as D_{max} increases. This can be attributed to the fact that as D_{max} increases, the interleaver algorithm is less constrained, thus bits can be placed as far as possible giving a higher spreading factor s . A higher value of s provides better randomization of errors, therefore improving the decoding capability of the decoder and resulting in fewer retransmissions.

Figure 4.3 shows simulation results for the correlated channel with burst length, $B_L = 5$. Note that the delay profile is flat similar to the i.i.d. case. In fact even at lower error rate the delay offered by the proposed scheme is better than ROHC-O. The throughput also exhibits similar characteristic except at higher packet loss rates. This is due to the fact that at higher error rates we have longer burst of errors. Decoding failure leads to more retransmissions and hence throughput suffers. For most channel conditions the proposed scheme performs very well compared to ROHC-O. For the throughput and delay, the 95% confidence interval for all simulations is within $\pm 1.3\%$ and $\pm 0.61\%$ of the reported results.

We also analyze the variation in throughput and delay of the proposed system with different constraint length. In particular we consider $K = 4, 6$ from Table 3.1. As expected we note that as K increases so does the performance of the system. Also note that for $D_{max} = 900$ bits, increasing the constraint length from $K = 3$ to 4 and 6 improves the throughput drastically (see Figure 4.4, 4.5, 4.6 and 4.7).

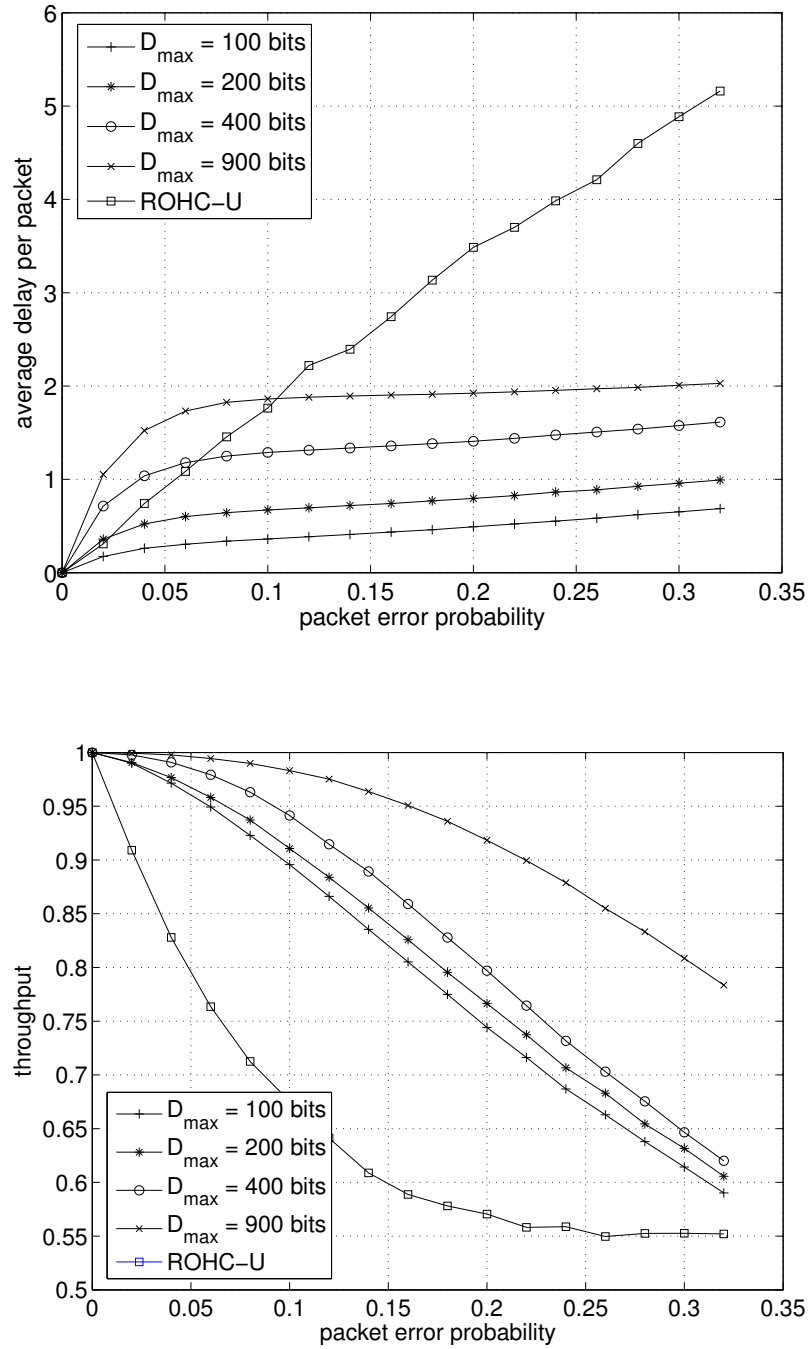


Figure 4.2. Performance of the predictive hybrid ARQ method over an i.i.d. channel with encoder constraint length $K = 3$

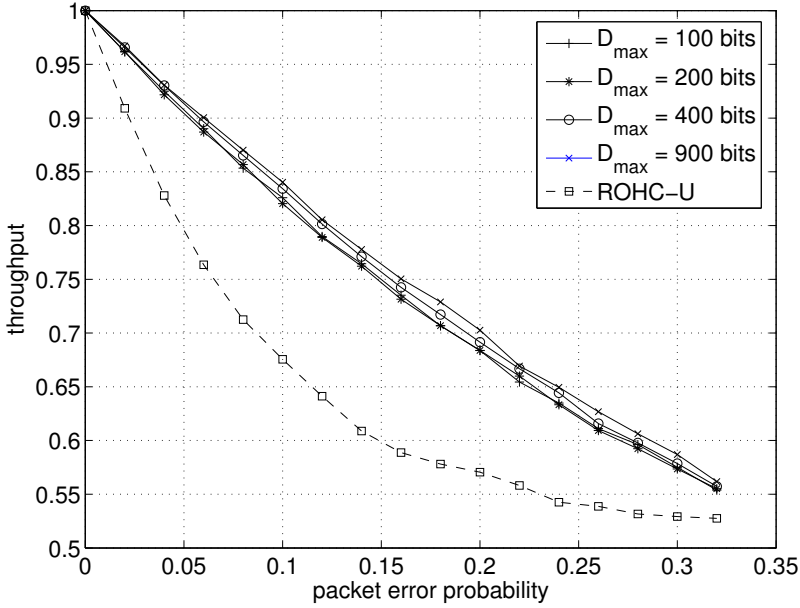
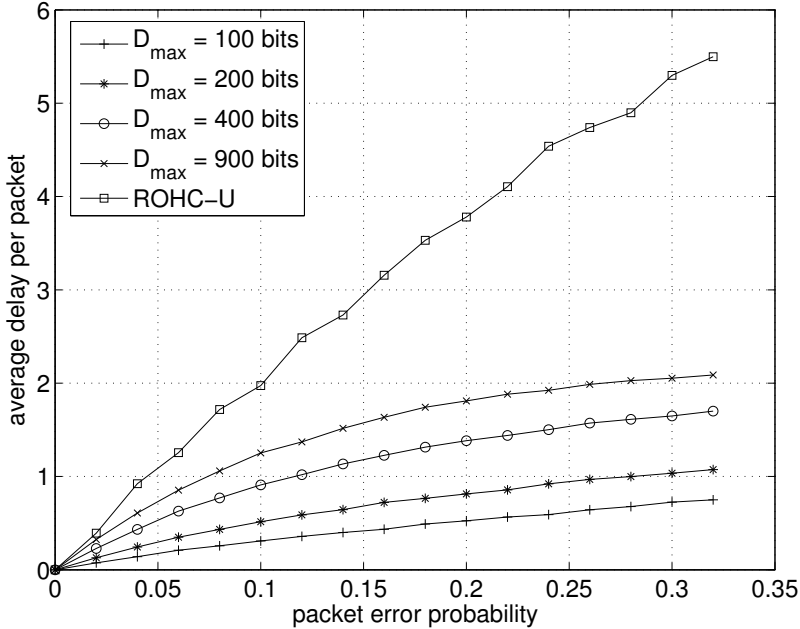


Figure 4.3. Performance of the predictive hybrid ARQ method over a correlated channel ($B_L = 5$) with encoder constraint length $K = 3$

4.4 Summary

In this chapter we present coding techniques in the presence of feedback. We noted that the ARQ mechanism of header compression systems is substantially different from existing hybrid ARQ techniques, because retransmission requests are generated multiple times in a codeword. A predictive hybrid ARQ mechanism is proposed which takes advantage of these multiple retransmissions. Simulation results indicate that the proposed scheme performs well compared to ROHC-O under most channel conditions. It is also observed that the delay profile of the proposed scheme remains consistent under all channel conditions.

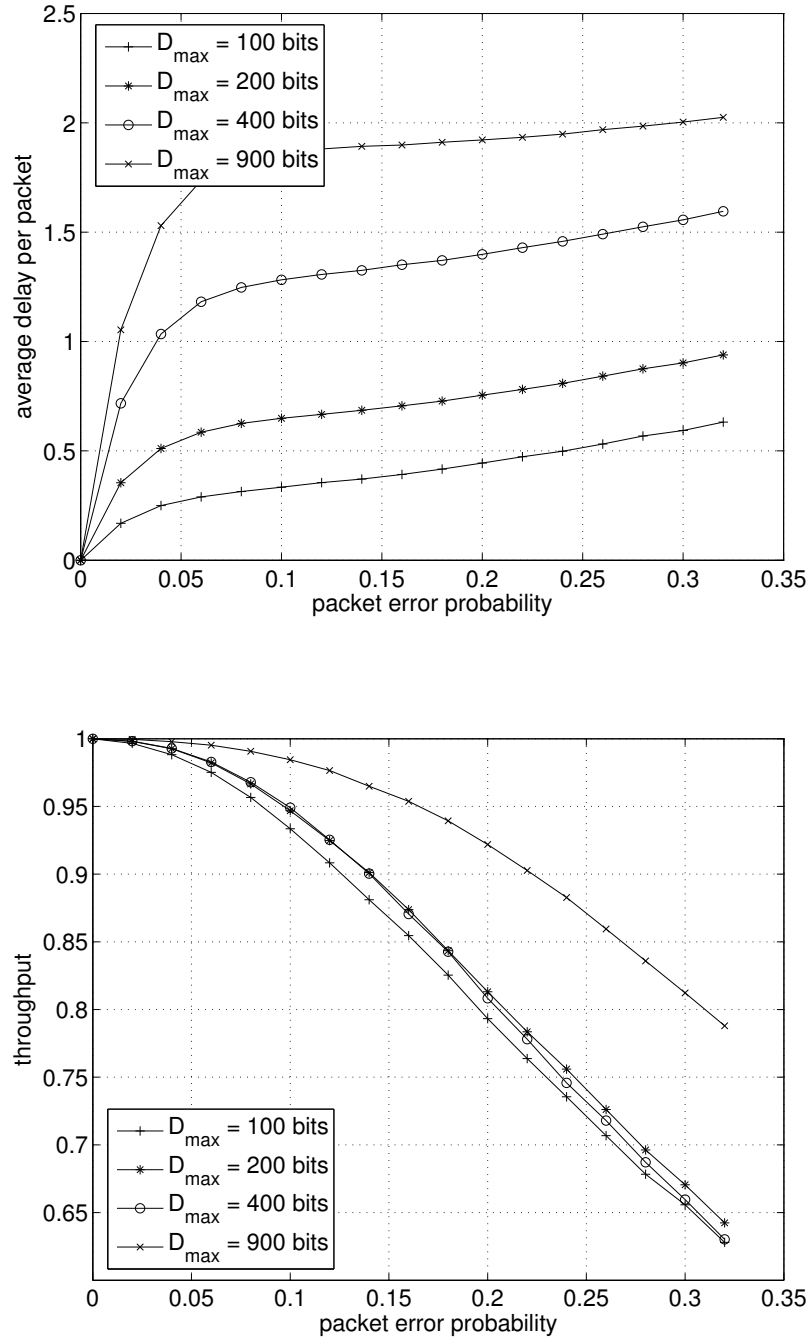


Figure 4.4. Delay and throughput performance of predictive hybrid ARQ over an i.i.d. channel. Constraint length, $K = 4$

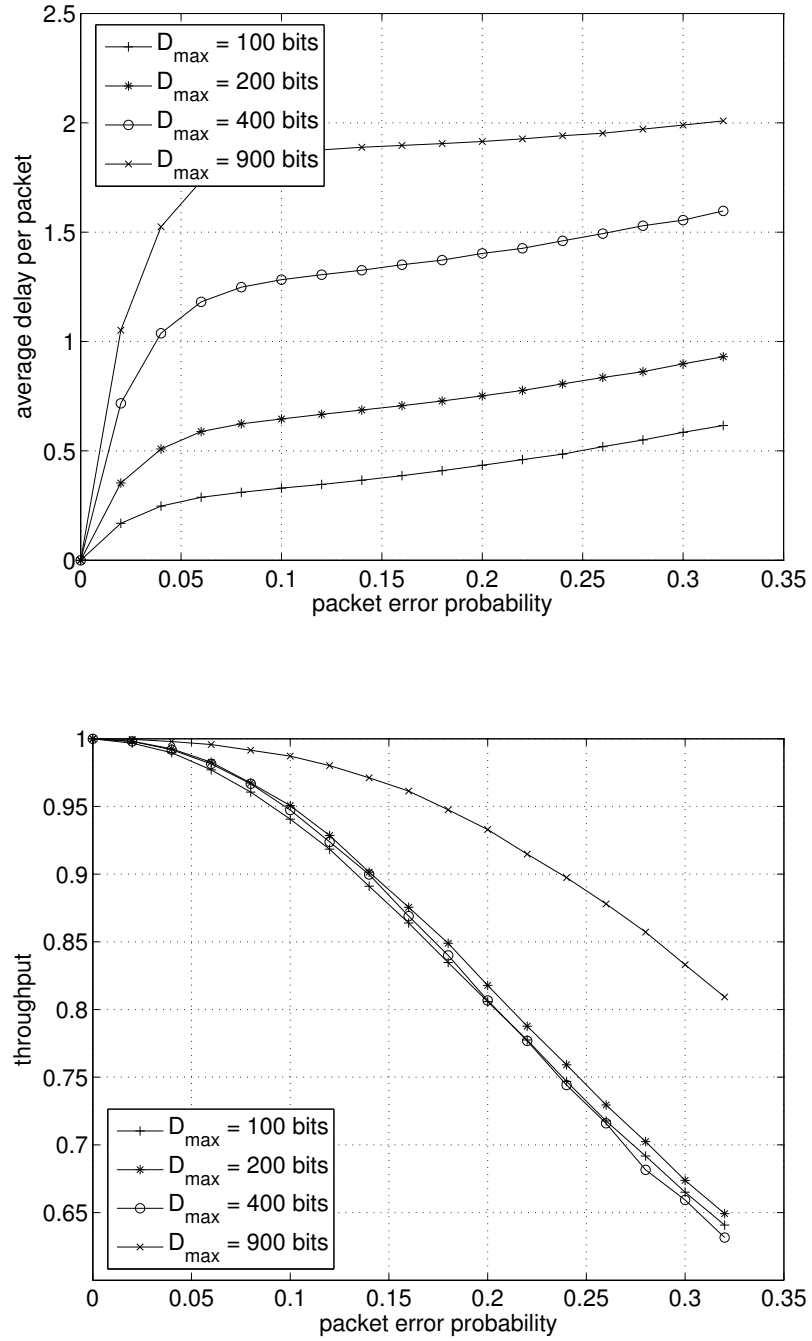


Figure 4.5. Delay and throughput performance of predictive hybrid ARQ over an i.i.d. channel. Constraint length, $K = 6$

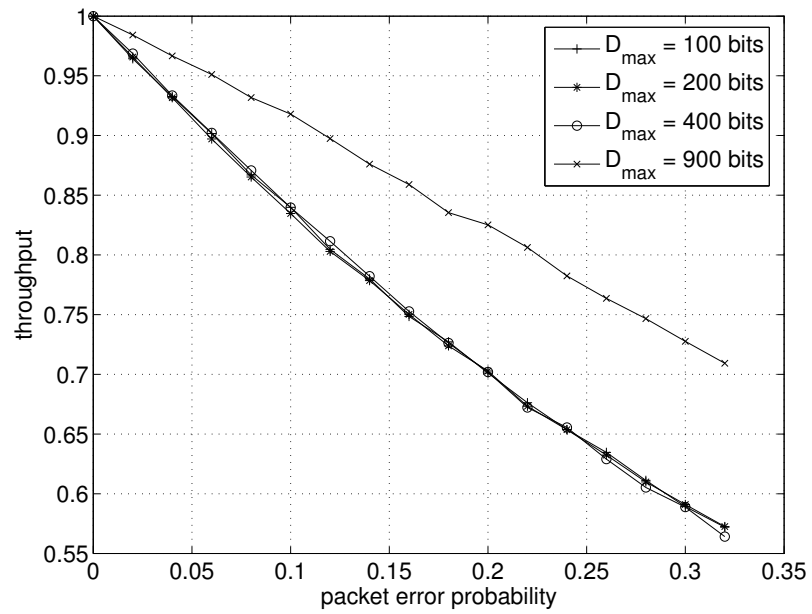
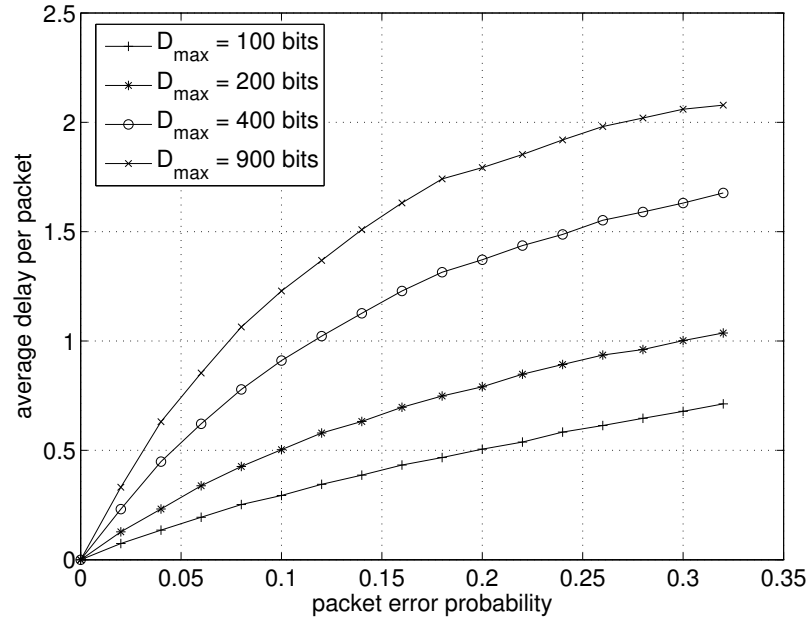


Figure 4.6. Delay and throughput performance of predictive hybrid ARQ over a correlated channel ($B_L = 5$). Constraint length, $K = 4$

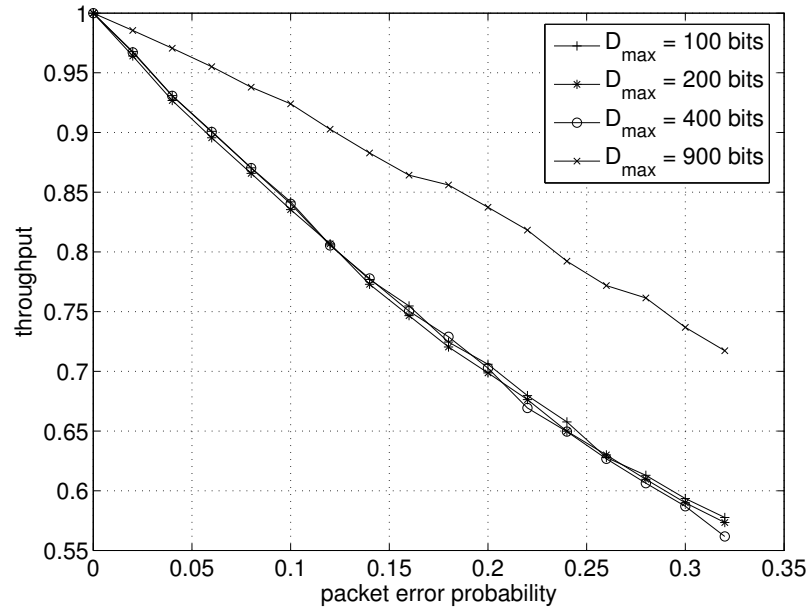
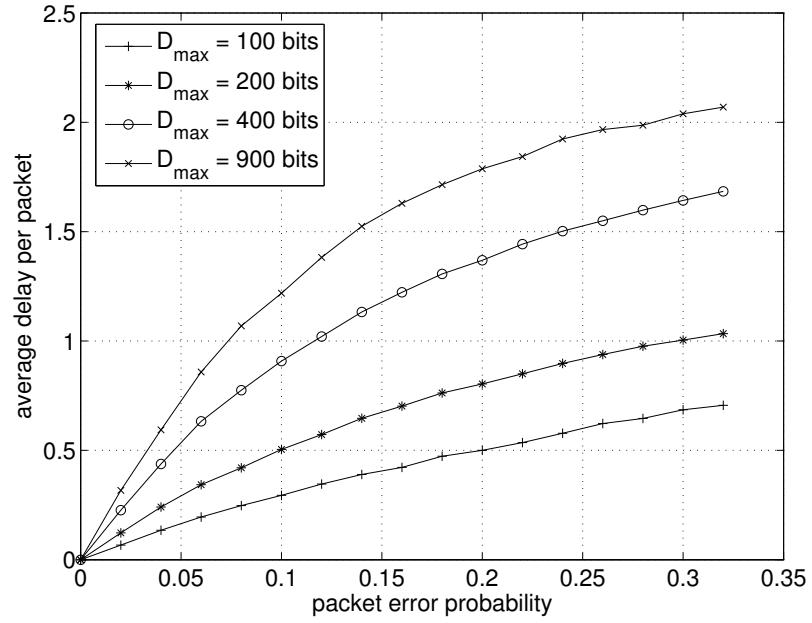


Figure 4.7. Delay and throughput performance of predictive hybrid ARQ over a correlated channel ($B_L = 5$). Constraint length, $K = 6$

CHAPTER 5

CONCLUSION AND FUTURE WORK

This thesis presents coding techniques applicable to packet header compression. We present design techniques based on Reed-Solomon and convolutional codes. One key advantage of our methods is their minimal interference with the existing protocol stack, yet providing excellent performance. Also proposed is an algorithm to design delay-limited interleavers to be used with convolutional codes. Over uni-directional links, these techniques provide superior performance compared to existing techniques. In the bi-directional case we note that the retransmission mechanism of header compression schemes is substantially different from existing hybrid ARQ schemes. Multiple retransmissions are generated within a codeword. Therefore, we introduce a new predictive hybrid ARQ technique for coded header compression schemes. This hybrid ARQ method is shown to outperform existing schemes under same conditions.

The header compression schemes presented in this thesis are for non-TCP packets. Van Jacobson's algorithm (RFC 1144) can be employed for compressing TCP headers but performance degrades over lossy channels [27, 28]. The IETF ROHC group is working towards a robust header compression solution for TCP protocols. The work is currently underway and expected to materialize in early 2006 [29]. Future work includes evaluating the performance of coded TCP header compression against the final draft on ROHC-TCP.

Another extension of this work is to apply other advanced error correcting codes. Turbo codes [30, 22] can be a good start in this direction. Turbo codes are known to achieve excellent performance with moderate complexity. Also, low-density parity

check codes (LDPC) [31] have become popular due to their excellent erasure correcting capabilities. Recent investigations [32] indicate that LDPC codes can come within 0.0045 dB of the Shannon limit. Another excellent class of erasure correcting codes are the Raptor codes [33]. Interestingly Raptor codes have excellent erasure correcting capability as well linear complexity encoding and decoding algorithms. Future work includes designing robust header compression with these advanced codes

Designing coded header compression schemes via these advanced error correcting codes will require special care. Minimal interference with the protocol stack, interoperability, and lower complexity should be the design objectives.

BIBLIOGRAPHY

- [1] NLANR, “Wan packet size distribution: www.nlanr.net/na/learn/packetsizes.html,” HTTP Link, January 1997.
- [2] P. Frenger, P. Orten, and T. Ottosson, “Convolutional codes with optimum distance spectrum,” *IEEE Communications Letters*, vol. 3, pp. 317–319, November 1999.
- [3] A. Tannenbaum, *Computer Networks*, Prentice Hall PTR, 2003.
- [4] CAIDA, “Packet length distributions: www.caida.org/analysis/aix/,” HTTP Link, August 2004.
- [5] 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>.
- [6] 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>.
- [7] 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>.
- [8] 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)*, June 1997, vol. 2, pp. 1025–1029.

- [9] 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.
- [10] A. Calveras, M. Arnau, and J. Paradells, “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)*, July/August. Orlando, FL. 1999, vol. 4, pp. 39–46.
- [11] 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.
- [12] M. Rossi, A. Giovanardi, M. Zorzi, and G. Mazzini, “TCP/IP header compression: Proposal and performance investigation on a WCDMA air interface,” in *Proc. of the 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Sept 2001, vol. 1, pp. A 78–A 82.
- [13] M. Degermak, M. Engan, B. Nordgren, and S. Pink, “Low loss TCP/IP header compression for wireless networks,” in *Proceedings of ACM Mobicom 96*, New York, NY, October 1997, pp. 375–396.
- [14] L. N. Kanal and A. R. K. Sastry, “Models for channels with memory and their applications to error control,” *Proceeding of IEEE*, vol. 66, no. 7, pp. 724–744, July 1978.
- [15] M. S. Borella, S. Uludag, G. Brewster, and D. Swider, “Internet packet loss: Measurement and implications for end-to-end QoS,” in *International Conference on Parallel Processing Workshops*, Jan 1998, pp. 3–12.

- [16] H. S. Wang and N. Moayeri, "Finite-state Markov channel - a useful model for radio communication channels," *IEEE Transaction on Vehicular Technology*, vol. 43, no. 1, pp. 163–171, Feb. 1995.
- [17] M. Zorzi, R. R. Rao, and L. B. Milstein, "On the accuracy of a first-order Markov model for data block transmission on fading channels," in *Proc. IEEE ICUPC*, Tokyo, Japan, November 1995, pp. 211–215.
- [18] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Englewood Cliffs, NJ: Prentice Hall, 1995.
- [19] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*, Pearson–Prentice-Hall., 2004.
- [20] 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.
- [21] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. 13, pp. 260–269, April 1967.
- [22] C. Heegard and S. B. Wicker, *Turbo Coding*, Kluwer International Series in Engineering and Computer Science, 1999.
- [23] V. Tarokh and B. Hochwald, "Existence and construction of block interleavers," in *Proc. of IEEE International Conference on Communications (ICC)*, April 2002, vol. 25, pp. 1855–1857.
- [24] 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.

- [25] H. S. Cheng, G. Fairhurst, and N. Samaraweera, "Efficient partial retransmission ARQ strategy with error detection codes by feedback channel," in *IEE Proceedings - Communications*, October 2000, vol. 147, pp. 263–268.
- [26] A. Yiannakoulis and E. Kuehn, "Evaluation of header compression schemes for IP-based wireless access system's," *IEEE Wireless Communications [see also IEEE Personal Communications]*, vol. 12, pp. 68–74, February 2005.
- [27] Y. Wu, L. Sun, J. Zheng, K. Huang, and Y. Liao, "Channel state dependent robust TCP/IP header compression for 3G wireless networks," in *IEEE International Conference on Performance, Computing, and Communications*, Phoenix, AZ, April 2004, pp. 141–145.
- [28] R. Wang, "An experimental study of TCP/IP's Van Jacobson header compression behavior in lossy space environment," in *Proc. IEEE Vehicular Technology Conference*, Los Angeles, CA, September 2004, pp. 4046 – 4050.
- [29] G. Pelletier, L. Jonsson, K. Sandlund, and M. West, "RObust Header Compression (ROHC):A Profile for TCP/IP (ROHC-TCP)," IETF Network Working Group, July 2005.
- [30] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 409 –428, March 1996.
- [31] R. G. Gallager, "Low Density Parity Check Codes," MIT Press, 1963.
- [32] S. Chung, G. D. Forney Jr.and T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Communications Letters*, , no. 1, pp. 1449–1454, November 2001.

- [33] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, “Efficient erasure correcting codes,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 569–584, February 2001.

VITA

Vijay Suryavanshi was born in Mumbai (formerly Bombay), India on April 1, 1979. After finishing high school in 1997 from K. C. College, Mumbai, he pursued his career in Electronics and Telecommunications engineering. In June 2001 he received the Bachelor of Engineering degree from University of Mumbai, Mumbai, India. In Fall 2002 he was admitted to the Master's program at the University of Texas at Dallas, Richardson, USA. Currently he is completing his Master's in Electrical Engineering- Telecommunications at the same institution. From May 2004 until May 2005 he worked as a Research Intern at Nokia Research Center, Irving, USA.

Permanent address: 200/24, Kazi Syed St,
Mandvi Koliwada,
Mumbai-400003,
Maharashtra India.

This thesis was typeset with L^AT_EX[†] by the author.

[†]L^AT_EX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's T_EX Program.