WIRELESS DATA TRANSMISSION: ERROR CHARACTERIZATION AND PERFORMANCE ANALYSIS

by

CHANGLI JIAO

DISSERTATION

Submitted to the Graduate School
of Wayne State University,
Detroit, Michigan
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

2002

MAJOR: COMPUTER ENGINEERING

Approved by:

_____

Adviser Date

_____

_____

_____

Dedication

To My Parents and Christina

Acknowledgements

First of all, I thank my advisor, Professor Loren Schwiebert, for his guidance and assistance throughout my graduate study. Without his vision and insight, this dissertation would never be started. Without his research results, invaluable advice, and numerical suggestions, the contribution of this dissertation would become only much less substantial. Without his enthusiasm and encouragement, this dissertation would never be completed. Dr. Schwiebert has taught me not only computer networking but also the way to succeed. His advice beyond the research scope has been and will be highly appreciated.

I also thank Professor Vipin Chaudhary, Professor Cheng-Zhong Xu, and Professor Sherali Zeadally, for serving on my committee. Their comments and suggestions helped improving the correctness and clarity of this dissertation.

Specially, I thank Professor George Yin, for his many helpful comments on the mathematical models and analysis used in this dissertation.

My graduate study at Wayne State University was both fruitful and pleasant. This was partially made possible by other graduate students with whom I had the opportunity to work together. I thank Dr. Sumit Roy, Dr. Jialin Ju, Vikas Sinha, Shi Jia, Padmanabham Menon, Niranjan Ghate, Manoj Kona, for their help and sharing experiences, both in computer science and graduate life. I thank Manish Shah, Dan Hu, Yibing Nie, Zhihui Zhao for having classes together and staying late in the Lab. together. I thank the *Networking Wireless Sensors Lab.* members, Manish Kochhal, Ayad Salhieh, Kamran Jamshaid, Manpreet Bajwa, Yong Xi, for having group meeting together, discussing papers and research issues together.

Especially, I want to express my appreciation to my family member for their love,

moral support, and encouragement. It was impossible for me to concentrate on the dissertation if my in-laws did not devote to taking care of everyday life. My husband, Dr. Bin Xu, is my source of infinite inspiration. I also thank him for participating in mathematical analysis of this dissertation. Finally, I thank my daughter, Christina, for her unconditional love, for the joy and excitement she brings to my life. There was a lot time that I should have spent and I would like to spend with her; but I could not. I thank in advance for her future understanding.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this dissertation, the accuracy of data delivered from the transport layer is discussed. For wireless channels, packet error statistics and connection performance of different error control mechanisms are also studied. First, the performance of the internet checksum is analyzed. Second, the problems with existing header compression algorithms are pointed out and an adaptive algorithm is proposed. Third, how to model packet error statistics in wireless channels with bursty errors is studied. And finally, the performance analysis of error correction mechanisms in wireless channels is also covered.

1.1 1's complement checksum

When data are transmitted over a medium, it is possible that some bits will get corrupted. So we need error detection technology to prevent errors from getting passed to users. In Chapter 2, we analyze the error detection performance of 1's complement checksum, the arithmetic used for detecting errors in the transport layer.

When TCP/IP datagrams are sent over the Ethernet, the link layer uses Cyclic Redundancy Check (CRC) to detect errors. The Point-to-Point Protocol (PPP) also uses CRC. In most wireless data networks, CRC is also the way to detect errors in the link layer [52]. In the transport layer, both TCP and UDP use internet checksum [46] [45] to do error detection on headers and data, although the checksum is optional with UDP.

CRCs are based on polynomial arithmetic, base 2. It has long been known that CRCs are very powerful for error detection. CRC-32 is the most commonly used

CRC in the TCP/IP suite. CRC-32 can detect all bursty errors with length less than 32 bits and all 2-bit errors less than 2048 bits apart. For all other error patterns, the chance of not detecting is only 1 in 2^{32}. Internet checksum [41] [50] is a 16-bit 1's complement sum of the header and data. Compared with CRC using the same number of bits, it is weaker in error detection, whereas less calculation is needed.

Given CRCs' strong error detection ability, one could argue that the TCP or UDP checksum is not necessary. Practically, this was tried in the 1980s [58]. For some Network File Server (NFS) implementations, UDP checksum was disabled based on this argument. But this idea resulted in file corruption and ultimately was discarded as a bad idea. Recently, Stone and Partridge have shown for the Internet today, there are a wide variety of error sources that cannot be detected by link-level CRCs [58]. Defective hardware, buggy software, and problems in both end-systems and routers can all cause errors that will pass link-level error detection. Newly emerging software and hardware are especially vulnerable to these problems. In essence, any errors introduced at protocol layers above the link layer will not be detected by the link-layer CRC. Changes should be made to eliminate these error sources. But, before old error sources are cleaned up, or, after new sources are introduced, the best solution is to detect these errors. For the TCP and UDP protocols, this task can be done only by the internet checksum. The property of internet checksum, i.e., 1's complement checksum, should be analyzed in order to understand the error detection ability and to guide protocol design and improvement.

The error masking probability of only 2's complement checksum has been computed [9] before. For 1's complement checksum, the probability has not been computed fully. Desaki, et al. [14] partially solved this problem. They calculated the error passing probability of 2 or 3 errors. However, the method provided is hard to extend

to more bits errors. And, extending the method might lead to wrong results since some error passing patterns are ignored. In addition, the upper and lower bounds on the total probability provided in [14] are not very tight.

In this dissertation, we give a brief review of the previous work in Section 2.1. We also correct the small oversight made by Desaki, et al. [14] and explain the group of error patterns not considered before. In Section 2.2.2 and Section 2.2.3, we prove several properties on error values of one word, which are used to provide an exact calculation of the error passing probability for a block. The calculation formula is given in Section 2.2.4. Since too many calculations are needed to get the probability for a big block, in Section 2.2.5, we prove tighter upper and lower bounds of the error masking probability. We also use some special properties of the formula to ease the computation in Section 2.2.6. After achieving the exact value of the error masking probability, finally we present some discussion in Section 2.3. We summarize some features of the probability. We also find out *traditional belief on error detection performance of internet checksum is not correct*. Tradition thinking is that only around $\frac{1}{2^{16}-1}$ of all the corrupted packets can pass the internet checksum. However, the exact error masking probability shows that internet checksum may perform much worse than this. Having achieved the error masking probability, we can also compare the error detection capability of 1's complement checksum and 2's complement checksum. The results show that 1's complement checksum performs better under normal communication situations. In summary, the contributions of Chapter 2 include:

- A formula to calculate the exact error masking probability of 1's complement checksum.
- Tighter upper and lower bounds as well as a simplified way to perform the computation.

- The traditional belief on the error masking probability of the internet checksum, i.e., around $\frac{1}{2^{16}-1}$, is not correct.

- Under normal communication situations, 1's complement checksum performs better than 2's complement checksum.

Part of the work on 1's complement checksum was presented as a poster in [30] and published in *The 10th International Conference on Computer Communications and Networks* [29]. The results we have achieved on 1's complement checksum have also been submitted to *Information Processing Letters* [31].

1.2 Header Compression

Header compression has long been an important issue in the TCP/IP protocol suite. We discuss the following issues in Chapter 3:

- The problems of header compression algorithms under non-pathological packet reordering and errors avoiding link layer detection are discussed.

- A new header compression algorithm, adaptive to channel BER to approach a good tradeoff between throughput and compression ratio.

After a header compression algorithm is applied, most packets are expected to contain shorter headers to be transferred over the communication medium. In this way, compressed packets can usually reach the destination after a shorter period of time compared to communication without compression. From the standpoint of the communication medium, the channel usage will be improved, which means more packets can be carried during a given period of time.

TCP/IP packets with compressed headers normally traverse only a single link. Because the router needs the whole header to perform routing, making the router

understand the compressed header would involve too much change or may not even be feasible. So there is a compressor on one side of the communication channel and on the other side, a decompressor. However, the compression and decompression process could introduce errors. Usually, headers are compressed based on some information known by the decompressor. Correctly decompressed packets refresh the information on the decompressor side, which allows the subsequent packets to be recovered. A corrupted packet obviously cannot be decompressed and has to be dropped. And this packet's failure to be decompressed could cause the decompressor to become desynchronized. Thus, subsequent packets received by the decompressor may also be dropped even though they are transmitted correctly. This is called *error propagation*, which means the packet dropping is caused by errors in previous packets instead of this packet itself.

Continuously discarding packets on the forward path, due to error propagation, results in ACKs not being sent. Similarly, error propagation on the reverse path will prevent the TCP sender from receiving the ACKs. In the TCP sender, ACKs are used to indicate network conditions. The sender will even transmit packets at a lower speed if the gap between ACKs is too big. This will have two side results. First, the efficiency of compression is lower, or perhaps even more bits need to be sent compared to transmitting without compression. Second, the correct information is received later, maybe even later than transmission without compression. This could be harmful for real-time or delay-sensitive applications. In some real-time situations, applications can still use the damaged information, in which case the tradeoff between delay and having to use erroneous information needs full consideration. These are the bad influences when the decompressor has to drop packets. If, for some reason, the decompressor cannot detect that a packet was corrupted, worse results may emerge.

Wrong information will be passed to higher layers. Even though there may exist other error detection mechanisms at higher protocol layers, the correctness should be questioned, since it is applied on an error base.

There are a number of proposals for TCP/IP header compression [28] [12] [17] [19]. At the same time, recent research shows that packet reordering is a common phenomenon in modern computer networks [6], and that there are various sources of errors that are passed to the transport layer [58]. In Chapter 3, we will analyze the performance of header compression proposals under these two phenomenon. We will also present a new algorithm for header compression over wireless networks that achieves a good tradeoff between throughput and compression ratio.

The work on header compression in wireless channels was published in *The 26th Annual IEEE Conference on Local Computer Networks* [32].

1.3 Packet Error Statistics in Wireless Channels

More and more modern communication applications are using wireless channels since mobility provides a lot of convenience to communication end users. However, wireless links usually have to face many more errors than their wired counterparts. This is basically the direct result of wireless channel characteristics such as multi-path, transmission interference, and signal fading [49]. The Gilbert-Elliott model [24] [18] is a well-known model for the bit errors in wireless channels. For mobile radio channels, especially those used for typical transmissions from a base station to mobile users in an urban environment, Rayleigh fading is the widely accepted model. And the Gilbert-Elliott model has been proved to model this kind of fading channel with sufficient accuracy [61] [65].

When packets are transferred over the channel, each packet is composed of a

block of consecutive bits. Considering packet transmission without error correction technologies such as Forward Error Correction (FEC) and interleaving, the packet is correct only when all the continuous bits are transmitted correctly. Knowledge of the packet error statistics is more important to discuss the communication performance. Even though the common model of the bit level error statistics is a first-order Markov process, the models for packet level statistics differ a lot as shown in previous work [64] [43] [35]. Theoretically, a Markov chain has been proved to be a proper model for packet level statistics [64]. However, real world tracing shows it should be modeled as a modified Markov chain [43], or not a Markov Chain but a non-stationary stochastic process [35]. This forced us to question why the theoretical result is so different from the trace results. In Chapter 4, we present our work on the packet level statistics and compare it with previous research results. Our result shows that the packet error statistics can be modeled only as a general Markov chain without a constant transition matrix. Moreover, this process is not stationary. Some other models are proposed instead to describe the packet error process.

In Section 4.1, both the mathematical background and the previous work on bit and packet level statistics are reviewed. In Section 4.2, the packet level statistics are proved to be a time-inhomogeneous Markov chain. The stationarity of packet level statistics is discussed as well. Since a time-homogeneous Markov chain is not appropriate, finding the average transition matrix and analyzing connection performance accordingly cannot give correct results. Instead, a Hidden Markov Model (HMM) and a gap model are given in Section 4.3 as the models to discuss the statistics in this field. One application of using the gap model is also given in Section 4.3. We also use simulation results to validate the theoretical results. The contents of Chapter 4 can be summarized as:

- The previous Markov chain model is not a proper model for packet error statistics in wireless channels.
- The HMM model and Gap model are proposed for packet error statistics in wireless channels.
- Packet error statistics in wireless channels cannot be modeled by a stationary process.
- Packet error/error-free length distributions are calculated and simulated to prove the correctness of the proposed models.

Part of the modeling work on packet error statistics in wireless channels will appear in *The 27th Annual IEEE Conference on Local Computer Networks (LCN)* [33].

1.4 Performance Analysis of ARQ and FEC

After having found the proper models for packet error statistics in wireless channels, we then analyze the performance of Automatic Repeat Request (ARQ) in Chapter 5. We also evaluate the optimal packet length to achieve certain aspects of the connection performance.

Wireless channels have different characteristics compared to their wired counterparts. One difference is that a wireless channel is a more scarce resource. Thus how to utilize the bandwidth more efficiently is always a challenge for wireless communications. Second, wireless end-point devices are usually battery powered. Changing or recharging the battery is either inconvenient, e.g., a laptop computer, or even impossible, for example in some wireless sensor networks. So how to make the communication more power efficient is another challenge. Besides selecting the proper protocol in every communication layer, using the optimal packet length is critical to achieve these

two purposes. In Chapter 5, the term packet is used to refer to the packet/frame transmitted on the physical layer, which may or may not be one data link frame. Since wireless channels usually have to face many more errors, packet length could have a huge influence on the packet error rate and communication performance. We use models proposed in Chapter 4 to evaluate the optimal packet length in bursty channels.

In Section 5.1, several ARQ protocols are reviewed. Previous work on the optimal packet length is also reviewed. In Section 5.2, performance analysis while using both the HMM and the gap model are given. The general discussion on optimal packet length is given in Section 5.3 and the numerical results are given in Section 5.4, for the HMM model, gap model, and the Markov model, which was used in some previous work. We also verify the analysis results through simulation. The main purpose of the discussion on optimal packet length is not to give the optimal value for a particular situation. The contribution is to provide the performance analysis procedure using the correct models. We also verify the analysis with simulation results.

ARQ and FEC are two common error control mechanisms in computer networks. So after discussing ARQ performance, we analyze FEC performance in Chapter 6. While discussing FEC performance, we focus on the link layer forward error correction in wireless sensor networks.

Wireless sensor networks are possible and have only recently emerged. Recent advances in IC fabrication technology and digital electronics have made low-cost, low-power, and small-size sensor nodes possible. Moreover, the progress in wireless communications has enabled short-distance packet transmission. These sensor nodes consist of sensing, data processing, and communication components. Unlike traditional sensors, a large number of these tiny sensors combine to form a wire-

less network. Each of them will perform the tasks of sensing, signal processing, and sending/receiving data packets to/from other nodes following certain communication protocols. Even though each sensor has only limited capabilities, the coordinated effort of the whole sensor network could achieve some big tasks. Sensor networks thus have a huge range of application areas. Although wireless communication is critical for the flexibility provided by these sensor networks, the wireless links usually have to face many more errors than their wired counterparts. How to provide error correction becomes one of the challenges for the sensor network. In Chapter 6, we will conduct an analysis of FEC performance for wireless sensor networks. FEC uses certain error correction codes to correct some errors introduced by the transmission channel. The error correction code performs better for uniformly distributed errors. However, the errors that occur in wireless channels are usually bursty. In order to overcome this problem, interleaving technology is used to shuffle the bursty errors to become more uniformly distributed among multiple packets. But this technology may not be feasible for wireless sensor networks. Sensors usually transmit packets only periodically – interleaving will introduce longer delays into the information transmission, which may not be allowed for some applications. Moreover, sensors only have limited memory; adopting interleaving technology could put too much pressure on this limited resource. In Chapter 6, we will present the analysis of FEC performance in bursty channels without interleaving.

In Section 6.1, previous work on FEC performance analysis is reviewed. How to calculate the conditional packet error probability after FEC recovery in wireless sensor networks is discussed in Section 6.2. A gap model is set up and the analysis on some FEC performance aspects is given. Finally, we give the numerical results and discussion in Section 6.3. Again, simulation is used to verify the theoretical analysis.

Chapter 2

Error Masking Probability of 1's Complement Checksum

2.1 Previous Work

A checksum is calculated over a block of data for use in error detection. When one or more bits in the data block are changed, usually the checksum value also changes. If bits are changed but the checksum remains the same, the errors in these bits cannot be detected via the checksum. The chance of the erroneous block matching the original checksum is called the *error masking probability* or *error passing probability*. In TCP/UDP, the checksum is used to protect a packet. In this chapter, we make no distinguish between the term block and packet.

Suppose a block of data is composed of s words, where each word contains n bits and is treated as a binary integer. 2's complement checksum is the sum of these words modulo 2^n, which means the carry bits are discarded. 1's complement checksum is calculated through a different way. If the sum grows larger than n bits, the carry bits are added to the final sum. For either checksum, there are a number of block values that will give the same checksum result. As indicated in [53], the number of blocks that have the same 2's complement checksum is 2^{ns-n}. For 1's complement checksum, the checksum is 0 only when all the words in the block are 0. The possible ways of generating any other checksum value are $\frac{2^{ns}-1}{2^n-1}$.

For 2's complement checksum, the closed-form error masking probability was given in [9], under the assumption that every bit in the block has the same probability to be 0 or 1, and, each bit has the same probability to be corrupted. There is no similar result for 1's complement checksum. In [14], double and triple errors that could pass checksum error detection were analyzed, for $n \geq 3, s \geq 3$ and $n \geq 4, s \geq 5$,

respectively. However, the analysis does not include all the error possibilities. For triple errors, it was indicated that for any kind of block, the errors could pass the error detection if and only if three errors occurred in two adjacent or contiguous columns. Two errors occur in the same column, both changing 0 to 1, or both 1 to 0. The third error falls on some row in the next higher column, changing 1 to 0, or 0 to 1, correspondingly. This conclusion does not hold for 3-bit word blocks. For example, taking three words when $n = 3$, the following three cases all have the same checksum value of 011.

$$B_1 = \begin{matrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{matrix} \qquad B_2 = \begin{matrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{matrix} \qquad B_3 = \begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Suppose the original block contains B_1. When three errors occur that change these three words to B_2 or B_3, the checksum cannot detect these errors. Then, these errors will pass the error detection. Now consider case B_4, which is not included in [14]. This also gives the same checksum result, and the number of bits that differ from those in B_1 is also three. So, this error pattern will also pass the checksum error detection.

$$B_4 = \begin{matrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{matrix}$$

This suggests that a group of error patterns may not have been considered before by [14]. The method used was to analyze the blocks which can give the same summation results, expand the patterns by treating the lowest column as the next higher column of the highest column, count the number of packets falling into the pattern, and calculate the probabilities. However, when the number of bit errors is greater than or equal to n, we also have to consider the cases where the summation differs by a multiple of $2^n - 1$ from the original block. Omitting this kind of error masking pattern cannot give us correct results.

Previously, the method of getting the error probability was to analyze the patterns of error masking, count the number of packets falling into the pattern, and calculate the probabilities. The problem with this method is that finding all the error masking patterns is very difficult in some situations. First, when the number of bits corrupted, i, becomes bigger, the number of patterns, $\binom{ns}{i}$, increases until $i \geq ns/2$. In other words, the number of patterns increases until the number of bit errors is half of the number of bits in the data block. Analyzing all these patterns and figuring out which packets fall into them and thus pass the checksum will become difficult. Second, like the case we pointed out, error patterns can be complex when the number of bit errors is greater than or equal to n, even when the value of n may not be too big. In the next section, we provide a new way of calculating the error masking probability. This procedure will count directly the number of packets passing the checksum when i bits are wrong without analyzing the patterns. This allows us to provide a better estimate of the probability of undetected errors.

2.2 Error Masking Probability

2.2.1 Assumptions

We make the assumption that each bit has the same probability to be 0 or 1. We use this to simplify the analysis, even though for real TCP/UDP packets, the data distribution may not be uniform [44]. We also assume that each bit has the same probability, p, to be corrupted. After data transmission/storage, a bit with a value of 0 could be changed to 1 or vice versa. These assumptions are also made in previous work [9] [14]. We also assume that the checksum word is independent of the data words, which is also assumed in [9]. If the checksum word can only be treated as dependent, the results can be used as errors occurring only in datum words.

2.2.2 Error Value Probability of One Word

For a block of $n \times s$ bits, where the checksum is applied, each row is an n-bit word. Let v_1 be the original decimal value of one word. After transmission/storage, the word has a decimal value of v_2. The error value is defined as $v_2 - v_1$. Let $f(n, i)$ be the probability that the error value is i for an n-bit word. In other words, given a possible value of an n-bit word, there is a certain probability that some errors happen and the error value is equal to i. $f(n, i)$ is the sum of this certain probability for all 2^n possible n-bit words. We have the following relations for $f(n, i)$:

Lemma 1 *For an n-bit word with bit error probability p, $f(n, i)$, the probability that the error value is i has the following properties:*

- $f(n, i) = f(n, -i)$, *where* $|i| \leq 2^n - 1$
- $f(n, 2i) = (1 - p)f(n - 1, i)$, *where* $|i| \leq 2^{n-1} - 1$
- $f(n, 2i + 1) = \frac{p}{2}f(n - 1, i) + \frac{p}{2}f(n - 1, i + 1)$, *where* $|i| \leq 2^{n-1} - 2$
- $f(1, 0) = 1 - p$
- $f(1, 1) = \frac{1}{2}p$

Proof For one n-bit word, define the complement word as the one which contains the same number of bits, and every bit is the 1's complement of the bit which is in the same position of the original word. For any word that has an error value of i when certain bits are corrupted, its complement word will have an error value of $-i$ when the same bits are erroneous. These two events have the same probability of occurring. So, $f(n, i) \leq f(n, -i)$. On the other hand, for every case that has an error value of $-i$, the same probability will cause the error value of i in the complement word, which means that $f(n, -i) \leq f(n, i)$. According to these two relations, $f(n, -i) = f(n, i)$.

For $f(n, 2i)$, the only case that causes this error value is that no error happened on the lowest bit, whereas the higher $n-1$ bits have an error value of i. Since the lowest bit can be either 0 or 1, as long as no error occurred on this bit, this event has the probability of $1-p$. The probability of the independent event, higher $n-1$ bits with the error value of i, is $f(n-1, i)$. So, $f(n, 2i) = (1-p)f(n-1, i)$.

There are two cases that generate $f(n, 2i+1)$. Errors on higher bits will contribute an even number to the error value. In order to have an odd error value, there must be an error occurring on the lowest bit. The error value of the last bit can only be 1 or 0. An error value of 1 is generated when a 0 is corrupted to 1, while a 1 changed to 0 will give an error value of -1. When the error value of the last bit is 1, the error value of the higher $n-1$ bits should be i to make that of the whole n bits be $2i+1$. The probability that the last bit is 0 and got corrupted is $\frac{p}{2}$. So, the probability of this situation is $\frac{p}{2}f(n-1, i)$. On the other hand, when the error value of the lowest bit is -1, the higher $n-1$ bits should have an error value of $i+1$. The probability of this case is $\frac{p}{2}f(n-1, i+1)$. Because these two cases enumerate all the error patterns that will give an error value of $2i+1$, $f(n, 2i+1)$ can be expressed as $\frac{p}{2}f(n-1, i) + \frac{p}{2}f(n-1, i+1)$.

For a single bit word, the error value is 0 if and only if no error happens. The probability of this situation is $1-p$. If and only if bit 0 is changed to 1 the error value is 1, therefore the probability is $\frac{1}{2}p$. □

From Lemma 1, we can get some relationships for an arbitrary value of n:

- $f(n, 0) = (1-p)^n$
- $f(n, 1) = \frac{1}{2}p(1-p)^{n-1} + \frac{1}{2^2}p^2(1-p)^{n-2} + \frac{1}{2^3}p^3(1-p)^{n-3} + \cdots + \frac{1}{2^{n-1}}p^{n-1}(1-p) + \frac{1}{2^n}p^n$
- $f(n, 2^n - 1) = \frac{1}{2^n}p^n$

Any value of i can be expressed as the addition/subtraction of values of the form 2^k, where $0 \leq k \leq n-1$. For a certain value of i, there are one or more expressions with the fewest number of such values. It is clear that if the fewest such values for i is j, $f(n,i)$ will contain at least term $p^j(1-p)^{n-j}$. It may also contain terms with higher powers of p.

2.2.3 1's Complement Error Value Probability of One Word

From $f(n,i)$ we can define $g(n,i)$, the probability that a 1's complement error value is i, where $0 \leq i \leq 2^n - 2$. In other words, $g(n,i)$ is the sum of $f(n,j)$ for all js with the same 1's complement value of i. Now, we assume both error values 0 and $2^n - 1$ will contribute to error masking. The fact that a word composed of all 0s is the only case that generates a 1's complement checksum of 0 will be considered in Section 2.2.4.

Define the sum of one word to be *sum*. As long as $sum = A \times (2^n - 1) + i$, where A is an integer and $0 \leq i \leq 2^n - 2$, the 1's complement error value is i. The error values of 0, $2^n - 1$, and $1 - 2^n$ all generate a 1's complement value of 0. When $i \neq 0$, since the error value range of one word is $[-(2^n - 1), 2^n - 1]$, only two cases can have a matching 1's complement error value. One case is $f(n,i)$. The other one is $f(n, -(2^n - 1) + i)$.

So, we have $g(n,i) = f(n,i) + f(n, -(2^n - 1) + i)$, where $0 < i \leq 2^n - 2$ and $g(n,0) = f(n,0) + f(n, 2^n - 1) + f(n, 1 - 2^n)$. For $g(n,i)$, we have the following lemma:

Lemma 2 *The following two properties hold for $g(n,i)$, the probability that the 1's complement error value is i for an n-bit word:*

- $g(n,i) = g(n, 2^n - 1 - i)$

- $g(n, 2i) = g(n, i)$, *where* $0 < 2i \leq 2^n - 2$

Proof Using the definition of $g(n, i)$ and Lemma 1, we can get $g(n, i) = f(n, i) + f(n, -(2^n - 1) + i) = f(n, -i) + f(n, (2^n - 1) - i) = f(n, -(2^n - 1) + ((2^n - 1) - i))) + f(n, (2^n - 1) - i) = g(n, 2^n - 1 - i)$. The proof procedure can actually be explained as follows. If and only if the error value is $A(2^n - 1) + i$, where A is an integer, the 1's complement error value is i.

Similarly, if and only if the error value is $B(2^n - 1) - i$, where B is an integer, the 1's complement error value is $2^n - 1 - i$. Within the range of error values, $-(A \times (2^n - 1) + i)$ will actually enumerate all the possible values of $B(2^n - 1) - i$. According to Lemma 1, the probability of error value j is the same as $-j$, where $|j| \leq 2^n - 1$. Thus we can prove that the probability of 1's complement error value i is the same as the probability of $2^n - 1 - i$.

Define the 1-to-left shift operation as shift a word one bit to the left, and the highest bit will be added to the lowest bit. For one word, which has a 1's complement error value of i after transmission, where $0 \leq i \leq 2^{n-1} - 1$, assume the original value is v_1 and the value is v_2 after transmission. Consider the 1-to-left shift operation on both the words. The original value will be $2v_1$ or $2v_1 - 2^n + 1$. And the value after errors is $2v_2$ or $2v_2 - 2^n + 1$. So the 1's complement error value of the shifted word will be $2i$. Since the probabilities of both the words are the same, and the number of error bits are also the same in both, the probability of these two events is the same, which means $g(n, 2i) \geq g(n, i)$. Similarly, from the 1-to-right shift operation we can get $g(n, i) \geq g(n, 2i)$. Thus, we have $g(n, i) = g(n, 2i)$. □

2.2.4 1's Complement Checksum of a Block

Given $g(n,i)$ for one n-bit word, we can calculate the probability of any checksum for an s-word block. Define $b(n,s,i)$, the probability that the 1's complement checksum of an s n-bit word block is i. Making no distinction between checksum values of 0 and 2^n-1, the calculation formula of $b(n,s,i)$ is as follows:

- $b(n,1,i) = g(n,i)$
- $b(n,s+1,i) = \sum_{x_s=0}^{2^n-2} b(n,s,x_s)g(n,x)$, where $x_s + x = A \times (2^n-1)$, A is an integer.

Similar to $g(n,i)$, $b(n,s,i)$ has the following features:

Lemma 3 *The following two properties hold for $b(n,s,i)$, the probability that the 1's complement error value is i for a block of s n-bit words:*

- $b(n,s,i) = b(n,s,2^n-1-i)$
- $b(n,s,2i) = b(n,s,i)$, *where* $0 < 2i \leq 2^n-2$

This lemma can be proved easily using the similar method as in Lemma 2. And the probability that the sum of the error values in the whole block is a multiple of 2^n-1 is actually $b(n,s,0)$.

Remember that the 1's complement checksum of 0 can be generated only when all the bits in one block are 0. But, this expression includes the probability that the checksum of a block is changed from 0 to 2^n-1 and vice versa. So we need to remove these two events from $b(n,s,0)$. Define $h(n,i)$ as the probability that the 1's complement error value is i for an n-bit word composed of zeroes. Obviously, $h(n,i)$ has the following values:

- $h(n,0) = \frac{1}{2^n}[(1-p)^n + p^n]$

- $h(n,i) = \frac{1}{2^n} p^j (1-p)^{n-j}$, where $0 < i \leq 2^n - 2$ and j is the number of ones in the binary expression of value i.

For example, when $n = 3$, we have $h(3,i)$ as:

i	0	1	2	3	4	5	6
$h(3,i)$	$\frac{1}{8}[(1-p)^3 + p^3]$	$\frac{1}{8}p(1-p)^2$	$\frac{1}{8}p(1-p)^2$	$\frac{1}{8}p^2(1-p)$	$\frac{1}{8}p(1-p)^2$	$\frac{1}{8}p^2(1-p)$	$\frac{1}{8}p^2(1-p)$

Since the probability that the checksum of a block is changed from 0 to $2^n - 1$ is the same as the probability of being changed from $2^n - 1$ to 0, we can get the probability of errors that can pass the checksum, i.e., the errors resulting in no change to the original non-zero checksum value:

$$b(n,s,0) - 2 \times \sum_{x_1=0}^{2^n-2} \sum_{x_2=0}^{2^n-2} \cdots \sum_{x_{s-1}=0}^{2^n-2} h(n,x_1)h(n,x_2)\cdots h(n,x_{s-1})h(n,x_s), \qquad (2.1)$$

where $x_1 + x_2 + \cdots + x_{s-1} + x_s = A \times (2^n - 1)$, A is an integer.

This expression is the complete error passing probability, since it enumerates all the cases that give the same 1's complement checksum. Following this expression, we can calculate and get the exact probability, whereas the analysis provided in [14] is hard to extend to an arbitrary number of bit errors.

We now give an example of calculating the probability of error passing when $n = 3$.

According to Lemma 1, we can calculate any error value probability for one 1-bit word, then one 2-bit word, and finally the 3-bit word. When $n = 3$, $f(n,i)$ is

$$\begin{aligned}
f(3,0) &= \tfrac{1}{8}[8(1-p)^3 + 0p(1-p)^2 + 0p^2(1-p) + 0p^3] \\
f(3,1) &= \tfrac{1}{8}[0(1-p)^3 + 4p(1-p)^2 + 2p^2(1-p) + 1p^3] \\
f(3,2) &= \tfrac{1}{8}[0(1-p)^3 + 4p(1-p)^2 + 2p^2(1-p) + 0p^3] \\
f(3,3) &= \tfrac{1}{8}[0(1-p)^3 + 0p(1-p)^2 + 4p^2(1-p) + 1p^3] \\
f(3,4) &= \tfrac{1}{8}[0(1-p)^3 + 4p(1-p)^2 + 0p^2(1-p) + 0p^3] \\
f(3,5) &= \tfrac{1}{8}[0(1-p)^3 + 0p(1-p)^2 + 2p^2(1-p) + 1p^3] \\
f(3,6) &= \tfrac{1}{8}[0(1-p)^3 + 0p(1-p)^2 + 2p^2(1-p) + 0p^3] \\
f(3,7) &= \tfrac{1}{8}[0(1-p)^3 + 0p(1-p)^2 + 0p^2(1-p) + 1p^3]
\end{aligned}$$

Similarly, we can get $g(3,i)$ from $f(3,i)$, according to our definition.

$$\begin{aligned}
g(3,0) &= \tfrac{1}{8}[8(1-p)^3 + 0p(1-p)^2 + 0p^2(1-p) + 2p^3] \\
g(3,1) &= \tfrac{1}{8}[0(1-p)^3 + 4p(1-p)^2 + 4p^2(1-p) + 1p^3] \\
g(3,2) &= \tfrac{1}{8}[0(1-p)^3 + 4p(1-p)^2 + 4p^2(1-p) + 1p^3] \\
g(3,3) &= \tfrac{1}{8}[0(1-p)^3 + 4p(1-p)^2 + 4p^2(1-p) + 1p^3] \\
g(3,4) &= \tfrac{1}{8}[0(1-p)^3 + 4p(1-p)^2 + 4p^2(1-p) + 1p^3] \\
g(3,5) &= \tfrac{1}{8}[0(1-p)^3 + 4p(1-p)^2 + 4p^2(1-p) + 1p^3] \\
g(3,6) &= \tfrac{1}{8}[0(1-p)^3 + 4p(1-p)^2 + 4p^2(1-p) + 1p^3]
\end{aligned}$$

According to property 2 of Lemma 2, we know that $g(3,1) = g(3,2) = g(3,4)$, and $g(3,3) = g(3,6)$. From property 1, $g(3,1) = g(3,6)$, $g(3,2) = g(3,5)$, and $g(3,4) = g(3,3)$. So, when $i \neq 0$, $g(3,i)$ has the same probability. From the expression of $g(n,i)$, this relationship is obvious. This feature makes the calculation of the error passing probability easier. According to expression (2.1), we can calculate the error passing probability for a 3-bit word block. For example, the probability for a 2-word block to eventually pass the checksum is

$$\frac{1}{2^6}\left[96p^2(1-p)^4 + 216p^3(1-p)^3 + 144p^4(1-p)^2 + 48p^5(1-p) + 8p^6\right]$$

and the probability for a 3-word block is

$$\begin{aligned}
&\tfrac{1}{2^9}[2304p^2(1-p)^7 + 6858p^3(1-p)^6 + 9198p^4(1-p)^5 + 8910p^5(1-p)^4 \\
&\quad + 6138p^6(1-p)^3 + 2664p^7(1-p)^2 + 648p^8(1-p) + 72p^9]
\end{aligned}$$

2.2.5 Estimation of Error Passing Probability

When $n = 3$, the relationship between $g(3,i)$, where $0 \leq i \leq 2^n - 2$, makes the calculation easier. Unfortunately, when $n \neq 3$, $g(n,i)$ lacks such relationships to ease calculating, so too many calculations could be involved. When n is a fixed number, we can determine how many calculations are needed as the block becomes bigger. Expression (2.1) contains $(2^n-1)^{s-1}$ terms to be added together, which grows exponentially with s. The number of multiplications needed for each term is $\frac{n^2 s(s-1)}{2}$. The number of additions for each term is $\frac{n^2 s(s-1)}{2} - n$. These two numbers increase

almost quadratically with s. The total number of arithmetic operations grows even faster than an exponential function as s becomes infinite. As s becomes bigger, the calculation becomes more and more computationally intensive and eventually computationally infeasible.

Since the calculations needed for the exact probability computation grows too fast with s, the number of words in a block, estimation of the probability becomes necessary.

There is one method of estimation provided in [14]. The estimation uses the actual error masking probability caused by 2 and 3 bit errors as the lower bound. For an upper bound, all i-bit errors, where $i > 3$, are assumed to result in error passing. The probability of all the i-bit errors is $\binom{ns}{i} p^i (1-p)^{ns-i}$. This probability can be omitted only when $ns \ll \frac{1}{p}$. Under this situation only, the upper bound is almost the same as the lower bound, which means the bounds provide a good estimation of the error passing probability.

In order to get a better estimate, we analyze how the error passing probability changes when s is increased for a fixed value of n. Define x_{ab} as the probability that a 1's complement checksum of the error value equals a in an s word block when b errors occur. Define y_{ab} as the probability that the 1's complement error value of one n-bit word is a if b bits get corrupted. y_{ab} is actually the coefficient of terms in $g(n, a)$. When one more word is added to the block, s is increased by 1. $x_{0b}\prime$, the new probability of error passing caused by b errors, can be calculated as following.

When $b < n$

$$\begin{aligned} x_{0b}\prime \;&<\; x_{0b}y_{00} + x_{1b}y_{(2^n-2)0} + \cdots + x_{(2^n-2)b}y_{10} \\ &+\; x_{0(b-1)}y_{01} + x_{1(b-1)}y_{(2^n-2)1} + \cdots + x_{(2^n-2)(b-1)}y_{11} \\ &+\; \cdots \\ &+\; x_{00}y_{0b} + x_{10}y_{(2^n-2)b} + \cdots + x_{(2^n-2)0}y_{1b} \end{aligned} \tag{2.2}$$

Since one more word is added, the right part of the inequality includes new terms

that the checksum of the block is changed from 0 to $2^n - 1$ and vice versa, which should be excluded to get $x_{0b}\prime$.

Similary, when $n \leq b \leq ns$,

$$\begin{aligned} x_{0b}\prime &< x_{0b}y_{00} + x_{1b}y_{(2^n-2)0} + \cdots + x_{(2^n-2)b}y_{10} \\ &+ x_{0(b-1)}y_{01} + x_{1(b-1)}y_{(2^n-2)1} + \cdots + x_{(2^n-2)(b-1)}y_{11} \\ &+ \cdots \\ &+ x_{0(b-n)}y_{0n} + x_{1(b-n)}y_{(2^n-2)n} + \cdots + x_{(2^n-2)(b-n)}y_{1n} \end{aligned}$$

When $ns < b \leq n(s+1)$

$$\begin{aligned} x_{0b}\prime &< x_{0(ns)}y_{0(b-ns)} + x_{1(ns)}y_{(2^n-2)(b-ns)} + \cdots + x_{(2^n-2)(ns)}y_{1(b-ns)} \\ &+ x_{0(ns-1)}y_{0(b-ns+1)} + x_{1(ns-1)}y_{(2^n-2)(b-ns+1)} + \cdots + x_{(2^n-2)(ns-1)}y_{1(b-ns+1)} \\ &+ \cdots \\ &+ x_{0(b-n)}y_{0n} + x_{1(b-n)}y_{(2^n-2)n} + \cdots + x_{(2^n-2)(b-n)}y_{1n} \end{aligned}$$

And the probability that b bits got corrupted, $\sum_{a=0}^{2^n-2} x_{ab}\prime$, is

when $b < n$

$$\sum_{a=0}^{2^n-2} x_{ab} \sum_{a=0}^{2^n-2} y_{a0} + \sum_{a=0}^{2^n-2} x_{a(b-1)} \sum_{a=0}^{2^n-2} y_{a1} + \cdots + \sum_{a=0}^{2^n-2} x_{a0} \sum_{a=0}^{2^n-2} y_{ab}, \tag{2.3}$$

when $n \leq b \leq ns$

$\sum_{a=0}^{2^n-2} x_{ab} \sum_{a=0}^{2^n-2} y_{a0} + \sum_{a=0}^{2^n-2} x_{a(b-1)} \sum_{a=0}^{2^n-2} y_{a1} + \cdots + \sum_{a=0}^{2^n-2} x_{a(b-n)} \sum_{a=0}^{2^n-2} y_{an}$,

and when $ns < b \leq n(s+1)$

$\sum_{a=0}^{2^n-2} x_{a(ns)} \sum_{a=0}^{2^n-2} y_{a(b-ns)} + \sum_{a=0}^{2^n-2} x_{a(ns-1)} \sum_{a=0}^{2^n-2} y_{a(b-ns+1)} + \cdots + \sum_{a=0}^{2^n-2} x_{a(b-n)} \sum_{a=0}^{2^n-2} y_{an}$.

Now, we can calculate the ratio of $x_{0b}\prime$ to $\sum_{a=0}^{2^n-2} x_{ab}\prime$. Consider the case $b < n$. If the ratios $\frac{y_{i0}}{\sum_{a=0}^{2^n-2} y_{a0}}$, $\frac{y_{i1}}{\sum_{a=0}^{2^n-2} y_{a1}}$,..., $\frac{y_{ib}}{\sum_{a=0}^{2^n-2} y_{ab}}$ all fall in one range for any $0 \leq i \leq 2^n - 2$, then the ratio of $x_{0b}\prime$ to $\sum_{a=0}^{2^n-2} x_{ab}\prime$ should have the same upper bound. And we will use this idea to estimate the ratio.

Still consider the ratio of expression (2.2) to expression (2.3). When $n = 5$, the range of $\frac{y_{ib}}{\sum_{a=0}^{2^n-2} y_{ab}}$, where $0 \leq i \leq 2^n - 2$, is as follows:

errors	0	1	2	3	4	5
range	$0-1$	$0-\frac{1}{10}$	$0-\frac{1}{20}$	$0-\frac{1}{20}$	$0-\frac{1}{20}$	$0-\frac{2}{33}$

From the list, it is clear that the range of the coefficient ratios is $[0, 1]$, which means that the probability of passing erroneous packets is $[0, 1]$. But, further analysis makes the range $[0, \frac{1}{10}]$, i.e., the range of the coefficients for non-0-error terms in $g(n, i)$.

Notice that the upper bound of y_{i0}, 1, is the biggest among all the coefficients. 0 errors in one n-bit word can only result in an error value of 0 (no error). So, $y_{00} = 1$, and $y_{i0} = 0$, when $i \neq 0$. Thus, the ratio of the first line in equation (2.2) to the first term in equation (2.3) becomes

$$\frac{x_{0b}y_{00} + x_{1b}y_{(2^n-1-1)0} + x_{2b}y_{(2^n-1-2)0} + \cdots + x_{(2^n-1-1)b}y_{10}}{\sum_{a=0}^{2^n-2} x_{ab} \sum_{a=0}^{2^n-2} y_{a0}} = \frac{x_{0b}}{\sum_{a=0}^{2^n-2} x_{ab}}.$$

Thus, $\frac{x_{0b'}}{\sum_{a=0}^{2^n-2} x_{ab'}}$ is revised to fall in the common range of $\frac{x_{0b}}{\sum_{a=0}^{2^n-2} x_{ab}}$, $\frac{y_{i1}}{\sum_{a=0}^{2^n-2} y_{a1}}$, $\frac{y_{i2}}{\sum_{a=0}^{2^n-2} y_{a2}}$, $\ldots$, $\frac{y_{ib}}{\sum_{a=0}^{2^n-2} y_{ab}}$. The range of all these ratios except the first one is $[0, \frac{1}{10}]$ for $n = 5$. For a one word block, it is obvious that the error masking probability is 0, which also belongs to the range of $[0, \frac{1}{10}]$. So, no matter how many words are in the block, $\frac{x_{0b'}}{\sum_{a=0}^{2^n-2} x_{ab'}}$ will always fall in $[0, \frac{1}{10}]$.

Similarly, we can show that for other cases of b, i.e., $n \leq b \leq ns$ or $ns \leq b \leq n(s+1)$, the probability also falls in the range of the coefficients for non-0-error terms in $g(n, i)$. For $n = 5$, the range is $[0, \frac{1}{10}]$. Actually, we have the following lemma for the coefficient of $g(n, i)$:

Lemma 4 *For an n-bit word, $n \geq 4$,*

$$\frac{y_{ib}}{\sum_{a=0}^{2^n-2} y_{ab}} \leq \frac{1}{2n}$$

where $0 < i \leq 2^n - 1$ and $0 < b \leq n$. The upper limit is reached when $b = 1$.

Proof The proof is given in Appendix A. □

From this Lemma, we can get directly that for any n-bit word block, $n \geq 4$, where i bits were corrupted, less than $\frac{1}{2n}$ of all the i-bit erroneous packets can pass the checksum.

2.2.6 Using Properties to Simplify Calculation

When calculating the probability using expression (2.1), we can also use Lemma 3 to simplify the calculations. According to Lemma 3, some error values have the same probability of occurrence. So we can group the error values based on whether they have the same probability. Instead of calculating all the possible $2^n - 1$ values for $b(n, s, i)$, where $0 \leq i \leq 2^n - 2$, we need to calculate only one value for each group. For example, when $n = 16$, $b(n, s, i)$ has $2^{16} - 1 = 65535$ possible values for i, which can be put in 2068 groups. So we need to get only 2068 values for $b(n, s, i)$, which is much less than 65535. Notice that even using this method, the complexity of calculating $b(n, s, i)$ is still $O(sn^2)$, which means the calculation could become difficult or even impossible when either n or s becomes big. But we can indeed calculate the exact values easily under certain situations, e.g., a 256 16-bit word block. We can also calculate the values for a 750 16-bit word block, where 1500 bytes is the maximum length of IP packets. From these results, we can observe some properties of the 1's complement error passing probability. We discuss the performance of the internet checksum as well.

2.3 Discussion

2.3.1 Properties of Error Passing Probability

For a packet composed of s n-bit words, the total number of packets with i bit errors is $\binom{ns}{i}$. Only some of these packets can pass the checksum. The exact number of erroneous packets that can pass the checksum is calculated following expression (2.1), Lemma 1, and Lemma 2. We then calculate the ratio of the total number of erroneous packets to the actual number of passing packets. The results for a 4-bit word block and also a 7-bit word block are presented. In Table 2.1 and Table 2.2, the leftmost

column gives the possible number of error bits. Every other column presents the result for a block composed of a certain number of words. Each value in these columns represents the reciprocal ratio of erroneous packets which can pass the checksum when the corresponding number of errors happen. For example, a value of 10 means $\frac{1}{10}$ of the erroneous packets pass the checksum. In Table 2.2, we only show the reciprocal ratio up to 35 errors for the blocks containing more than 5 words. All other ratios not shown have the value of 127.

From Table 2.1 and Table 2.2, we observe two phenomenon. These phenomenon are also observed in error masking probabilities calculated for other different packet sizes. Let i be the number of errors in the block.

- When $1 < i < n$, the proportion of all possible i-bit error packets that can pass the checksum increases with the size of the block.
- When $i > 2n$, the proportion passing the checksum trends to approach $\frac{1}{2^n-1}$.
- When $1 < i < 2n$, for a certain value of s, the proportion trends to decrease continuously with the increase of i and finally approaches $\frac{1}{2^n-1}$.

We give the reciprocal ratios for 16-bit word blocks in Figure 2.1, the same situation as the TCP/UDP header checksum. The plot of error passing probability for a 10 word block is shown. This is the minimum size of the TCP header. We also give the plot for a 256 16-bit word block. 512 bytes is a reasonable size for TCP packets in Ethernet. This length could also be proper for wireless links with a high Bit Error Rate (BER), since a smaller packet size increases the chance of successful transmission. From this figure, the properties are also clear.

Actually, we also noticed that when i, the number of errors, becomes larger, the proportion of all the possible erroneous packets that can generate any checksum value

Table 2.1: Reciprocal Ratio of Erroneous Packets Passing Checksum, $n = 4$

number of error bits	number of words in the block							
	2	3	4	5	6	7	8	9
2	14	11	10	9.5	9.2	9	8.85714	8.75
3	28	24.4444	23.3333	22.8	22.4889	22.2857	22.1429	22.037
4	13.0233	13.3476	13.4325	13.4351	13.4167	13.3944	13.373	13.3537
5	14	15.3095	15.6564	15.8003	15.8745	15.9184	15.9468	15.9665
6	14	14.3995	14.4748	14.5067	14.5225	14.531	14.5358	14.5387
7	16	15.4778	15.3098	15.2655	15.2497	15.2431	15.24	15.2385
8	16	15.0716	14.9273	14.8995	14.8908	14.8874	14.8858	14.885
9		15.0267	15.0468	15.0558	15.0582	15.0589	15.059	15.0591
10		14.8995	14.9476	14.9645	14.9686	14.9699	14.9703	14.9706
11		15.0588	15.0107	15.0136	15.0142	15.0144	15.0145	15.0146
12		15.0588	15.0016	14.995	14.9933	14.9929	14.9927	14.9927
13			15.0052	15.0051	15.0041	15.0038	15.0037	15.0037
14			14.9927	14.9978	14.9981	14.9981	14.9982	14.9982
15			15.0037	15.0004	15.0007	15.0009	15.0009	15.0009
16			15.0037	14.9998	14.9995	14.9995	14.9995	14.9995
17				15.0005	15.0003	15.0003	15.0002	15.0002
18				14.9995	14.9999	14.9999	14.9999	14.9999
19				15.0002	15	15.0001	15.0001	15.0001
20				15.0002	15	15	15	15
21					15	15	15	15
22					15	15	15	15
23					15	15	15	15
24					15	15	15	15
25						15	15	15
26						15	15	15
27						15	15	15
28						15	15	15
29							15	15
30							15	15
31							15	15
32							15	15
33								15
34								15
35								15
36								15

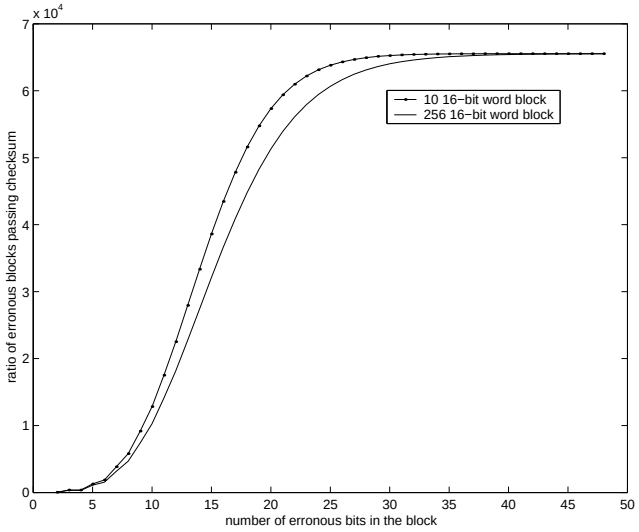


Figure 2.1: Reciprocal Error Passing Ratio Comparison for 16-bit Word Blocks

approaches $\frac{1}{2^n-1}$. One direct explanation is that when i becomes larger, the combination of where these errors can occur increases. So the checksum of the corrupted block will be distributed more evenly within the range $[1, 2^n - 1]$, which means that the probability of causing any checksum value tends to be $\frac{1}{2^n-1}$.

2.3.2 Selecting Proper Packet Length

Having calculated the error passing ratios, we can plot the error passing probability. We also plot the estimation given by [14]. The estimation uses the actual error masking probability caused by 2 and 3 bit errors as the lower bound. For an upper bound, all i-bit errors, where $i > 3$, are assumed to result in error passing. The probability of all the i-bit errors is $\binom{ns}{i} p^i (1-p)^{ns-i}$. This probability can be omitted only when $ns \ll \frac{1}{p}$.

Under this situation only, the upper bound is almost the same as the lower bound, which means the bounds provide a good estimation of the error passing probability. From Figure 2.2 and Figure 2.3, we can find that there is big difference between the

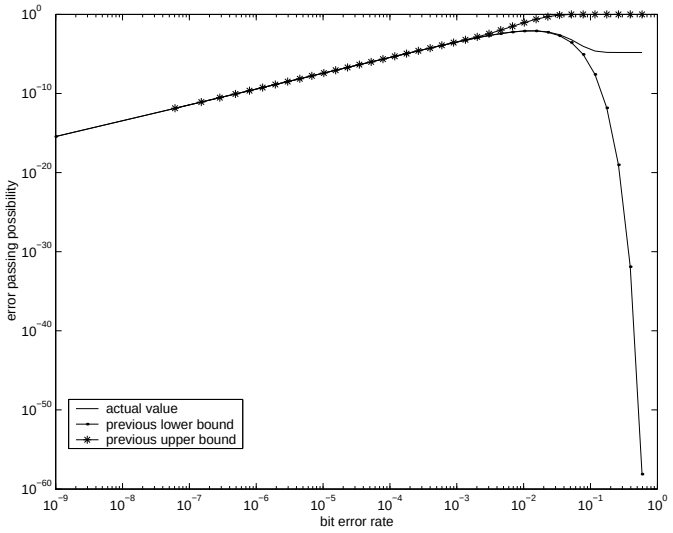


Figure 2.2: Error Masking Probability For 10 16-bit Word Block

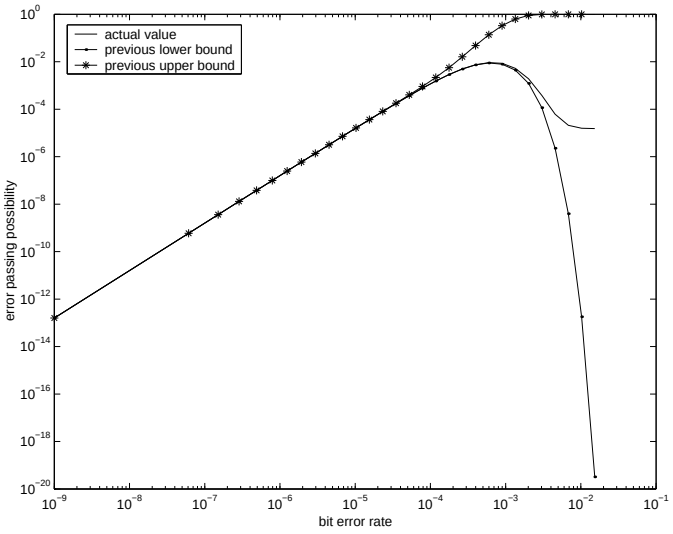


Figure 2.3: Error Masking Probability For 256 16-bit Word Block

actual error passing probability and the previous bounds when the BER becomes big. For a 10 16-bit word block, as shown in Figure 2.2, the previous bounds are good approximations for bit error rates less than 10^{-3}. The error masking probability for a 256 16-bit word block is given in Figure 2.3. 512 bytes is a reasonable size for TCP

packets in Ethernet. This length could also be proper for wireless links with high Bit Error Rate (BER), since a smaller packet size increases the chance of successful transmission. Remember that the previous bounds are accurate when $ns \ll \frac{1}{p}$, which means they are good with a lower BER when there are more words in a block. It is shown in Figure 2.3 that the bounds are tight for a BER less than 10^{-4}.

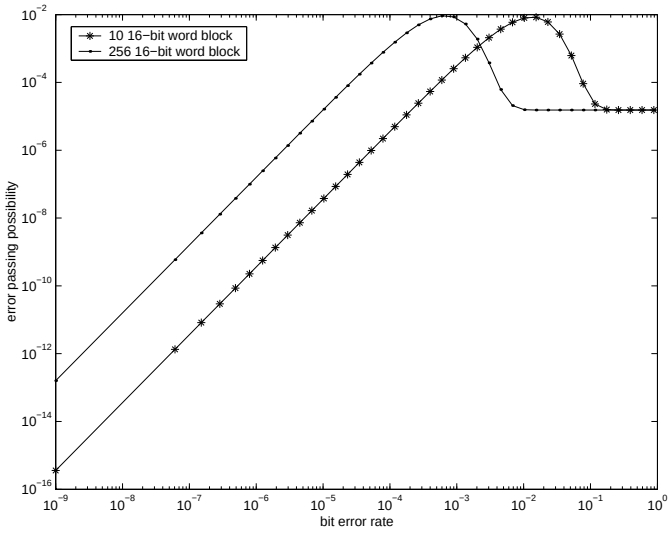


Figure 2.4: Error Passing Probability Comparison for 16-bit Word Blocks

In Figure 2.4, the actual error passing probabilities for a 10 and a 256 16-bit word packet are given. When the BER becomes big enough, the error passing probability for both a 10 word block and a 256 word block is around $\frac{1}{2^{16}-1}$. One interesting feature of the error passing probability is that for any packet size, the checksum performs worst under a certain BER. And the error passing probability of a shorter packet is not always lower than that of a larger packet. For example, when the BER $= 2 \times 10^{-3}$, a 256 word packet even has a better error detection performance than a 10 word block. According to this feature, we are able to select a better packet length given a BER, which will improve the error detection performance.

2.3.3 Performance Comparison of 1's and 2's Complement Checksum

As pointed out in Section 2.1, the number of blocks that have the same 2's complement checksum is 2^{ns-n} [53]. So the number of error patterns that can pass 2's complement checksum is $2^{ns-n}-1$. On the other hand, for 1's complement checksum, the possible ways of generating a non-zero checksum value are $\frac{2^{ns}-1}{2^n-1}$ [53], i.e., the number of error passing patterns is $\frac{2^{ns}-1}{2^n-1}-1$. It is obvious that $\frac{2^{ns}-1}{2^n-1}-1 > 2^{ns-n}-1$ when $s > 2$, which means compared to 2's complement arithmetic, more error patterns will pass 1's complement checksum.

On the other hand, people have long believed that 1's complement checksum is better than 2's complement checksum. As stated in [60], overflow bits in 1's complement arithmetic are carried over into the sum and therefore all bit positions are sampled evenly. So 1's complement checksum has the ability to detect long burst errors, which is common in communication systems. However, there are several technologies, e.g., coding and interleaving that can actually shuffle the burst errors to the more uniformly distributed errors. Under this situation, only the actual error passing probability for these two types of error detection mechanisms can decide which one has better performance. Before, only the closed form expression of error passing probability under 2's complement arithmetic was given [9], so no performance comparison could be conducted. Whereas, using the method we propose in this dissertation, we are able to calculate the error passing probability of 1's complement checksum and then compare the performance.

We give the results for a block composed of 7 words, each word containing 5 bits, to demonstrate how the error passing probability differs and why. We choose this particular block size only because using a smaller block size makes comparisons clear. We actually notice the same features with different block sizes, and the features are

expected to hold for any block size.

For a 7 5-bit word block, the error passing probability ratio of 1's complement to 2's complement checksum is plotted in Figure 2.5. When the BER is low, the ratio is lower than 1. This indicates that 1's complement checksum has lower error passing probability than 2's complement checksum, i.e., 1's complement performs better. When the BER is high, in this case, when BER > 0.2, 1's complement checksum shows worse error detecting performance.

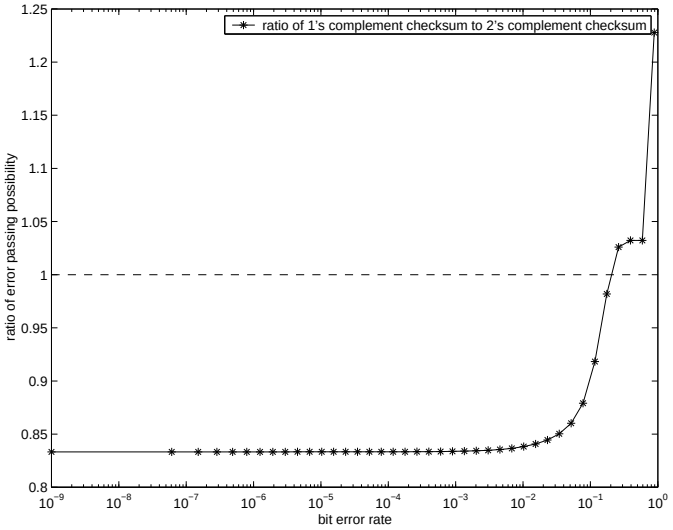


Figure 2.5: Error Passing Probability Comparison for 7 5-bit Word Block

In order to find the reason why these two kinds of checksums have different performance like this, we plot the ratio of the error patterns that can pass the checksum in Figure 2.6, again, 1's complement checksum to 2's complement checksum. We have known that more errors can pass 1's complement checksum. However, it has a different error passing distribution compared to 2's complement checksum. From this figure, we find that fewer error patterns can pass the 1's complement checksum only when the error number is small. On the other hand, when the number of errors becomes larger, more error patterns can pass under 1's complement arithmetic. When

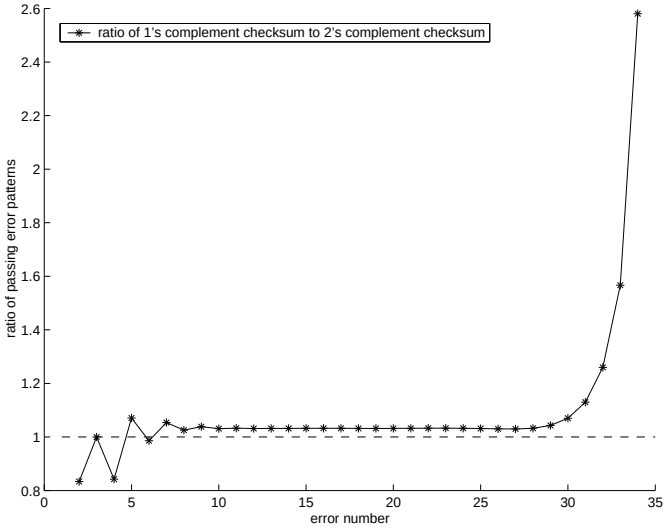


Figure 2.6: Number of Error Passing Pattern Comparison for 7 5-bit Word Block

BER < 0.5, which is the normal case in physical communication channels, the error probability decreases when the number of errors increases. When the BER > 0.5, the opposite result happens. So, even though more error patterns cannot be detected when 1's complement checksum is adopted, the actual probability of these events is smaller when the BER $<<$ 0.5. 1's complement checksum could give worse error detecting probability only when the BER becomes bigger.

2.3.4 Internet Checksum Performance Over Real Data

In [44], the performance of internet checksum and CRC are traced over real data. The trace was conducted for an IP over ATM system, using an FTP transfer between UNIX machines. The trace was actually to replace part of a TCP packet with a certain part of the following packet and then to verify whether CRC or TCP checksum could detect that some errors had occurred. For CRC, the tracing result is that the error detection performance is almost the same as the theoretical result. However, TCP checksum has error detection performance 10 to 100 times worse than $\frac{1}{2^{16}-1}$. The

authors of [44] then did more tests. It turns out that the data in real world are not distributed uniformly, so the distribution of the checksum values is not uniform. The hot spots of checksum values actually cause the tracing results.

Obtaining the error passing probabilities of a 16-bit word block, we can get a deeper understanding of this phenomenon. In real world tracing, the block can be replaced by the following packet containing only 6 to 48 words. When the part of the following packet has the same checksum value as what it replaces, the checksum cannot detect the errors. The traditional thinking is that the internet checksum can catch any single error, any burst error of 15 bits or less, and all 16-bit burst errors except for those which change all zeros to all ones and vice versa. And it is expected to catch all other types of errors at a rate proportional to 1 in 2^{16}. Under this assumption, the expected error passing probability should never exceed $\frac{1}{2^{16}}$. Whereas the real case is not like this. When the block can be replaced by one composed by arbitrary bits, it is equal to the case that the bits can be corrupted with a probability of 0.5. From Figure 2.4, it is clear that the error passing probability is equal to $\frac{1}{2^{16}-1}$. However, when the block can be replaced only by a somehow related one, as what happened in the "real world" test, each bit has much less probability to be corrupted. As shown in Figure 2.4, the error passing probability could be much bigger than $\frac{1}{2^{16}-1}$.

Actually, even when 0 and 1 are distributed evenly in real world data, the error masking probability could be bigger than $\frac{1}{2^{16}-1}$. We give the expected error passing probability following this assumption in Figure 2.7. The actual error passing probability is always larger than the expected value until the BER is very big. In the traditional belief, the error masking probability can never be bigger than $\frac{1}{2^{16}-1}$. However, the actual probability can be much bigger than this.

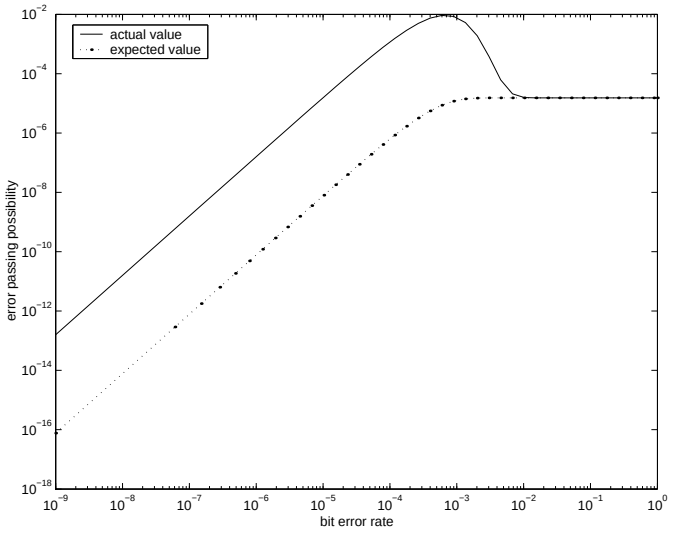


Figure 2.7: Error Passing Probability for 200 16-bit Word Block

2.4 Conclusion

So far, we have provided an exact formula for error passing probability for 1's complement checksum. We also use some features to simplify the computation of actual values.

Since too much calculation is needed for bigger data blocks, we also make estimates on the probability. We achieve an upper bound for n-bit word block, $\frac{1}{2n}$, when $n \geq 4$, which is much better than previous work. We also simplify the calculation of error masking probabilities directly using the properties of $g(n, i)$, where $0 \leq i \leq 2^n - 2$. Even though the complexity of the simplified calculation is still $O(sn^2)$, we can use it to get the error masking probabilities for a 256 or 750 16-bit word block. From the results, we observe and analyze some properties. We also compare the performance of 1's complement and 2's complement checksum.

The discussion of the influence on error detecting ability caused by some protocol proposals [28] [12] is necessary. In the VJ TCP/IP header compression protocol [28],

some header fields are transferred using only the difference between those of the previous packet, instead of the original values. The influence on error detecting probability needs further consideration. For the "twice" header compression protocol [12], the TCP header checksum is also used to help recover the value of these fields if some compressed packets are lost. The error masking probability of this algorithm is definitely higher than that of VJ compression, since errors passing CRC could contribute to generating wrong values for these fields. Thus, this issue needs more careful consideration.

Table 2.2: Reciprocal Ratio of Erroneous Packets Passing Checksum, $n = 7$

number of error bits	number of words in the block							
	2	3	4	5	6	7	8	9
2	26	20	18	17	16.4	16	15.7143	15.5
3	104	84.4444	78	74.8	72.8889	71.619	70.7143	70.037
4	114.4	84.4444	74.2857	69.1792	66.1085	64.0593	62.5947	61.4959
5	163.429	129.2	117.244	111.163	107.484	105.019	103.252	101.924
6	163.429	135.777	124.457	118.297	114.426	111.769	109.833	108.36
7	138.754	135.822	131.262	128.05	125.795	124.149	122.902	121.927
8	120.421	130.3	129.69	128.263	126.991	125.957	125.124	124.445
9	108.952	125.698	127.815	127.917	127.616	127.248	126.898	126.587
10	106.419	124.402	126.783	127.285	127.315	127.206	127.058	126.906
11	110.933	125.038	126.452	126.933	127.09	127.12	127.098	127.054
12	118.857	126.639	126.632	126.862	126.982	127.031	127.042	127.034
13	128	128.269	126.939	126.911	126.959	126.993	127.01	127.017
14	128	129.156	127.151	126.982	126.975	126.987	126.997	127.003
15		129.152	127.205	127.026	126.994	126.992	126.996	126.999
16		128.54	127.13	127.033	127.005	126.998	126.998	126.998
17		127.744	126.999	127.019	127.007	127.001	126.999	126.999
18		127.148	126.88	126.999	127.004	127.002	127	127
19		126.909	126.812	126.986	127	127.001	127	127
20		127.008	126.807	126.984	126.998	127	127	127
21		127.008	126.85	126.991	126.998	127	127	127
22			126.912	127.002	126.999	127	127	127
23			126.966	127.012	127	127	127	127
24			126.996	127.017	127.001	127	127	127
25			127.003	127.018	127.001	127	127	127
26			126.999	127.014	127.001	127	127	127
27			127	127.009	127	127	127	127
28			127	127.004	126.999	127	127	127
29				127.001	126.998	127	127	127
30				127	126.998	127	127	127
31				127	126.999	127	127	127
32				127	126.999	127	127	127
33				127	127	127	127	127
34				127	127	127	127	127
35				127	127	127	127	127

Chapter 3

Header Compression for Wireless Networks

3.1 Previous Work

In the early stage of computer networking, most of the popular mediums for connecting home PCs to the Internet were low speed serial links. One example is the modem, which could support 300 to 19,200 bps. At that time, people wanted to conserve the bandwidth of this kind of link, and be able to connect with the Internet faster. So, in 1990, Van Jacobson proposed a TCP/IP header compression algorithm for low-speed links [28]. This algorithm was designed for TCP running over IPv4. Jacobson carefully analyzed how the TCP/IP headers change for each packet throughout a connection. By using the changing pattern, the algorithm can compress the usual header size of 40 bytes down to 4-17 bytes. VJ compression is a proposed IETF standard, and it has been very widely deployed.

Even today, the bandwidth provided by modems is still quite limited, less than 100Kbps. At the same time, the communication mediums used in computer networks have changed much. Wireless is becoming a popular way for connecting computers to the Internet. The bandwidth of wireless links varies over a wide range. Wide-area networks, due to constraints of the physical medium, can only provide low-speed bandwidth, on the order of 10Kbps. Some emerging new protocols will support higher data transmission over conventional voice channels [52], on the order of 100Kbps or several Mbps. The other kind of network, local-area wireless networks, operates on a smaller geographical area, which in turn faces a better physical environment. It can operate at a few Mbps, which is also expected to be improved. But, due to the regulatory limitations on the use of radio frequencies and inherent information

carrying limits, all wireless links will remain scarce resources even after these new protocols are fully deployed. Again, people need to conserve the bandwidth and to use the link more effectively.

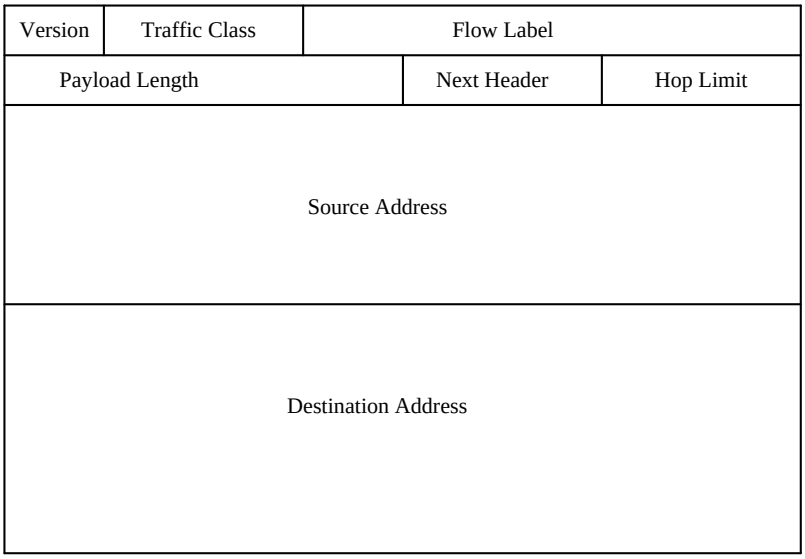
Besides the change of network mediums, Internet protocols also face changes.

The first change is that the Internet Protocol, IPv4, will finally be replaced by a new protocol, IPv6 [13]. In IPv6, the address size will be increased from 4 bytes to 16 bytes. In addition, some fields of IPv4 are removed from the basic IPv6 header in order to speed the packet processing on routers. So the basic IPv6 header is 40 bytes, while the minimum IPv4 header is 20 bytes. Moreover, various extension headers can be added to the basic IPv6 header to provide various routing, security, and other features. The larger header will definitely add more challenges on wireless links.

The second change in Internet protocols is particular to mobile users. The need for wireless data communication arises partially because of the need of mobile computing and partially for some specialized applications. Under some cases for mobile computing, it is required that one IP header be encapsulated in another one, or a routing header is added to an IPv6 header. Even though mobility provides convenient access to the Internet and has become an important use of wireless networks, fulfilling this task requires more bandwidth, which puts more pressure on wireless networking resources.

In 1996, Degermark et al. proposed a header compression algorithm for UDP and TCP for IPv6 networks [11]. The header compression for TCP is quite similar to VJ compression. The basic idea of compression is not transferring redundant information whenever possible. Thus, the number of bits in a compressed header will become less. Figure 3.1 is the structure of the IPv6 header [13].

The TCP header format is described in Figure 3.7 as presented in [46].



Version	4-bit Internet Protocol version number = 6.
Traffic Class	8-bit traffic class field.
Flow Label	20-bit flow label.
Payload Length	16-bit unsigned integer. Length of the IPv6 payload, i.e., the rest of the packet following this IPv6 header, in octets.
Next Header	8-bit selector. Identifies the type of header immediately following the IPv6 header. Uses the same values as the IPv4 Protocol field.
Hop Limit	8-bit unsigned integer. Decremented by 1 by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero.
Source Address	128-bit address of the originator of the packet.
Destination Address	128-bit address of the intended recipient of the packet (possibly not the ultimate recipient, if a Routing header is present).

Figure 3.1: IPV6 Header

The TCP/IP header fields can be divided into the following groups:

- Constant fields. These fields usually do not change during the lifetime of a connection, for example, source address, source port, destination address, and destination port.
- Inferable fields. These fields can be inferred from other fields, like the size of the frame.

- Delta fields. These fields are expected to change only slightly from the fields of the previous packet.

- Random fields. These fields have no relationship with other packets, like the checksum in the TCP header.

Constant fields can be eliminated from most packets. Instead, a unique number, the Compression Identifier (CID), is assigned and added to the packet header to identify these fields. Inferable fields will not be transferred at all. The decompressor will calculate and fill them in. Delta fields will be transferred using only the differences, which can be expressed with fewer bits and are referred to as "delta" values. No change will be made to random fields, since there is no way to predict or calculate them. Thus, several new kinds of packet types are defined under header compression. The packet types defined in [28] [12] [17] [19] are quite similar. The compression method uses the following packet types in addition to the IPv4 and IPv6 packet types:

- UNCOMPRESSED_TCP - a packet with a full TCP/IP header as well as the CID.

- COMPRESSED_TCP - a packet with all fields compressed and the CID added.

- SEMICOMPRESSED_TCP - a packet with all fields compressed except delta fields and with the CID added.

The decompressor records or revises fields when receiving UNCOMPRESSED_TCP packets. For other types of packets, the CID will be used to recover the constant fields. In the case of SEMICOMPRESSED_TCP packets, delta fields will be recorded. For COMPRESSED_TCP, delta fields will be calculated and recorded.

In [11], one more mechanism is designed to repair the de-synchronization between the compressor and the decompressor, which we will refer to as the "twice" algorithm.

When a packet cannot be decompressed correctly, the decompressor will assume the reason is that one or more previous packets are lost. The decompressor also assumes that all the packets carry the same delta values and then tries to decompress the packets using these values two or more times. The twice algorithm improves the performance of header compression in certain circumstances. This algorithm has been developed as a proposed IETF standard [12].

Usually, wireless links are prone to experience significant error rates and large delay times, which puts more difficulties on header compression. The Robust Header Compression (rohc) working group was created in IETF to improve header compression performance under this situation. In [17], some mechanisms are summarized for robust header compression over links with significant error rates and long round-trip times. One of the encoding methods, window_based LSB, is improved and presented as TCP_Aware RObust Header Compression (TAROC) in [19]. Window_based LSB [17] is an encoding method for delta values. If the compressor can ensure that the decompressor has received a group of packets, it can then transfer the delta values as the differences from that of the group, which are still expected to occupy fewer bits. TAROC uses a feature of TCP congestion control to decide which packets have already been received. The TCP sender keeps a sliding window. A new packet cannot be transferred without getting all the acknowledgments for the previous window. So the compressor tries to track the size of the sliding window according to the arriving sequence of packets. It will then know which the packets have been received by the decompressor and use window_based LSB accordingly.

3.2 New Challenges

A high frequency of some problems for computer networking was shown recently through tracing network connections [6] [58]. Header compression should be studied under the existence of these problems, since these problems could be encountered by users under typical operating conditions.

3.2.1 Packet Reordering

Packet reordering is not a new problem in computer networking. Unlike circuit-switching networks, where all the signals of one connection go through and wholly occupy a predefined route, the information may go through different routes in computer communication. In computer networks, the information that needs to be transferred between two ends is usually packaged into multiple packets. Each packet carries the address of the destination and enters the network. The routers will then try to deliver each packet according to the address as well as the current availability of network resources. So, packets of one connection are not guaranteed to all use the same route, which makes the sequence of arrival at the destination unknown. The usual belief is that when some network components are not working correctly, or there are changes to the network configuration, routers will transfer packets through different paths, which will cause packet reordering at the destination. This reasoning leads to the conclusion that the frequency and magnitude of the phenomenon are not severe.

Recently, Bennett et al. showed that reordering is not necessarily a rare occurrence in the Internet [6]. Even when a consistent path is chosen between a source and destination, the existence of multiple or redundant paths between routing neighbors, or within the data paths of a single router, can cause packet reordering. In order to handle high speed networking, more and more multiple paths are being used between

routing neighbors to form a logical link. At the same time, routers are also implementing more and more internal parallelism, which includes parallel data processing paths and parallel decision making for packet forwarding. When packets belonging to one connection are coming in rapidly or the router's load is high, the packets can take any path through the routers and then reordering may be unavoidable. Packet reordering is the natural result of the logical link as well as the router parallelism. Making changes to maintain the packet sequence within each connection might decrease the packet exchange speed. And keeping a high routing speed while eliminating packet reordering becomes quite expensive or even very difficult theoretically.

This non-pathological cause of reordering makes it more prevalent than previously believed. Reordering caused by the selection of different routes for packets associated with a particular connection may not be severe enough to cause significant problems in header compression. However, frequent and distant packet reordering caused within one route challenges header compression algorithms. One could argue that non-pathological packet reordering is not severe if the data transmission is really slow for a connection. This could be true under certain situations. But, as long as the bandwidth-delay product is large, the sliding window size of the TCP sender is large. It is possible that the sender has many packets to send and it can send as many as the window size out quite rapidly if the link connecting to the sender is not slow. Then these packets do have a high probability to be reordered. Some header compression algorithms assume and use the feature that the packets arrive with the same order as sent by the sender or with minor reordering. These algorithms need to be evaluated carefully considering more frequent packet reordering.

3.2.2 Errors passing CRC

When TCP/IP datagrams are passed over Ethernet, the link layer uses Cyclic Redundancy Check (CRC) to detect errors. PPP also uses CRC. In most wireless data networks, CRC is also the way to detect errors at the link layer. It has long been known that CRCs are very powerful for error detection. CRCs are based on polynomial arithmetic, base 2. CRC-32 is the most commonly used CRC in the TCP/IP suite. CRC-32 can detect all bursty errors with length less than 32 bits and all 2-bit errors less than 2048 bits apart. For all other types of errors, the chance of not detecting is just 1 in 2^{32}. Given this point, one can argue that the TCP or UDP checksum is not necessary. Practically, this was tried in the 1980s [58]. For some Network File Server (NFS) implementations, UDP checksum was disabled based on this argument. But this idea resulted in file corruption and ultimately was discarded as a bad idea.

Recently, Stone and Partridge have shown that for the Internet today, there are a wide variety of error sources which cannot be detected by link-level CRCs [58]. Defective hardware, buggy software, and problems in both end-systems and routers can all cause errors that will pass link-level error detection. Newly emerging software and hardware are especially vulnerable to these problems. In essence, any errors introduced at protocol layers above the link layer will not be detected by the link-layer CRC. Changes should be made to eliminate these error sources. But, before old error sources are cleaned up, or after new sources are introduced, the best solution is to detect these errors. In TCP, this task can be done only by Internet checksum [46] [50]. When errors pass the checksum, these errors will be passed to higher layers, including the application layer. The performance of header compression algorithms needs reconsideration under these errors.

3.3 Performance of Header Compression Algorithms Under New Challenges

For each header compression algorithm we can analyze the probability that the packets cannot be decompressed, including corrupted packets and packets influenced by error propagation. Assume every bit has the same probability to be corrupted and every bit error will be detected by the error detection mechanism, either by the link layer or by the transport layer. Also, assume all the packets are of the same length. Then all the packets with the original header will have the same error probability, define it to be q_o. And all the packets with the compressed header will have the same error probability and define it to be q_c. Since the compressed header is shorter than the original header, $q_c < q_o$. We will conduct the analysis using the length of an UNCOMPRESSED_TCP packet as 612 bytes and the length of a COMPRESSED_TCP packet as 517 bytes. The analysis for existing header compression algorithms is described as follows.

3.3.1 VJ Compression

The problem with VJ header compression basically comes from the packet error or loss caused by error propagation. Since TCP uses the ACKs from the receiver as the congestion indication, the packet error or loss has more influence on TCP performance than just the erroneous/lost packet itself. This issue was discussed clearly in [11]. We give the packet error probability for VJ header compression in Figure 3.2, when the bit error rate (BER) is 10^{-5}. A corrupted COMPRESSED_TCP packet will be dropped. A COMPRESSED_TCP packet transferred correctly will also be discarded if the previous packet was dropped. So the packet error probability increases almost linearly until an UNCOMPRESSED_TCP packet is received. After that, the packet error probability will drop and start increasing again.

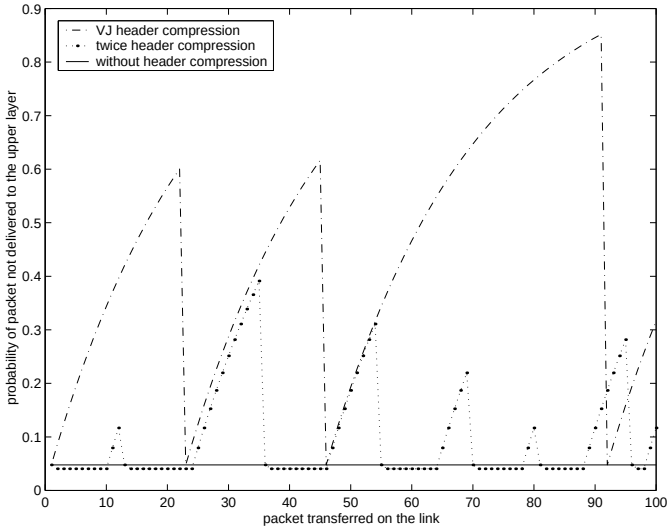


Figure 3.2: Packet error probability comparison, $BER = 10^{-5}$

Moreover, the compression efficiency is influenced by packet reordering. In the VJ compression algorithm, uncompressed reference packets are sent when a new CID is created, when constant fields change for a CID, and when the decompressor goes out of synchronization. When a packet cannot be decompressed, TCP will detect it by timeout or duplicated ACKs and retransmit it. The compressor will find this retransmitted packet has a smaller sequence number than the previous packet, which causes a negative delta value. The compression algorithm will disable the occurrence of a negative delta value and force a full header to be transferred in this case. This mechanism ensures that the information of one CID is refreshed on the decompressor side. But, it is also activated by packet reordering even when no packet loss occurs. When reordering happens, the sequence of packets arriving at the compressor is not the same as when they left the sender. There are packets that arrive at the compressor later than some packets which, at the TCP sender, enter the network following these packets and have larger sequence numbers. These packets will be sent with a full header instead of a compressed header. This introduces difficulties on estimating

compression efficiency. Let B be the header size before compression and A the average compressed header size, supposing all other factors except packet reordering have been considered. If reordering is only a pathological behavior, which is rare, the compression ratio can be estimated as $R_{estimated} = A/B$. But since reordering could be more prevalent, we need to consider the effect on the compression ratio. Define x to be the percentage of packets that are received after their subsequently transmitted packets. Then, the compression ratio should be

$$R_{actual} = (A(1-x) + Bx)/B \tag{3.1}$$

Now, let's analyze $(R_{estimated} - R_{actual})/R_{actual}$, the error rate of the estimated compression ratio.

$$(R_{estimated} - R_{actual})/R_{actual} = 1 - 1/(1 + (B/A - 1)x) \tag{3.2}$$

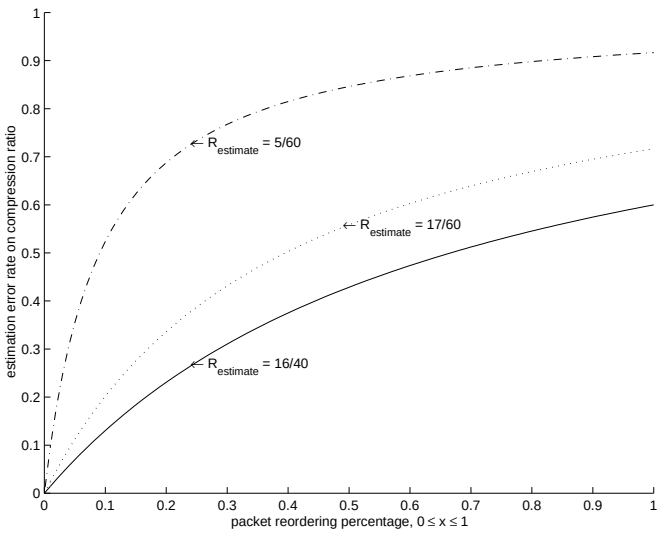


Figure 3.3: Estimation error rate of the compression ratio

This ratio is plotted in Figure 3.3. While using VJ compression, the typical size of the COMPRESSED_TCP ranges from 4 to 17 bytes when no optional fields exist

for the original headers. We show three curves for the estimated error rate of the compression ratio. One is when the designed compression ratio is low, at 16 to 40. This corresponds to a TCP (20 bytes) and an IPv4 header (20 bytes) being compressed to 16 bytes. When packets need to be tunneled to a mobile node, the original packet is encapsulated in one extra IPv4 header. With one layer of encapsulation, the total header will be 60 bytes. It can be compressed to 17 bytes, which is shown in the second curve. It can also be compressed to 5 bytes and this is depicted in the third curve. We can find that the more efficient the compression algorithm, which is indicated by a smaller value of A/B, the more error that is on estimating the compression ratio given a certain value of x.

3.3.2 Twice Algorithm

Packet reordering also triggers the twice algorithm to send UNCOMPRESSED_TCP packets. This will decrease the compression ratio, the same as discussed for VJ compression.

Packet reordering has more influence on the performance of the twice algorithm. In this algorithm, if one packet is lost, subsequent packets are tried to be decompressed by applying their delta values two or even more times. If packets are of the same size and there is no reordering/loss before the compressor, the algorithm will always recover the loss on the link between the compressor and decompressor. But, when there is reordering before the compressor, the result will be different.

Consider several packets belonging to one connection. We number them from 1 to 6, as the sequence when they were sent out from the sender. Assume the TCP data payload of the packets are all of the same length, d. Then, the compressed packets will all have the delta value d for the sequence number field.

$packet$	1	2	3	4	5	6
$deltavalue$	/	d	d	d	d	d

No matter which packet is lost on the link, or even if several consecutive packets are corrupted, the decompressor can recover all other packets after the loss.

But, if packets 4 and 5 are reordered before the compressor, the delta value carried by the compressed packets are quite different and the twice algorithm cannot work correctly under some cases. If packet 3 was corrupted, packet 5 cannot be decompressed. The reason is the delta value carried by packet 5 is $2d$, applying it two or more times still cannot recover the original value. Similarly, if (4), (5, 4), or (3, 5, 4) are lost, 6 cannot be decompressed. Obviously, when packets are not of the same size, the twice algorithm also fails to eliminate error propagation.

$packet$	1	2	3	5	4	6
$deltavalue$	/	d	d	$2d$	$full-header$	$2d$

The reason for this error propagation phenomenon is that the twice algorithm assumes that most compressed packets have the same delta value for many applications. But, even for these applications, reordered packets make this assumption invalid, which causes the twice algorithm to not perform well. So if a packet cannot be decompressed, using the packet length instead of the delta value to recover the sequence number could be helpful.

When a packet cannot be decompressed, the twice algorithm assumes one or more packets are lost on the link between the compressor and decompressor. But, packet reordering makes the sequence between packets more complex. Again, take the above example. This time, packet 6 comes before packets 4 and 5.

$packet$	1	2	3	6	4	5
$deltavalue$	/	d	d	$3d$	$full-header$	d

If packet 4 is lost, then packet 5 cannot be decompressed using twice. Now, packet 5 seems to be coming later than the previous packet seen by the decompressor, instead of one packet missing before packet 5. In this situation, extending the twice algorithm to also subtract the delta value from the sequence number will be helpful.

In the examples we used, TCP/IP is assumed to have packet reordering only within a small range, up to 3 packets. It is clear that packet reordering does hamper the performance of the twice algorithm even in this situation.

The packet error probability for the twice algorithm is also shown in Figure 3.2. The twice algorithm works fine if most of the packets are of the same size and no packet reordering happened. When packet reordering does occur, the packet error probability could increase almost linearly just as in VJ header compression. If applying delta values more times can decompress a subsequent packet, the error probability will become low again. Otherwise, the probability can drop only when a packet with a full header is received by the decompressor.

The errors that can pass link layer detection also influence the performance of the twice algorithm. We define *error masking probability* as the probability that those errors are not detected by the TCP checksum and thus passed to higher layer. The error masking probability of TCP checksum needs more analysis. The influence on VJ header compression of this issue also needs further consideration. But, one point is clear, the error masking probability of the twice algorithm is higher than that of VJ compression. If the delta values are applied once and the TCP checksum is not correct, the actual reason may be some errors in the data payload or other bytes in the header of the packet. Then, when the delta values are applied two or even more times, the probability increases that the TCP checksum matches the packet. In this case, errors that could be found by the VJ algorithm will be passed to higher

layers. For example, assume the decompressor is synchronized with the compressor and the previous sequence number is xxxxxxxx,xxxxxxxx,xxxxxx1x,xxxxxxxx, where x means either 0 or 1. And assume the decompressor receives a compressed packet with the delta value of sequence number 512, the delta value of ACK number 0. But in this packet, there is an error that avoided the link-level CRC. That is, one of the 2-octets as is used for TCP checksum, either in the header or in the data payload part, was changed from xxxxxxxx,xxxxxxxx,xxxxx1xx,xxxxxxxx to xxxxxxxx,xxxxxxxx,xxxxx0xx,xxxxxxxx. And only this one bit got corrupted. Then, when the decompressor tries to recover the packet, the TCP checksum indicates an error if the delta value is applied once. The TCP checksum is correct when the delta value is used twice, which is an indication that the packet has been decompressed correctly. Thus, the error is passed to the higher layer with a wrong sequence number.

Given that an error has passed the link-layer CRC, the more times the delta value is applied, the more likely that the error can be passed to the layer above TCP. Similarly, if a sophisticated implementation of the twice algorithm [12] is used to make more educated guesses for the acknowledgment stream, where the delta value for the ACK number is not regular due to the delayed ACK mechanism, there is an increasing probability that the errors can pass the transport layer error detection.

3.3.3 TCP_Aware RObust Header Compression (TAROC)

TCP_Aware RObust Header Compression (TAROC) improved one of the encoding methods, window_based LSB. W_LSB sends the delta fields only as the changed lower bits relative to the corresponding fields belonging to a certain window. TAROC restricts the size of the window according to the sliding window of TCP flow control. The TCP sender can transmit a packet only after all the packets belonging to the

previous sliding window have been acknowledged by the receiver. In other words, if the window size is *cwnd* when one packet is transmitted, the decompressor must have received correctly all those packets *cwnd* before this packet. TAROC then uses this knowledge and W_LSB encoding to compress the header. Theoretically, TAROC will work perfectly if the compressor knows when the sender changes the window size. Unfortunately, this is not an easy task. TAROC uses the packet arrival sequence as the window size indication. Again, packet reordering will influence the accuracy of this estimate. If packet reordering will be assumed as an indication of packet loss at the compressor, the compressor believes that the sender has adopted a smaller window size following that. Thus, the decompressor is assumed to has received a number of packets correctly, where the number is more than what actually has been received. This will potentially make the decompressor work incorrectly. We give the packet error probability in Figure 3.4. In order to show the difference between TAROC and TCP/IP without header compression, we use a logarithmic scale on the Y-axis. The probability for TAROC is very low if the compressor can make an accurate estimate of the sliding window size. However, if the compressor makes a smaller estimate, all the packets will be dropped until an UNCOMPRESSED_TCP packet is received.

On the other hand, for some TCP connections with large bandwidth-delay products, the sliding window size could be very large. The advantage of W_LSB will become smaller or even no bits can be compressed in this situation.

3.4 Adaptive Header Compression Algorithm

3.4.1 Wireless Link and Connection Consideration

For mobile radio channels, especially the channels used for typical transmissions from a base station to mobile users in an urban environment, Rayleigh fading is the widely accepted model. Fading causes periods of significant degradation of the received

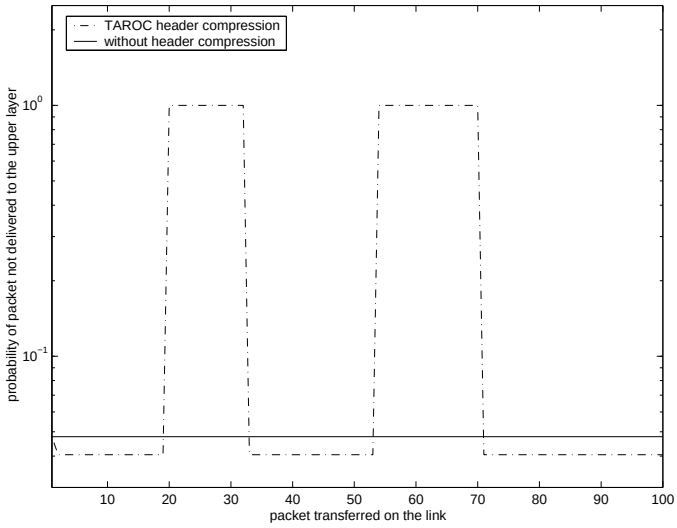


Figure 3.4: Packet error probability comparison, $BER = 10^{-5}$

signals. Received signals determine the bit error rate (BER) seen by the receiver. When the signal to noise ratio is above a certain threshold, the channel is assumed to be in the *good* state with a low BER. On the other hand, when the ratio is below this threshold, the channel is in the *bad* state, where the BER is quite high. Under the Rayleigh fading model, the wireless channel is considered to have these two states and keep changing between the states.

When the wireless channel is in the *good* state, original packets can be transferred correctly with high probability. Compressed packets can have an even higher probability of correct transmission. Any saving on the bandwidth will be beneficial for the particular connection as well as the whole network. The only task in this state is to lessen or eliminate error propagation. When the wireless channel is in the *bad* state, both original packets and compressed packets can barely be transmitted correctly. As long as the header compression algorithm can lessen the error propagation for the next *good* period, the algorithm will have good performance. Unfortunately, it is almost impossible for the compressor of a wireless channel to know the current state

of the link. But the sender may know some general characteristics of the wireless link and select a proper header compression algorithm. The algorithm can also be adapted following the changes in channel conditions throughout the connection period.

From the performance discussion of previous header compression algorithms, we can reach the conclusion that assuming no reordering or minor reordering will introduce problems in typical connections, especially those with large bandwidth-delay products. Although the influence of packet reordering on TCP performance needs more consideration, the best use of the communication channel is to transfer these packets efficiently. So the task of header compression becomes transferring the packets in the sequence as received from the previous node, using as little bandwidth as possible.

3.4.2 Adaptive Header Compression

Based on the previous discussion, we present the following algorithm, the adaptive header compression algorithm, to achieve a good tradeoff between the throughput and the compression ratio.

The packet types we use are similar to those defined in [28] [12] [17] [19]. We use the W_LSB to do the delta field encoding. We can also combine all the delta fields together and use padding to keep them aligned to byte boundaries. For each packet, we use a certain number of packets received beforehand to do the W_LSB encoding. Since assuming a fixed sequence between the TCP packets will potentially hamper the performance, we make no such assumption. As long as the decompressor has received any packet of this group correctly, it will decompress this packet correctly. For wireless links in the *good* state or if the period of *bad* state is short, the probability that a group of packets are all corrupted is still low. In a long-term *bad* state, the probability could be high. In order to handle this, we send UNCOMPRESSED_TCP

packets periodically to refresh the decompressor. By designing the algorithm carefully, we can send more packets during the *good* state while making decompressor work properly shortly after the *bad* state ends.

The compressor performs as follows:

- Define two variables, *windowSize* and *distance*. These two variables can be adapted according to the link condition and the average packet size. *windowSize* determines how many packets should be used to do W_LSB encoding, and *distant* defines how frequently a SEMICOMPRESSED_TCP should be sent to keep the decompressor synchronized with the compressor.

- For the first *windowSize* packets belonging to the connection, send an UNCOMPRESSED_TCP packet with exponential distance, send other packets as SEMICOMPRESSED_TCP. In other words, send packet $(2^i - 1)$ as UNCOMPRESSED_TCP and all other packets as SEMICOMPRESSED_TCP. If the *windowSize* is small, send all the first *windowSize* packets as UNCOMPRESSED_TCP.

- After this, send a SEMICOMPRESSED_TCP packet every *distance* packets. If within the previous *distance* packets one SEMICOMPRESSED_TCP has been sent, send a COMPRESSED_TCP instead.

- Whenever one or more constant fields change, send the packet as UNCOMPRESSED_TCP.

- If a packet has a sequence number seen by the compressor before, send it as UNCOMPRESSED_TCP.

- Other packets are sent as COMPRESSED_TCP, with delta fields using W_LSB

encoding. The window is composed of the previous *windowSize* packets sent by the compressor.

When the packet size is large, or there is severe packet reordering, or *windowSize* is big, COMPRESSED_TCP sometimes cannot compress on the delta fields. If this happens, as described for sending SEMICOMPRESSED_TCP, we will skip the next SEMICOMPRESSED_TCP. This will increase the compression ratio without increasing error propagation.

For the decompressor:

- Record or update the fields when receiving an UNCOMPRESSED_TCP packet and pass the packet to the upper layer.
- Recover the constant fields in a SEMICOMPRESSED_TCP packet and update the delta fields. If there is no UNCOMPRESSED_TCP received before this packet, store this packet and decompress it whenever an UNCOMPRESSED_TCP packet arrives.
- Recover the COMPRESSED_TCP packets based on the most recent received packet. If the decompressed packet is correct as indicated by the TCP checksum, update the delta fields.

From the compression method, it is clear that packet reordering will not make the compression malfunction. The only influence is that the length of delta values might be longer. The value of *windowSize* and *distance* are determined by the average BER of the link and the average packet size.

The higher the BER, the bigger the *windowSize* and the smaller the *distance*. On the other hand, the lower the BER, the smaller the *windowSize* and the bigger the *distance*. When $windowSize = \infty$ or $distance = 1$, all the packets are sent

as SEMICOMPRESSED_TCP, which is for links with a very high error rate. When *windowSize* becomes smaller, the COMPRESSED_TCP contains a shorter header, which will increase the compression ratio. But as the probability that all the packets in the *windowSize* got corrupted becomes high, the error propagation becomes severe. One thing to be noticed is that for a certain BER level and average packet length, there exists a proper *windowSize* value. This *windowSize* value will give good performance and increasing this value will not make much difference in the packet error rate. As *distance* grows, fewer SEMICOMPRESSED_TCP packets will be sent, which is suitable for links with lower BER. When *distance* decreases, more frequent SEMICOMPRESSED_TCP packets will refresh the state of the decompressor more often. This will give good decompression results even for higher BER.

Both *windowSize* and *distance* are influenced by the average packet size. For small packets, we can select a large *windowSize* and a large *distance*. A large *windowSize* decreases the error propagation, which can still yield a good compression ratio for small packets. Since the error propagation has been decreased by a large *windowSize*, we can then choose a larger *distance*, which in turn increases the compression ratio. Similarly, we can use small values of *windowSize* and *distance* for large packets.

3.4.3 Performance of Adaptive Header Compression

For our Adaptive Header Compression algorithm, define the error probability for the SEMICOMPRESSED_TCP to be q_s and let p_i be the error probability for packet i. Then, assuming no UNCOMPRESSED_TCP header is generated after the first *windowSize* packets, the error probability of every packet is as follows:

$p_i = q_o$ where $i < windowSize$ and $i = 2^j - 1$, $j = 1, 2, \cdots$

$p_i = q_s$ where $i < windowSize$ and $i \neq 2^j - 1$, $j = 1, 2, \cdots$

$p_i = q_s$ where $i \geq windowSize$ and $\frac{i-windowSize}{distance} = j$, $j = 0, 1, \cdots$

$p_i = q_c + \prod_{k=1}^{windowSize} q_{i-k} \times (1 - q_c)$ where $i \geq windowSize$ and $\frac{i-windowSize}{distance} \neq j$, $j = 0, 1, \cdots$

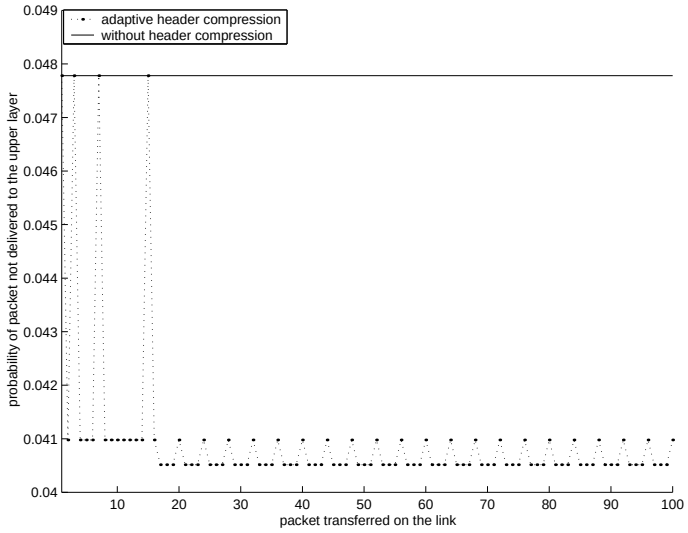


Figure 3.5: Packet error probability using Adaptive compression, $windowSize = 16$, $distance = 4$, $BER = 10^{-5}$

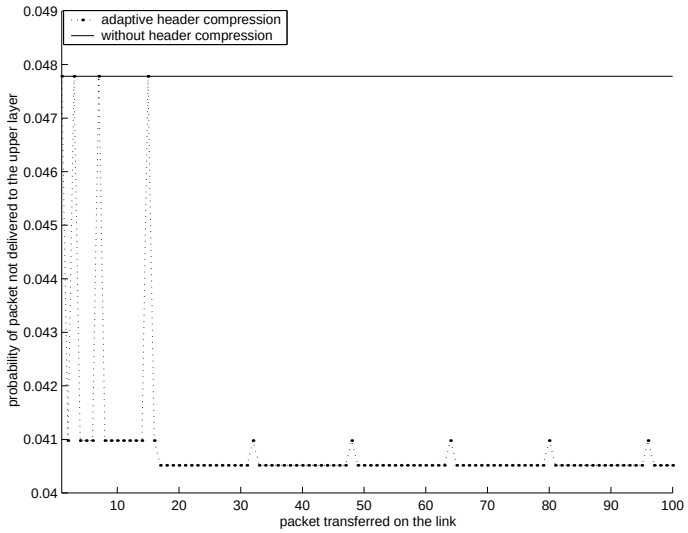


Figure 3.6: Packet error probability using Adaptive compression, $windowSize = 16$, $distance = 16$, $BER = 10^{-5}$

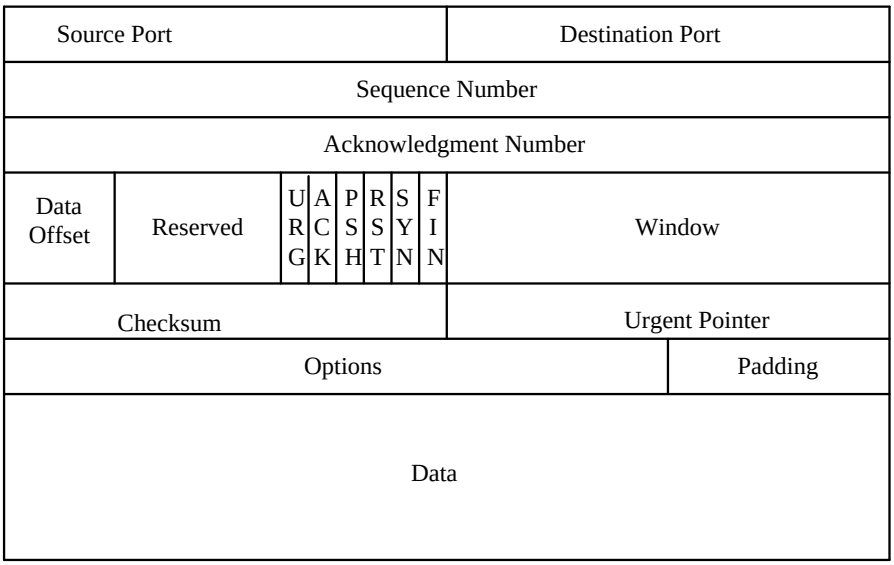
We give the calculated packet error probability in Figure 3.5 and Figure 3.6. These figures demonstrate the error probability for each packet sent from the compressor. We use a bit error rate of 10^{-5}, the length of an UNCOMPRESSED_TCP packet as 612 bytes, the length of a SEMICOMPRESSED_TCP packet as 523 bytes, and the length of a COMPRESSED_TCP packet as 517 bytes. We also assume no constant fields of the header have changed. In other words, all packets are sent using the compressed header when possible. The packet error probabilities are plotted in Figure 3.2 and Figure 3.4 when VJ, twice, and TAROC are used under the same conditions. Using the algorithm we proposed, the packet error rates are either q_o or q_s for the first $windowSize$ packets. After that, the error rate oscillates within a range. Periodically, a packet with original delta fields will be sent, which has a higher error probability. Between these packets, packets are sent using the compressed header, with almost a constant lower error probability. Figure 3.5 presents the situation when *distance* is selected as 4. Figure 3.6 is the situation with *distance* as 16.

It is clear that the average packet error probability is even smaller when *distance* becomes larger within a certain range. This seems to conflict with the design objective. However, recall that the formula was computed using the assumption that the errors are uniformly distributed. The graphs thus indicate the situation when the wireless link is in the *good* state. When the wireless link just exits the *bad* state, a smaller *distance* will refresh the state of the decompressor quicker and give better decompression results.

3.5 Conclusion

So far, we have discussed some recently observed computer network problems. These problems worsen the performance of existing header compression algorithms, which

are also described. Moreover, we provide a new algorithm, which can achieve better performance when used over wireless links and can address these problems. In the future, if we can find a proper model to describe packet reordering, the performance of existing header compression algorithms can then be analyzed. Moreover, there are various protocols for improving TCP performance on wireless networks [3] [7] [4]. The performance of header compression algorithms under all these protocols also need to be analyzed.



Source Port: 16 bits. The source port number.
Destination Port: 16 bits. The destination port number.
Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Acknowledgment Number: 32 bits. If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.
Data Offset: 4 bits. The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.
Reserved: 6 bits. Reserved for future use. Must be zero.
Control Bits: 6 bits (from left to right):
URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push Function
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender
Window: 16 bits. The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.
Checksum: 16 bits. The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.
Urgent Pointer: 16 bits. This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be to interpreted in segments with the URG control bit set.
Options: variable.
Padding: variable. The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

Figure 3.7: TCP Header

Chapter 4

Packet Error Statistics in Bursty Channels

4.1 Related Work

4.1.1 Mathematical Background

Stationarity

A discrete stochastic process is a set of values observed at discrete times [51] [34] [20] [25]. Each value is a probability variable itself. Define the variable at time t as $X(t)$. A stochastic process can be either stationary or nonstationary. There are two types of stationarity. A stochastic process is said to be strictly stationary if the joint distribution of $X(t_1), X(t_2), \ldots, X(t_n)$ is the same as the joint distribution of $X(t_1 + \tau), X(t_2 + \tau), \ldots, X(t_n + \tau)$ for all $t_1, t_2, \ldots, t_n, \tau$. In a less restricted way, a process is called second-order stationary (or weakly stationary) if its mean is constant and its autocovariance function depends only on the lag, so that $E[X(t)] = \mu$ and $Cov[X(t), X(t + \tau)] = \gamma(\tau)$. Considering both the first and second moment, second-order stationarity makes no assumptions about higher moments. This relaxed restriction makes second-order stationarity used more often when analyzing real world stochastic processes. Like many other publications [51] [34] [20], stationarity will be used to refer to second-order stationary in this dissertation unless otherwise stated.

Markov Chain

A Markov chain [20] [34] [51] is a special discrete time stochastic process. It has the property that the state at the current time is decided by only the immediately preceding one, and is not influenced by the state at any previous time. In formal terms, a process is said to be Markovian if $P(X(t_i) = x_i | X(t_1) = x_1, \ldots, X(t_{i-2}) =$

$x_{i-2}, X(t_{i-1}) = x_{i-1}) = P(X(t_i) = x_i|X(t_{i-1}) = x_{i-1})$ for all time points t_i and all states $x_1, \ldots, x_{i-2}, x_{i-1}, x_i$. This conditional probability is called the transition probability from x_{i-1} to x_i at time t_{i-1}. The matrix that contains all the transition probabilities at a certain time is called the transition matrix.

Markov chains can be further grouped into two types. If the transition probabilities do not change with time t, the Markov chain is described as having constant (stationary) transition probabilities, also called a time-homogeneous Markov chain. Otherwise, the Markov chain is usually called a general Markov chain.

An n-th order Markov chain is actually a semi-Markov chain where the current state depends only on the previous n states. Formally, a stochastic process is an n-th order Markov chain if and only if $P(X(t_i) = x_i|X(t_1) = x_1, \ldots, X(t_{i-n}) = x_{i-n}, \ldots, X(t_{i-1}) = x_{i-1}) = P(X(t_i) = x_i|X(t_{i-n}) = x_{i-n}, \ldots, X(t_{i-1}) = x_{i-1})$ for all t_i and all states $x_1, \ldots, x_{i-n}, \ldots, x_{i-1}, x_i$.

Runs Test

When we need to analyze a random process and we do not have much knowledge about the underlying mathematical model, some statistical procedures assume a specific distribution function, e.g., a Gaussian distribution. This probably can be used with reasonable accuracy for many random variables that do not fit this particular distribution. However, there is no clear measure as to how much a random variable may deviate before this distribution is no longer valid. This problem can often be avoided by using distribution-free (nonparametric) procedures. The Runs test [5] is one of these procedures. Consider a sequence of N observed values of a random variable, where each observation is classified into one of two mutually exclusive categories. A run is defined as a sequence of identical observations that is followed and preceded by a different observation or no observation at all. The number of runs that occur

in a sequence of observations gives an indication as to whether or not the results are independent random observations of the same random variable.

Average Mutual Information

Average mutual information is calculated based on mutual information [22]. Let a_k and b_j denote two events belonging to two different event spaces. Mutual information, $I_{X|Y}(a_k; b_j)$, is defined as $\log \frac{P_{X|Y}(a_k|b_j)}{P_X(a_k)}$, which is the information provided about the event $X = a_k$ by the occurrence of the event $Y = b_j$. It actually uses the logistical value of the ratio of a posteriori to a priori probability to evaluate the relationship. Average mutual information, $I_{X|Y}(X;Y)$, is defined as $\sum_{a \in X, b \in Y} I_{X|Y}(a;b)$. $I_{X|Y}(a_k; b_j)$ is usually denoted simply as $I(a_k; b_j)$ and $I_{X|Y}(X;Y)$ as $I(X;Y)$.

4.1.2 Previous Work on Wireless Channel Error Models

Bit Level Model

Markov chain was first used to model bit errors in [24]. Gilbert modeled the channel as in two states, *Good* state and *Bad* state. When the channel is in the *Good* state, all the bits are transferred correctly; the channel is equal to the perfect channel. When the channel is in the *Bad* state, the channel is a binary symmetric channel. The bits transmitted in this state will suffer a specific bit error rate. The Gilbert model was modified by Elliott [18] where the *Good* state is also modeled as a binary symmetric channel.

Under the Gilbert-Elliott mode, the wireless channel is then modeled as a discrete time Markov chain (DTMC). The channel can be assumed as having two states. One is the *Good* state; the other is the *Bad* state. These two events are denoted as g and b, respectively. As shown in Figure 4.1, at any time, the probability of the next channel state is determined by only the current state and it has no relationship with

any previous state. The value of the transition matrix, P_{bg}, P_{gg}, P_{gb}, and P_{bb}, can be calculated according to the channel features or from the real world tracing results. No matter which state the channel is in, errors occur according to an independent and identical distribution (IID) model. This means that the bits sent over the wireless channel are facing a certain bit error rate (BER) to be corrupted, where the value of BER is determined by the channel state. When the channel is in the *Good* state, the Bit Error Rate, BER_g, is low. While the BER for the *Bad* state, BER_b, is high.

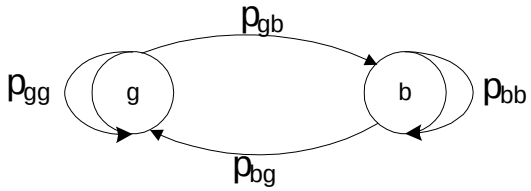


Figure 4.1: Markov Model for Wireless Link

Some previous work [61] [65] uses the average mutual information relationship to demonstrate that the first order Markov chain is accurate enough to describe the channel fading state. Let R_i, R_{i-1}, and R_{i-2} denote the random variable of a stochastic process at time i, $i-1$, and $i-2$. One property of the average mutual information is $I(R_i; R_{i-1}R_{i-2}) = I(R_i; R_{i-1}) + I(R_i; R_{i-2}|R_{i-1})$. The meaning of this equation is that the average mutual information among R_i, R_{i-1}, and R_{i-2} equals the addition of that between R_i and R_{i-1} and the conditional average mutual information between R_i and R_{i-2} given R_{i-1}. As pointed out in [61], the wireless channel state can be described as a higher order Markov chain [59] [62]. Then, this property, especially the ratio between $I(R_i; R_{i-1})$ and $I(R_i; R_{i-2}|R_{i-1})$, can be used to compare the accuracy of first order and second order Markov models. When $\frac{I(R_i;R_{i-2}|R_{i-1})}{I(R_i;R_{i-1})}$ is much less than 1, $I(R_i; R_{i-1})$ itself can be used to represent $I(R_i; R_{i-1}R_{i-2})$. When

this point holds, the second order Markov chain model can be simplified to a first order Markov chain model without much loss in accuracy. Discussing the average mutual information of the signal-to-noise ratio (SNR) at a wireless channel receiver, Wang et al. [61] prove that a wireless channel can be modeled as a first order Markov chain. Zorzi et al. [65] use a similar method and prove the first order Markov chain from the aspect of the received signal power. Note that [65] is discussing the packet level statistics. However, the channel is assumed as changing slowly compared to the transmission period of a packet so that no channel state transition is allowed within any packet. Since this requirement is actually true for bit level model, we may also treat this work as analyzing bit level statistics.

Packet Level Model

Given that the wireless channel can be modeled using the Gilbert-Elliott model, Zorzi et al. use the average mutual information relation again and prove that the proper model for packet level statistics is also a first order Markov chain [64]. When considering the packet level statistics, R_i is used to denote the probability variable of the state of packet i, i.e., R_i can be either *packet is correct* or *packet is in error*.

There are also papers that analyze the packet error probability based on tracking results [43] [35]. The conclusion in these papers is that the packet level statistics cannot be modeled simply as a Markov chain. In [43], the cumulated distribution function (CDF) is calculated from the tracking results and compared with that of a simple two state Markov chain. Then some modifications on the Markov chain are given to represent the packet level statistics. In [35], the Runs test is conducted on a real world tracking result, which shows that the packet level statistics are not stationary. Some statistical parameters are found to divide the tracking result into two series. One series contains only corrupted packets and the other one contains

both corrupted and correct packets, which could actually be modeled as a simple Markov chain. The two series could be simulated with ease and then the packet level situation is simulated by their combination.

The difference between the theoretical analysis and the experimental results is quite obvious. Even though the experimental results are produced when error correction is adopted, the difference still inspired us to question why there is such a huge difference. Our research result is shown in the following sections.

4.2 Analysis of Packet Error Statistics

4.2.1 Calculate the Conditional Packet Error Probability

In order to calculate the packet error probability, we need to define three stochastic processes. The first one, $X(i), i = 0, 1, 2, \ldots$, is the channel status sampled for each bit sent over the channel. The event set for X is $\{g, b\}$, where g stands for the *Good* channel status and b for the *Bad* one. Note that the bits sent over the channel may or may not belong to the same connection. X is a Markov chain and we have

$$\begin{aligned} P_X(X(i) = g|X(i-1) = g) &= p_{gg} \\ P_X(X(i) = b|X(i-1) = g) &= p_{gb} \\ P_X(X(i) = g|X(i-1) = b) &= p_{bg} \\ P_X(X(i) = b|X(i-1) = b) &= p_{bb} \end{aligned}$$

for all i.

The second process is the bit error/error-free status observed for each bit sent over the channel, $Y(i), i = 0, 1, 2, \ldots$. The bit status is either 0, which means that the bit is transferred correctly, or 1, bit not correct. We have the following relationship between X and Y:

$$\begin{aligned} P_Y(Y(i) = 0|X(i) = g) &= 1 - BER_g \\ P_Y(Y(i) = 1|X(i) = g) &= BER_g \\ P_Y(Y(i) = 0|X(i) = b) &= 1 - BER_b \\ P_Y(Y(i) = 1|X(i) = b) &= BER_b \end{aligned}$$

Finally, the third process is what we are most interested in, the packet status $Z(j), j = 0, 1, 2, \ldots$. The possible events of Z are error or error-free, denoted by A and B, respectively. Denote the length of packet j as l_j, and packet j as composed of bits $b_{l_j}, b_{l_j}+1, \ldots, b_{l_j}+l_j-1$. A packet is assumed to be correct only when all the bits it contains are transferred correctly. So we have

$$P(Z(j) = A) = \prod_{i=b_{l_j}}^{b_{l_j}+l_j-1} P(Y(i) = 0)$$

After defining the three processes, we can then calculate the packet error probability given the channel state when the first bit of a packet is sent. When a packet is being sent, the channel state could change at the edge of every bit. For each packet, we can calculate the packet error probability given the channel state when the first bit of a packet is sent. In this dissertation, we will refer to this channel state as the initial channel state of one packet. Remember the channel state could change at the edge of every bit. After knowing the initial channel state of packet j, the channel states for all other $l_j - 1$ bits can be either *Good* or *Bad*. So the total number of channel state series for the packet is 2^{l_j-1}. Given channel state transition probabilities p_{gg}, p_{gb}, p_{bb}, and p_{bg}, which are actually conditional probabilities such that $p_{gg} + p_{gb} = 1$, and $p_{bb} + p_{bg} = 1$, we can calculate the probability for each channel state series.

After calculating the probability of each channel state series by the property of process X, we can further get the error/error free status of each bit. Considering the error probability of each bit, the total probability theorem shall be used to calculate the packet error/error-free probability. The formula for the packet error/error-free

probability given the initial channel state is *Good* is expressed as follows:

$$\begin{cases} P(Z(j)=A|X(b_j)=g) & = & \sum\limits_{\substack{i=b_{l_j}+1,\ldots,b_{l_j}+l_j-1 \\ x_i\in\{g,b\}}} \phi(g, x_{b_{l_j}+1}, \ldots, x_{b_{l_j}+l_j-1}) \\ & where & \phi(x_0, x_1, \ldots, x_{l_j-1}) = P_Y(0|g)\prod\limits_{i=1}^{l_j-1} P_Y(0|x_i)P_X(x_i|x_{i-1}) \\ P(Z(j)=B|X(b_j)=g) & = & 1-P(Z(j)=A|X(b_j)=g) \end{cases} \tag{4.1}$$

A simple expression for $P(Z(j)=A|X(b_j)=g)$ is:

$$[1 \quad 0] \begin{bmatrix} p_{gg}(1-BER_g) & p_{gb}(1-BER_g) \\ p_{bg}(1-BER_b) & p_{bb}(1-BER_b) \end{bmatrix}^{l_j-1} \begin{bmatrix} 1-BER_g \\ 1-BER_b \end{bmatrix} \tag{4.2}$$

Similarly, the packet error-free probability given a *Bad* initial channel state can be calculated using the simple expression:

$$[0 \quad 1] \begin{bmatrix} p_{gg}(1-BER_g) & p_{gb}(1-BER_g) \\ p_{bg}(1-BER_b) & p_{bb}(1-BER_b) \end{bmatrix}^{l_j-1} \begin{bmatrix} 1-BER_g \\ 1-BER_b \end{bmatrix} \tag{4.3}$$

For each channel state series, the initial channel state for the following packet can also be analyzed. Having this information, the statistics for the following packet can also be calculated using the same method.

Thus, knowing the initial channel state of one packet and using this method, we can calculate the error/error-free probability of this packet as well as the probability of any combination of this packet and its following packets. After finding these values, further statistical analysis results can be calculated.

4.2.2 Time-inhomogeneous Markov Chain

Average Mutual Information Given Different Initial Channel State

Packet level error statistics have been proved to be approximated by a first order Markov chain in [64] through the relationship between average mutual information. However, this proof might be sufficient only for a generic Markov model, which does not verify that it should be a Markov chain with constant transition probabilities.

From the analysis in Section 4.2.1, it is clear that the initial channel state actually determines the error probability of one packet. It also determines the probability of the initial channel state for the following packet, from which the probability for that packet can be specified. Since the initial channel state of one packet can decide the error probability of itself as well as the following packet, it actually specifies the transition probability of the packet level statistics, which may not be the same for different initial states. When one packet is specified, its initial channel state is fixed, either *Good* or *Bad*. However, in the proof where the average mutual information is used, both initial channel states are combined together through the average mutual information, so the difference among transition probabilities is thus hidden and ignored.

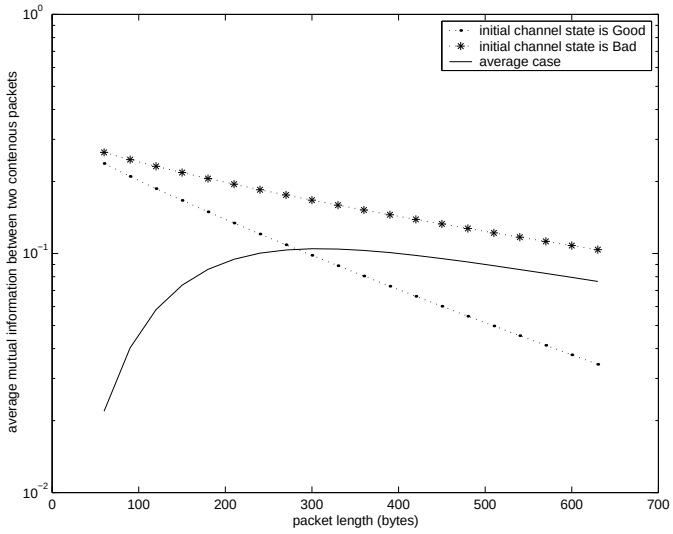


Figure 4.2: Comparison of Average Mutual Information

We calculate and compare the values of the average mutual information, $I(R_i; R_{i-1})$, for different initial channel states. Suppose the sender has enough packets and the packets are transmitted on the wireless channel periodically, which means that the time distance between any two adjacent packets is a fixed value. In Figure 4.2,

we plot the average mutual information when the packets are transmitted continuously, i.e., after each packet the following packet will be sent out immediately. The bit level model parameters are set as the car speed model used in [38], i.e., $p_{gb} = 0.0000167, p_{bg} = 0.00166, BER_g = 0.00001, BER_b = 0.01$. For all the figures shown in Chapter 4, we use this set of parameters.

As shown in Figure 4.2, the average mutual information for *Good* and *Bad* initial channel states and the average situation are quite different. It is true that the bit level statistics will influence the packet level analysis results. So the differences among average mutual information may not appear under certain circumstances. However, for either the difference in average mutual information or all other results in Chapter 4, we obtained similar results while using almost all possible parameter combinations. And these results hold for bit level parameters used in [38] for common wireless communication environments as well as the Gilbert-Elliott model used in [64].

When the packet is shorter, the average mutual information between the *Good* and *Bad* initial channel states is not big. However, the average case differs a lot from other situations. This actually confirms that using the average case may not be the appropriate choice. When one packet is sent, the initial channel state is either *Good* or *Bad*. And the probability on the following channel states differs a lot when we just consider the following short range of time. So combining these two cases together does not make much sense, especially when the packet length is not big enough, i.e., not approaching infinity. On the other hand, packets with too many bits are prone to be corrupted and are not realistic.

Transition Matrix Given Different Initial Channel State

The difference in average mutual information makes us question whether this process is a Markov chain with constant transition probabilities or not. In order to verify

this, we calculate the transition matrix of the packet level stochastic process when the bit level errors follow a Markov chain model.

We now give the transition matrices of the packet level statistics. Remember that the initial channel state of one packet actually determines the transition probability from an error/error-free packet to an error/error-free packet. The transition matrix given the initial channel state is *Good* is:

$$\begin{bmatrix} PZ_{AA|g} & PZ_{AB|g} \\ PZ_{BA|g} & PZ_{BB|g} \end{bmatrix}$$
$$= \begin{bmatrix} P(Z(j+1)=A|Z(j)=A, X(b_j)=g) & P(Z(j+1)=B|Z(j)=A, X(b_j)=g) \\ P(Z(j+1)=A|Z(j)=B, X(b_j)=g) & P(Z(j+1)=B|Z(j)=B, X(b_j)=g) \end{bmatrix}$$

Each item in this matrix can be calculated following the methods described in Section 4.2.1. For example, we have

$$PZ_{AA|g} = \frac{P(Z(j+1)=A, Z(j)=A|X(b_j)=g)}{P(Z(j)=A|X(b_j)=g)} \tag{4.4}$$

$$\begin{aligned} & P(Z(j+1)=A, Z(j)=A|X(b_j)=g) \\ = \; & [1 \quad 0] \begin{bmatrix} p_{gg}(1-BER_g) & p_{gb}(1-BER_g) \\ p_{bg}(1-BER_b) & p_{bb}(1-BER_b) \end{bmatrix}^{l_j+l_{j+1}-1} \begin{bmatrix} 1-BER_g \\ 1-BER_b \end{bmatrix} \end{aligned} \tag{4.5}$$

and $P(Z(j)=A|X(b_j)=g)$ can be calculated as in Expression (4.2).

Obviously, $PZ_{AB|g} = 1 - PZ_{AA|g}$. $PZ_{BA|g}$ and $PZ_{BB|g}$ can be calculated similarly. Using this idea, we can also get the transition matrix when the initial channel state is *Bad*.

In Figure 4.3, the transition probabilities of one correct packet given that the previous one is also correct are plotted. The initial channel status of one packet could be either *Good* or *Bad*. Transition probabilities for both situations are given. From this figure, it is clear that the transition probabilities are different. This feature can be explained as a property of the Markov chain as follows. If at one time a packet is correct with a *Good* initial state, the transition probability is different from the

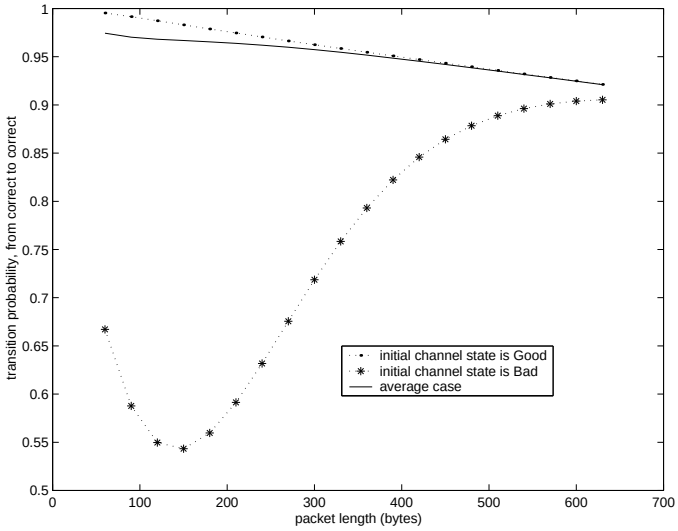


Figure 4.3: Comparison of Transition Probability, from Correct Packet to Another Correct Packet

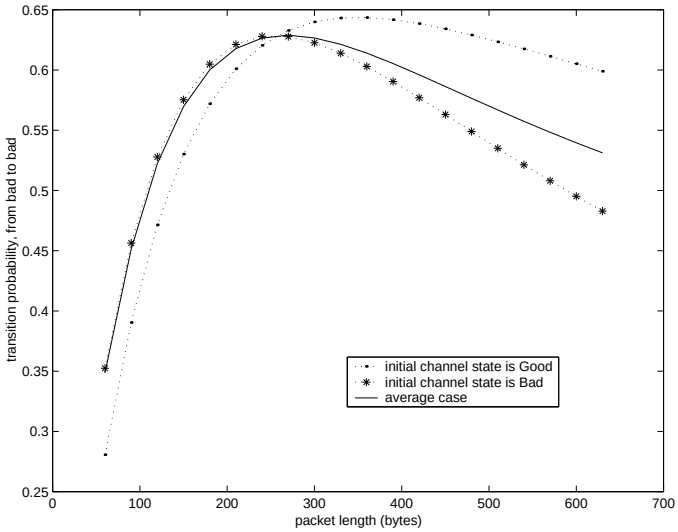


Figure 4.4: Comparison of Transition Probability, from Corrupted Packet to Another Corrupted Packet

case where the initial state is *Bad* and the packet is correct. Similarly, Figure 4.4 demonstrates that the transition probabilities from a corrupted packet to another corrupted one are also different.

According to the definition of the time-homogeneous Markov chain [20] [34] [51], the packet level statistics can be modeled only as a general Markov chain without constant transition probabilities. If it is assumed to be time-homogeneous and the transition matrix under the average case is thus used for evaluating communication performance, we may not reach the correct conclusion.

4.2.3 Stationarity of the Packet Error/Error-free Process

In [35], whether or not the packet error statistics are stationary is analyzed and the result actually determines how to simulate this process. So we also want to discuss this issue here. Even though in Section 4.2.2 we reach the conclusion that the packet level error statistics can be modeled as a time-inhomogeneous Markov chain, the stationarity feature cannot be determined simply from this conclusion. Remember that the property of a Markov chain is the conditional probability given the present state. A Markov chain, with constant transition probabilities or without, cannot be mapped directly to the property of stationarity [34]. Using the calculation method described in Section 4.2.1, we are able to calculate the mean and the autocovariance function for the packet level process. However, we use the Runs test to test the stationarity as in [35]. In real world conditions, it usually can be assumed that any nonstationarity of interest will be revealed by time trends in the mean value of the data. For the packet level error statistics, we can assume that the nonstationarity can be revealed by the run length of erroneous packets.

The first stage is to collect packet transmission statuses. We simulate them by using Matlab:

- Use the Markov chain model to generate series of the channel states.
- Use the IID model to simulate whether every bit transmitted in each state is

correct or not.

- For any packet, whether it is correct or not is determined by all the bits it contains. The packet is correct only when all the bits are transmitted correctly.

The uniform random number generator in Matlab 5.3 [42] uses a lagged Fibonacci generator, with a cache of 32 floating point numbers, combined with a shift register random integer generator. Starting with a *Good* channel state, we simulate $2,5000$ packets that are transferred continuously.

Similar to [35], the procedure for testing packet error stationarity is described as follows:

- Divide the sample record into N equal time intervals where the data in each interval may be considered independent.
- Count the run lengths in each interval.
- Compute a mean value for each interval and align these sample values in a time sequence.
- Test the sequence of mean values for the presence of underlying trends or variations other than those due to expected sampling variations.

Then, the same procedure as [35] is followed and the Runs test result is plotted in Figure 4.5, where the packet length is 30 bytes. Using 0.05% left and right tail cut-offs and comparing with the features a stationary stochastic process should have [5], the packet level error statistics are not stationary.

From both the theoretical and simulation results, we reach the conclusion that even if the packet error/error-free process can be modeled as a Markov chain, the transition matrix is decided by the initial channel state of the present packet and

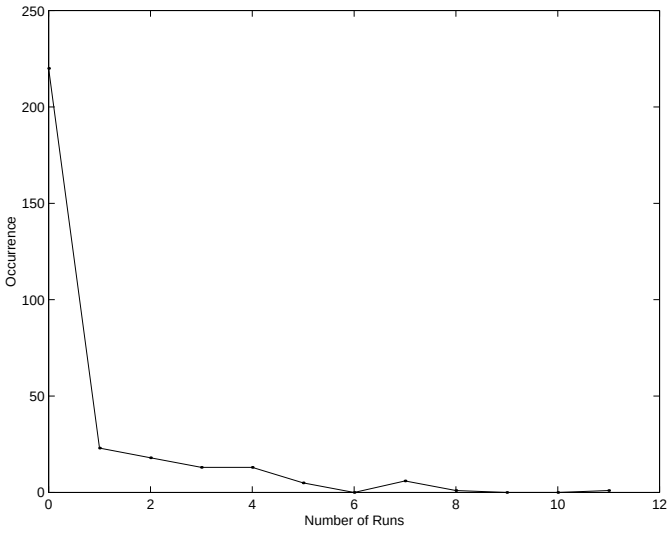


Figure 4.5: Run Test result

the values differ with different initial channel states. This Markov chain is not time-homogeneous and thus the average value of the transition matrix may not be an appropriate solution. We also reach the conclusion that the packet error/error-free process is not stationary.

4.3 Packet Error Model

4.3.1 Hidden Markov Chain Model

The Hidden Markov Model (HMM) [48] [8] [23] has been used a lot in many recognition problems where we are required to discover the underlying structure in a sequence of observed symbols. Some examples are speech recognition, optical character recognition, part-of-speech disambiguation, gesture recognition from video sequences, and finding structural motifs in DNA seqences. A HMM is a Markov chain, where each state generates an observation. A HMM is a *hidden* Markov model because we cannot see the states of the Markov chain, but just a function of them. We see only the observations, and the goal is to infer the hidden state sequence. HMMs are very useful for

time-series modelling, since the discrete state-space can be used to approximate many non-linear, non-Gaussian systems. The basic idea is to find a function/expression of the observation which forms a Markov Chain.

Even though HMM is usually used to solve some statistical processes that are not Markov chains, we can still use it to solve the packet error statistics in bursty channels. Although the packet error/error-free statuses we observed in bursty channels actually form a time-inhomogeneous Markov chain, this model does not help a lot for our further discussion on connection performance. From the transition matrices, which depend on the initial channel state, it is still hard to get some statistical properties of this process. For example, in order to discuss the connection performance, we may want to know the average packet error rate and the average packet error/error-free cluster length. From the transition matrices, it is hard to analyze these properties. Following the ideas of HMM, we can try to find a function of the states that we can observe. And the output of the function actually forms a time-homogeneous Markov chain.

Combining the initial channel state and the packet status together, we produce a new stochastic process, Z'. The event space of Z' is gA, gB, bA, and bB. As described in Section 4.2.2, the initial channel state of one packet can decide the status of this packet; it can also decide the status of the following packet. So $P(Z'_i|Z'_{i-1})$ has the same value for all valid i. This property indicates that the combined events actually form a time-homogeneous Markov chain. As long as we can find the transition matrix of this Markov chain, we can achieve some further statistical analysis results. Notice that after combining the packet initial channel state and the packet error/error-free status, we expand the event number of the statistical process. When we consider only the packet error/error-free status, the event set is A, B, and the transition matrix is a

2-by-2 matrix. When we combine the initial channel state and the packet error/error-free status, the event set is gA, gB, bA, bB, and the transition matrix is a 4-by-4 matrix. Using HMM to divide the packet process can give us more details about the phenomenon. Just by adding more details, the process is changed from a time-inhomogenous Markov chain to a time-homogeneous Markov chain.

The transition matrix of Z' is:

$$\begin{bmatrix} PZ'_{gAgA} & PZ'_{gAgB} & PZ'_{gAbA} & PZ'_{gAbB} \\ PZ'_{gBgA} & PZ'_{gBgB} & PZ'_{gBbA} & PZ'_{gBbB} \\ PZ'_{bAgA} & PZ'_{bAgB} & PZ'_{bAbA} & PZ'_{bAbB} \\ PZ'_{bBgA} & PZ'_{bBgB} & PZ'_{bBbA} & PZ'_{bBbB} \end{bmatrix}$$

We give the formulas for PZ'_{gAgA} and PZ'_{gAgB}. All other probabilities can be calculated similarly.

$$PZ'_{gAgA} = \frac{P(Z(j+1) = A, X(b_{j+1}) = g, Z(j) = A | X(b_j) = g)}{P(Z(j) = A | X(b_j) = g)} \tag{4.6}$$

In this formula, $P(Z(j) = A|X(b_j) = g)$ is given in Expression (4.2) and

$$\begin{aligned} & P(Z(j+1) = A, X(b_{j+1}) = g, Z(j) = A | X(b_j) = g) \\ = \; & [1 \quad 0] \begin{bmatrix} p_{gg}(1-BER_g) & p_{gb}(1-BER_g) \\ p_{bg}(1-BER_b) & p_{bb}(1-BER_b) \end{bmatrix}^{l_j - 1} \begin{bmatrix} p_{gg}(1-BER_g) & 0 \\ p_{bg}(1-BER_b) & 0 \end{bmatrix} \times \\ & \begin{bmatrix} p_{gg}(1-BER_g) & p_{gb}(1-BER_g) \\ p_{bg}(1-BER_b) & p_{bb}(1-BER_b) \end{bmatrix}^{l_{j+1} - 1} \begin{bmatrix} 1-BER_g \\ 1-BER_b \end{bmatrix} \end{aligned} \tag{4.7}$$

Formula for calculating PZ'_{gAgB} is:

$$PZ'_{gAgB} = \frac{P(X(b_{j+1}) = g, Z(j) = A | X(b_j) = g)}{P(Z(j) = A | X(b_j) = g)} - PZ'_{gAgA} \tag{4.8}$$

where

$$\begin{aligned} & P(X(b_{j+1}) = g, Z(j) = A | X(b_j) = g) \\ = \; & [1 \quad 0] \begin{bmatrix} p_{gg}(1-BER_g) & p_{gb}(1-BER_g) \\ p_{bg}(1-BER_b) & p_{bb}(1-BER_b) \end{bmatrix}^{l_j - 1} \begin{bmatrix} p_{gg}(1-BER_g) \\ p_{bg}(1-BER_b) \end{bmatrix} \end{aligned} \tag{4.9}$$

Given the transition matrix of Z', we can calculate the stationary distribution of each state, i.e., gA, gB, bA, and bB. From the stationary distribution, the packet

error/error-free probability can be calculated easily. The packet error probability is $gB + bB$ and the packet error/free probability is $gA + bA$. However, the HMM model may not be suitable for analyzing other statistical properties. For example, when we need to analyze the performance of GBN protocol, we need to calculate the probability that one packet is corrupted, then no matter what the states of the following packets are, the packet a certain slots later is correct. When using HMM to calculate this probability, too many calculations are needed. When the calculation complexity cannot be tolerated, we need another model in order to analyze the connection performance.

4.3.2 Gap Model

Another way to discuss packet level error statistics could be adopting the gap model [40]. Under any situation, two consecutive correct packets are separated by an arbitrary number of corrupted packets. The gap model uses the probability for the number of these corrupted packets to describe the block error statistics. The gap model actually describes the property of a stochastic process through analyzing the probability of different lengths of corrupted packet runs. In the transport layer protocols, the corrupted packet runs actually determine the flow control. Hence, the gap model is expected to describe the packet flow successfully.

In the gap model, $P(j)$ is defined as the probability that there exist $j-1$ corrupted packets between two consecutive correct ones. And the expression for $P(j)$ is given as:

$$P(j) = P(Z_{i+1} = B, Z_{i+2} = B, \ldots, Z_{i+j-1} = B, Z_{i+j} = A | Z_i = A) \ ,$$

where $j = 1, 2, 3, \ldots$.

Similarly, $Q(j)$ is defined as the probability that there exist $j - 1$ correct packets

between two consecutive corrupted ones. And the expression for $Q(j)$ is given as:

$$Q(j) = P(Z_{i+1} = A, Z_{i+2} = A, \ldots, Z_{i+j-1} = A, Z_{i+j} = B | Z_i = B) \ ,$$

where $j = 1, 2, 3, \ldots$.

From the analysis in Section 4.2, it is clear that the initial channel state of the first correct packet, packet i, will determine the value of $P(j)$ and $Q(j)$. So we actually have two sets of $P(j)$ and $Q(j)$:

$$\begin{cases} P(j|g) &= P(Z_{i+1} = B, Z_{i+2} = B, \ldots, Z_{i+j-1} = B, Z_{i+j} = A | Z_i = A, X_{b_i} = g) \\ P(j|b) &= P(Z_{i+1} = B, Z_{i+2} = B, \ldots, Z_{i+j-1} = B, Z_{i+j} = A | Z_i = A, X_{b_i} = b) \end{cases} \tag{4.10}$$

where $j = 1, 2, 3, \ldots$.

$$\begin{cases} Q(j|g) &= P(Z_{i+1} = A, Z_{i+2} = A, \ldots, Z_{i+j-1} = A, Z_{i+j} = B | Z_i = B, X_{b_i} = g) \\ Q(j|b) &= P(Z_{i+1} = A, Z_{i+2} = A, \ldots, Z_{i+j-1} = A, Z_{i+j} = B | Z_i = B, X_{b_i} = b) \end{cases} \tag{4.11}$$

where $j = 1, 2, 3, \ldots$.

From the formulas given in Section 4.2.1, it is easy to calculate the value of $P(j|g)$, $P(j|b)$, $Q(j|g)$, and $Q(j|b)$. After finding the probabilities in the gap model, we can further analyze the communication performance.

4.3.3 Packet Error/Error-free Length Distribution

One application of the gap model is to calculate the distribution of error/error-free packet lengths, which were obtained from tracking results in both [43] and [35]. The probability that the packet error/error-free cluster length is j, given that the cluster exists, can be calculated from the gap model, i.e., from $P(j|g)$, $P(j|b)$, $Q(j|g)$, and $Q(j|b)$. The probability that the packet error cluster length is j is:

$$P_X(g) \times P(j+1|g) + P_X(b) \times P(j+1|b) \tag{4.12}$$

where $j = 1, 2, 3, \ldots$.

$P_X(g)$ and $P_X(b)$ are the stationary distributions that the channel is in *Good* and *Bad* states, respectively. These values are thus the global probability that the channel is in the corresponding state.

Similarly, the probability that the packet error-free cluster length is j is:

$$P_X(g) \times Q(j+1|g) + P_X(b) \times Q(j+1|b) \tag{4.13}$$

where $j = 1, 2, 3, \ldots$.

The CDF functions can thus be calculated from Expressions (4.12) and (4.13). The results are plotted in Figure 4.6 and Figure 4.7, respectively, for a packet length of 30 bytes. In order to compare the importance of using an appropriate packet error model, we also give the distributions when the time-homogeneous Markov chain is adopted. The results show a big difference from those when the gap model is used. We also use the simulator described in Section 4.2.3 to simulate the packet status and plot the distribution of error/error-free packet length in Figure 4.6 and Figure 4.7, respectively. It is clear that only the gap model matches the simulation results. And the match is expected to be even better as the simulation duration increases.

There are several features that can be observed from Figure 4.6 and Figure 4.7. One feature noticed from the figures is that the distribution of packet error-free length is quite different from that of the packet error length. And the average packet error-free length is much larger than the average packet error length. The second feature is that the time-homogeneous Markov chain can describe only the average packet error/error-free rate successfully. It cannot give a correct image of the consecutive corrupted/correct packets, which in turn will introduce incorrect analysis results into the communication performance. Finally, if the distributions of packet error/error-

free length are calculated from real world tracking results and a Markov chain model is assumed, the transition probability can be calculated from the best-match geometric distribution. However, this model may not even describe the average packet error/error-free rate successfully. This point has been verified in [35].

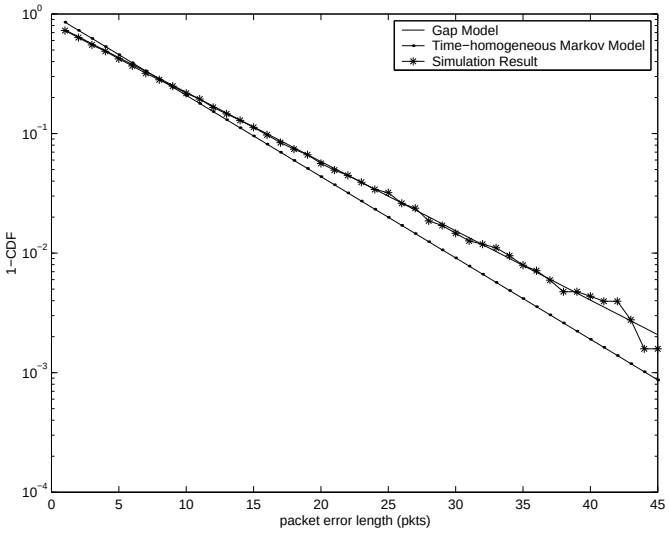


Figure 4.6: Distribution of Packet Error Length

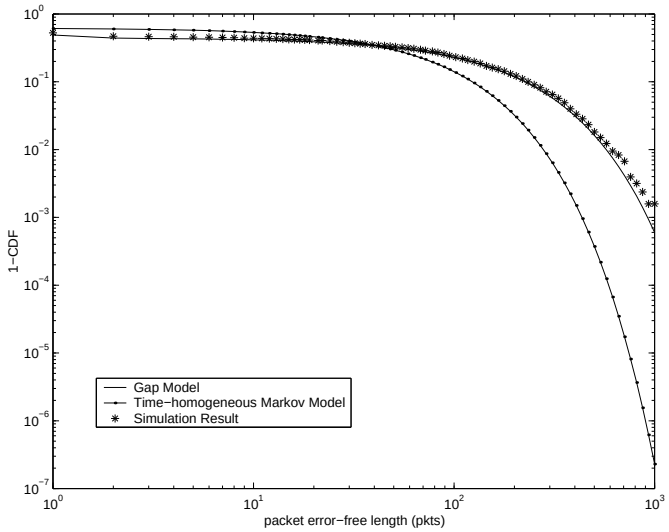


Figure 4.7: Distribution of Packet Error-free Length

4.4 Conclusion

So far, we have reached the conclusion that the packet error statistics can be modeled only as a time-inhomogeneous Markov chain and it is not stationary. Moreover, both the HMM model and the gap model are proposed as better models for packet error statistics analysis. Finally, the distribution of packet error/error-free cluster length is selected as one example and the comparison between the gap model and the time-homogeneous Markov chain is given. Simulation results are also given to verify the theoretical analysis results.

After defining the appropriate models, we can further use the models to compare communication performance with different packet lengths and with different error recovery mechanisms. In addition, the energy issue can also be discussed using this model in some energy-constrained wireless environments, such as wireless sensor networks and mobile ad-hoc networks.

Chapter 5

ARQ Performance in Bursty Channels

5.1 Related Work

5.1.1 ARQ Error Control

In Chapter 5, we focus on the discussion of connection performance when automatic repeat request (ARQ) error correction is used. ARQ is one of the error correction mechanisms used in computer networks. In ARQ, errors are detected by the use of some simple error detection codes and erroneous packets are retransmitted until they succeed. Positive acknowledgments (ACKs) or negative acknowledgments (NACKs) are returned by the receiver via a feedback channel to inform the sender whether a previously transmitted packet needs a retransmission.

There are three basic ARQ protocols: stop-and-wait (SW), go-back-n (GBN), and selective-repeat (SR). In SW, after sending one packet, the sender will stop and wait for the acknowledgment or timeout. It can send the next packet only when the receiver has received the current one correctly. In GBN, the sender will continue sending packets until it receives a NACK or there is timeout on the acknowledgment for one packet, both indicating the packet was not received successfully. Then the sender will resend this packet and all the following packets that have been sent out so far regardless of whether they have been received correctly or not. Whereas in SR, the sender needs to retransmit only the erroneous packets. Since SW can be analyzed as a special case of SR, we will not discuss SW further. SR shows the throughput upper limit in ARQ [40] at the expense of an infinite buffer at both the sender as well as the receiver and unavoidable delay at the receiver. The delay arises from the fact that data packets may arrive at the receiver out of order so that the receiver has

to buffer some packets and deliver them only when the preceding missing packets are retransmitted correctly.

There are several reasons that we conduct the performance analysis on ARQ. One of the reasons is that ARQ is the error correction protocol used in some Medium Access Control (MAC) protocols. For example, IEEE 802.11 [15] uses an Ethernet-like stochastic and distributed mechanism - Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA). And it uses an immediate NACK to recover from transmission errors. Moreover, in wireless communication, transceivers tend to limit the power for signal transmission so the distance between the transmitter and the receiver is expected to decrease. Thus the sender can receive the ACK or NACK more quickly and ARQ error correction is expected to be more efficient. Moreover, the delay introduced by ARQ protocols also decreases in this situation.

5.1.2 Optimal Packet Length

Finding the optimal packet length for wireless channels is not a new topic and it has been studied from various aspects. In [55], the authors discuss optimal packet lengths in wireless channels with Raleigh fading. Using mathematical channel models, data throughput is used as the performance criterion. Whether a bit is transferred correctly or not is determined by comparing the signal-to-noise ratio with a threshold. Since no error randomness is considered when the SNR is below or above the threshold, the bit error model is different from the common model for wireless channels.

Communication performance when using different packet lengths is also studied in [37]. A Markov chain is adopted to model the bit error statistics. At the same time, the channel state is assumed to be not changing within each packet so that the bit error probability of the first bit is applied to all the other bits in one packet. And the channel state for the first bit is taken as the stationary distribution of the Markov

chain. The average packet error rate is thus calculated and the connection performance is analyzed accordingly. However, the average packet error rate calculated in this way may not reflect the real world situations correctly. Moreover, when using the average packet error rate, each packet is assumed to have the same and independent error rate. The packet error statistics then follows IID model. The relationship between packet error occurrences is thus omitted.

Besides the theoretical analysis on the optimal packet length, some experimental and simulation results are also given in previous work. In [1], the communication between two laptops is set using different packet lengths and the performance is monitored. Since no quantitative model of the bit error statistics is given, the result is valid for only one specific environment. Selecting different packet lengths is also evaluated in [16] while the protocol harmonization between several layers is discussed and simulated. A Markov chain is adopted as the bit error model and the transition matrix is fixed. Transmission power will then influence BER_g and BER_b, which will influence higher layer performance and the total power consumed by the connection. Through simulation, the optimal packet length is then obtained to provide the best power efficiency. Since a fixed transition matrix is used at the bit level, the conclusion is again valid for only that specific environment. Perhaps [63] is closest to the scope of this chapter. The authors use a Markov chain with constant transition probabilities as the packet error statistics and evaluate the ARQ error control performance. Since the packet errors can be modeled only as a time-inhomogeneous Markov chain as described in Chapter 4, some of the conclusions could be invalid. The comparison of performance analysis results when using these two different Markov chain models is given in Section 5.4.

5.2 Performance Analysis of ARQ

5.2.1 Calculate the Values of HMM Model and Gap Model

HMM model

If we adopt the HMM model, the values can be calculated as in Section 4.3.1.

Gap model

In the gap model, $P(j|g)$ and $P(j|b)$, the conditional probability that there exist $j-1$ correct packets between two consecutive corrupted ones are given in Expression (4.10). In order to analyze the performance of the GBN protocol, we also define $P_2(j)$, the conditional probability that given one corrupted packet, there is a correct one j slots later. And there is not requirement on the $j-1$ packet statuses between these two packets. The probability, $P_2(j)$, is defined as:

$$P_2(j) = P(Z_{i+j} = B | Z_i = B) \ ,$$

where $j = 1, 2, 3, \ldots$.

From the analysis in Section 4.2, it is clear that the initial channel state of the corrupted packet, packet i, will determine the value of $P_2(j)$. So we have two sets of $P_2(j)$:

$$\begin{cases} P_2(j|g) &= P(Z_{i+j} = B | Z_i = B, X_{b_i} = g) \\ P_2(j|b) &= P(Z_{i+j} = B | Z_i = B, X_{b_i} = b) \end{cases} \tag{5.1}$$

where $j = 1, 2, 3, \ldots$.

From the formulas given in Section 4.2.1, it is easy to calculate the value of $P_2(j|g)$ and $P_2(j|b)$. After finding the probabilities in the gap model, we can further analyze the communication performance following the discussion in [40].

5.2.2 Performance of ARQ

HMM model

When adopting the HMM model, the stationary distribution can be calculated for a time-homogeneous Markov chain Z'. No matter what initial state the Markov chain starts with, the stationary distribution is the distribution of all the events when the chain is running for a long enough time. So, from the 4-by-4 transition matrix as described in Section 4.3.1, we can get the distribution of gA, gB, bA, and bB, i.e., P_{gA}, P_{gB}, P_{bA}, and P_{bB}. According to the definition of probability, $P_{gA} + P_{gB} + P_{bA} + P_{bB} = 1$. And $P_{gA} + P_{bA}$ is the probability that the packet transferred is correct no matter what the initial channel state of this packet is. This is then the packet error probability.

In the SR protocol, only the corrupted packets need to be retransmitted. So all the packets transferred correctly contribute to the connection throughput. Then the throughput efficiency, which is the connection throughput normalized to the bandwidth occupied by the connection, can be calculated as $P_{gA} + P_{bA}$.

In the GBN protocol, after the sender finds that one packet is corrupted, it will go back to send that packet again and all the packets after it. The total number of retransmission, n, is the packets that can be transmitted within the round trip time of the connection. So in the GBN protocol, a physically correctly transferred packet may or may not contribute to the connection throughput. If a correct packet is among the n packets sent after an erroneous one, it will be resent again and this correct transmission cannot contribute to the throughput. The conditional probability that given one corrupted packet, there is a correct one n slots later, as defined in Section 5.2.1, is important for calculating the connection throughput. Since this probability cannot be calculated easily in the HMM model, we will not use this

model for GBN protocol performance.

Gap model

In [40], throughput and delay analysis are given for both the gap model and the Markov chain model, where the Markov chain discussed has constant transition probabilities. In Chapter 4 we studied which model can describe packet error statistics in bursty channels. And the conclusion is that even though the packet error statistics can be modeled as a Markov chain, the transition matrix is decided by the initial channel state of the present packet and the values differ with different initial channel states. This Markov chain is not time-homogeneous and thus the average transition matrix may not be an appropriate solution. Instead the gap model is proposed as a proper model for the packet error statistics.

Packet error rate (PER) of the SR protocol can be calculated from $P(j|g)$ and $P(j|b)$:

$$PER = 1 - \frac{1}{P_X(g) \times \sum_{j=1}^{\infty} jP(j|g) + P_X(b) \times \sum_{j=1}^{\infty} jP(j|b)} \tag{5.2}$$

Remember that $P(j|g)$ and $P(j|b)$ are the probabilities that there exist $j-1$ corrupted packets between two consecutive correct packets, given that the initial channel state of the first correct one is *good* and *bad*, respectively. Then the expectation of $P(j)$, $P_X(g) \times \sum_{j=1}^{\infty} jP(j|g) + P_X(b) \times \sum_{j=1}^{\infty} jP(j|b)$, is the average length of such a group of packets that a correct packet is followed by corrupted ones. And each such packet group is followed by another group. So this expectation is the average number of packets among which one correct packet occurs. Thus, the reverse value of this expectation, i.e., the subtrahend in of Expression (5.2), is the probability that a packet is transmitted correctly. And Expression 5.2 is the packet error probability. Similarly, the packet error probability of the SR model can also be calculated from $Q(j|g)$ and

$Q(j|b)$, the probabilities that there exist $j-1$ correct packets between two consecutive corrupted packets given the initial channel conditions.

Markov Chain model

For a time-homogeneous Markov chain, the throughput and delay analysis are given in [40]. We will use the results directly.

In the following sections, we will analyze the packet length in order to achieve optimal communication performance under two metrics, i.e., goodput and energy consumption, according to the discussion on the gap model in [40]. We will then compare the performance analysis results when the gap model and the time-homogeneous Markov chain are adopted. We will also compare them through simulation.

5.3 Nonnumeric Analysis on Optimal Packet Length

5.3.1 Assumptions

We assume that the sender has enough information to send out, and packets with the same length are transmitted on the wireless channel periodically, which means that the time distance between any two adjacent packets is a fixed value. We also assume the packet header size is the same for every packet. When we use the term *packet header* in Chapter 5, we actually refer to both the MAC and PHY packet headers and trailers when fragmentation occurs in the MAC layer. When no fragments are needed, we use this term to refer to the MAC, PHY, data link, and transport layer headers. The time spent on transmitting each packet is denoted as one slot. In the GBN protocol, the round trip delay for determining if a packet is corrupted can then be measured in number of slots. This is also the number of packets that need to be retransmitted when a NACK is received. As discussed in Section 5.2, the packet length determines the packet error statistics, which in turn influence the ARQ throughput.

5.3.2 Decision Criteria

When discussing optimal packet length, we need to first make clear the decision criteria. There are many communication performance aspects. We can find the optimal packet length in order to achieve the best performance for each of them. These optimal lengths may not be the same. In this chapter, we consider both the goodput as in [1] and the energy consumption per useful bit as in [37]. When one packet arrives at the receiver correctly, the packet header is not useful to higher layers. Only the data payload contributes to the useful bits. Goodput is defined as the throughput of the useful bits seen by the receiver. This determines how fast the receiver can receive useful information correctly. Goodput is the most important performance aspect for some connections and the packet length to achieve the highest goodput needs to be found. Since goodput is actually the data payload part in the throughput, goodput can be calculated using the throughput and the packet header length.

On the other hand, in some mobile communication environments, energy constraints need to be considered. How to make the receiver get as many useful bits as possible using fixed power resources becomes more important than achieving the highest goodput. And the energy consumption per useful bit is the inverse of the number of useful bits given the power resource. In order to transmit/receive a packet, the transceiver has to be powered on during the transmission/reception period and must consume energy for every bit. The power needed for either sending or receiving each bit can be assumed as two fixed values, respectively. This is part of what we refer to as *energy consumption per bit*. This kind of energy consumption includes all that is proportional to the packet length. There is also power consumption needed for each packet, no matter how long the packet is. We refer to this as *energy consumption per*

packet. And there are many possible factors that can contribute to this power consumption. For example, in some MAC protocols, reservation on the wireless channel is required. The energy for channel reservation is thus necessary and is independent of the packet length. On the other hand, some protocols suggest some power saving techniques for the transceiver. One of them is that in order to save energy, the transceiver needs to be turned on just before sending a packet and turned off when the sending is finished. Turning on as well as turning off the transceiver requires some fixed power consumption no matter how long the packet is. So the energy consumption per useful bit is then decided by the throughput, the energy required per packet, the energy required per bit, and the packet header length.

For a correctly transmitted packet with larger size, we can save on the overhead on the packet header, both in throughput and in energy. On the other hand, when packet length increases, the packet error rate also increases. More erroneous packets will cause more retransmissions, where more bandwidth and more energy will be consumed. So analysis is needed to reach the optimal tradeoff.

5.4 Numerical Results and Discussion on Optimal Packet Length

5.4.1 Simulation Environment

We can calculate the optimal packet lengths for goodput and average energy per useful bit following the method described in Section 5.2. In order to verify the theoretical analysis, we also provide the simulation results. We simulate the connection by using Matlab as follows:

- Use the Markov chain model to generate the series of the channel states.
- Use the IID model to simulate whether every bit transmitted in each state is correct or not.

- The sender generates packets periodically.
- For any packet, whether it is correct or not is determined by all the bits it contains. The packet is correct only when all the bits are transmitted correctly.
- After a certain delay, the sender will receive the ACK/NACK for one packet. If the packet was not transmitted successfully, the sender will perform the retransmission according to a certain ARQ protocol.
- The receiver counts the correct packets. It will omit duplicate packets in the GBN protocol.

5.4.2 Goodput Versus Different Packet Lengths

When we analyze the communication performance, we assume a perfect feedback channel. The feasibility of this assumption lies on the fact that ACKs and NACKs are usually short so that the acknowledgments have much less probability to be corrupted.

First, we can find the formula for the throughput efficiency. As pointed out in Section 4.3.1, using the HMM model can achieve the correct results of packet error/error-free probability. And the results are expected to be the same as these from the gap model. Since the packet error-free probability is actually the throughput efficiency of the SR protocol, the HMM model can be used for the SR protocol. Obviously, the gap model can also be used for the SR protocol. In the GBN protocol, the distribution of packet error/error-free cluster lengths should be used to describe the throughput efficiency correctly. Since this distribution cannot be calculated easily in the HMM model, only the gap model can be used for the GBN protocol.

Second, we can find the formula for the goodput. Denote the throughput for one connection as α. Also define the length of a packet as L_{packet} and the header length

as L_{header}. Then goodput can be calculated as:

$$goodput = \alpha \times \frac{L_{packet} - L_{header}}{L_{packet}}$$

For both the theoretical analysis and the simulation, we use the parameters from the car speed model used in [38] for the bit level error model, i.e., $p_{gb} = 0.0000535, p_{bg} = 0.000496, BER_g = 0.00001, BER_b = 0.1$. For all the figures presented in Chapter 5, we use this set of parameters. We calculate and give the plot when the packets are transmitted continuously, i.e., after each packet the following packet will be sent out immediately.

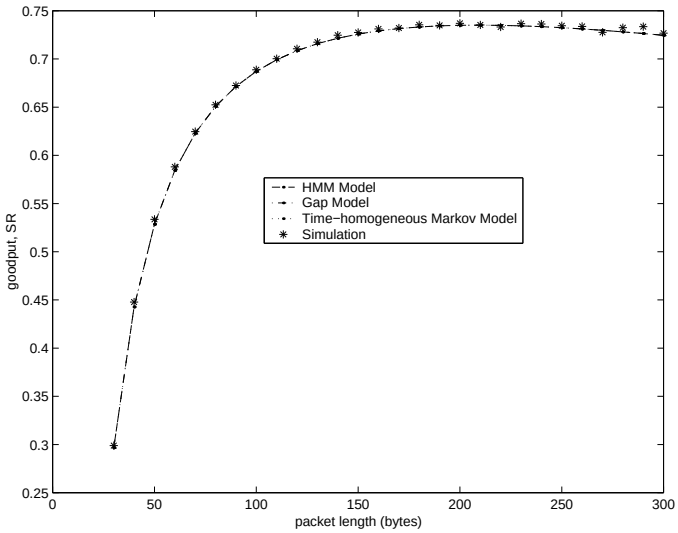


Figure 5.1: Goodput Efficiency Comparison Among Three Models, with 20 Bytes Header Length

In Figure 5.1, we plot the goodput efficiency achieved in the SR protocol when the HMM/gap model or the Markov chain model is used and header length is 20 bytes. As expected, the HMM model achieves the same results as the gap model. At the same time, the time-homogeneous Markov chain can describe the average packet error/error-free rate successfully and the connection throughput can be fully

determined by this value. So the curves achieved for both the Markov Chain and the HMM/gap model fully overlap. And the theoretical results from these three models match the simulation results.

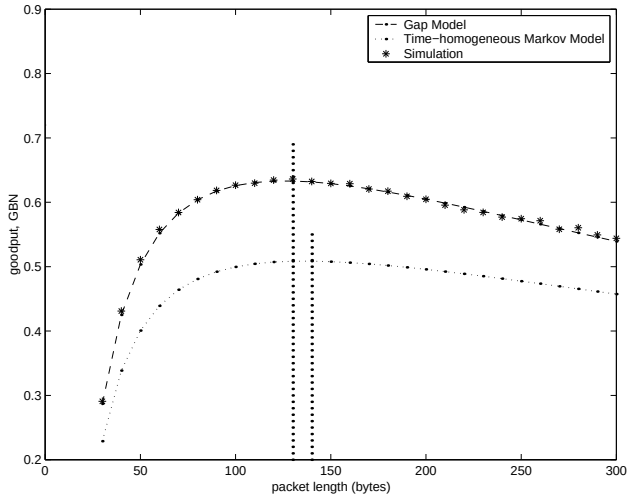


Figure 5.2: Goodput Comparison Between Two Models, with 20 Bytes Header Length, 3 Slots Delay

In Figure 5.2, we give the goodput efficiency in the GBN protocol when the gap model or the Markov chain model is used. Since the throughput cannot be derived easily from the HMM model, we do not consider this model in the GBN protocol evaluations. The delay is set to 3 slots. Again, the length of the header is set to 20 bytes. The vertical lines in Figure 5.2 indicate the optimal packet length - the best goodput for each model. The analysis results reached from the gap model match the simulation results closely. From Figure 5.1 and Figure 5.2, it is clear that the goodput efficiency of SR is better than the GBN protocol. It is also clear that the time-homogeneous Markov chain introduces incorrect analysis results while analyzing the GBN protocol even though it can give the correct goodput efficiency for the SR protocols. The reason is that the time-homogeneous Markov chain model

cannot describe the probability on the packets error/error-free length correctly (see Chapter 4) and this probability influences the GBN throughput directly. So using a Markov chain as the packet error model could give a wrong optimal packet length, from which the best goodput efficiency cannot be achieved. This model could also provide other wrong conclusions. For example, we need to compare the performance between ARQ and other error correction mechanisms, e.g., FEC, and the theoretical analysis is thus conducted. The comparison results may not be accurate and if we make a choice to get better performance based on these results, a wrong selection could be made.

5.4.3 Energy Consumed Versus Different Packet Lengths

As discussed in Section 5.3, a certain amount of energy is needed for each bit sent over the channel. At the same time, there is also a certain amount of energy needed for a whole packet, no matter how long the packet is. We define the energy needed per bit as E_{bit} and the energy per packet as E_{packet}. Still denote the throughput for one connection as α. The energy consumption per useful bit can be calculated as:

$$energy \quad consumption \quad per \quad useful \quad bit = \frac{E_{bit} \times L_{packet} + E_{packet}}{\alpha \times (L_{packet} - L_{header})}$$

We set $E_{bit} = 1$ and $E_{packet} = 30$ to make the result normalized to the power needed for the physical transmission of one bit. Since the Markov model can give correct results for the SR protocol, we give only the results for the GBN protocol in Figure 5.3, using the same parameter settings as in Section 5.4.2.

We also plot the vertical lines in Figure 5.3 to indicate the optimal packet length and the lowest energy consumption per useful bit under each model. Again, the results from the gap model match the simulation results and the Markov model gives inaccurate analysis results, both in the optimal packet length and the lowest energy

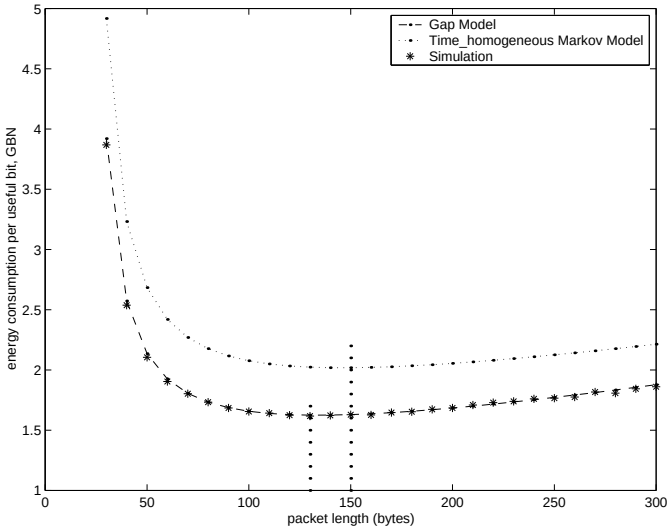


Figure 5.3: Energy Consumption Comparison Between Two Models, with 20 Bytes Header Length, 3 Slots Delay

consumption per useful bit.

5.5 Conclusion

So far, we have provided the HMM model and the gap model for the packet error statistics. And the optimal packet length is analyzed in order to get the highest good-put efficiency and the lowest energy consumption per useful bit. We also compare these two models with the time-homogeneous Markov chain. And the theoretical results are verified with simulation results. The conclusion is that the HMM model can be used to obtain the correct average packet error probability. Even though other aspects of connection performance can also be analyzed using the HMM model, we did not analyze them since a lot calculations may be involved. On the other hand, time-inhomogeneous Markov chains can model only the average packet error rate correctly. It cannot describe the distribution of packet error/error-free length correctly. So when this distribution influences the connection performance, e.g., the

GBN protocol, the Markov model could yield wrong analysis results. After achieving the performance analysis of the ARQ protocol in bursty channels, we can further evaluate the performance of FEC in bursty channels in some energy-constrained wireless environments, such as wireless sensor networks and mobile ad-hoc networks.

Chapter 6

FEC Performance in Bursty Channels

6.1 Related Work

6.1.1 FEC Error Control

Error occurrence is unavoidable, especially for the wireless communication channels. In order to recover from these errors, we can only transmit the same information redundantly to increase the probability that some of the information can reach the receiver without any error.

There are two primary mechanisms to achieve this purpose. The first one is ARQ. As described in Section 5.1.1, duplicate packets are sent out when necessary to increase the probability of delivering correct packets. ARQ will suffer long delay for connections with high round trip time. Moreover, if too many errors occur on the wireless channel, ARQ cannot perform well. FEC should be adopted in this situation. The fundamental idea of FEC is to transmit the original data together with some redundant data. Adopting certain error correction coding on the original data generates the redundant part. The receipt of only part of the whole data will permit the recovery of the original data. Thus the packet error rate after decoding is expected to be much less. In Chapter 6, we will refer to *packet error/error-free* as the packet status after the decoding of FEC unless otherwise stated. There are several kinds of error correction codes [39] [10] [57]. In this chapter, we will analyze the error correction codes that can recover all the packets with not more than k errors.

The drawback of FEC is the fixed overhead added to each packet. In order to achieve better performance, ARQ and FEC can be combined to try to achieve the positive aspects of both ARQ and FEC.

6.1.2 FEC Performance

Frossard [21] discusses the FEC performance of block correction codes, where k video packets are followed by $n-k$ FEC packets. If at least k out of the n packets are correctly received, the information carried by the video packets can be decoded. The FEC performance is analyzed when the packet error process can be modeled as a renewal error process. A Markov chain is then taken as an example for the packet error process and the numerical results are calculated. Treating one packet as one bit, the performance discussed in [21] can be used for wireless sensor networks. However, a renewal process is a process where the lengths of consecutive error/error-free clusters should follow an IID model. From our discussion in Section 4.1.2, it is clear that the bit errors in wireless channels cannot be modeled as a renewal process, so the analysis results in [21] cannot be used for the wireless sensor networks directly, where interleaving is not practical.

In [54], the authors examine the performance of convolutional codes. They measure the power consumption for coding/decoding and packet transmission. Then these values are used to analyze the FEC performance. However, the channel bit error model is assumed to be IID. When the bit errors can be modeled only as a Markov chain with a specific bit error rate in each state, the conclusion could be different.

6.2 Analysis of Packet Error Statistics after FEC Recovery

In this section, we will calculate the conditional packet error probability after FEC recovery. We assume that the sender will send out packets periodically. And every packet is of the same size, i.e., n bits. Denote the maximum number of errors that can be corrected by the FEC code as k. And assume the error correction code can recover all the corrupted packets with less than or equal to k errors.

6.2.1 Calculate the Conditional Packet Error Probability

In order to calculate the packet error probability for FEC, we need to use the stochastic processes defined in Section 4.2.1. Recall the first one, $X(i), i = 0, 1, 2, \ldots$, is the channel status sampled for each bit sent over the channel. The event set for X is $\{g, b\}$, where g stands for the *Good* channel status and b for the *Bad* one. The second process is the bit error/error-free status observed for each bit sent over the channel, $Y(i), i = 0, 1, 2, \ldots$. The bit status is either 0, which means that the bit is transferred correctly, or 1, bit not correct.

After defining these two processes, we can get define third one that we are most interested in while analyzing FEC performance, the packet status $\hat{Z}(j), j = 0, 1, 2, \ldots$. The possible events of $\hat{Z}$ are error-free or error, denoted by A and B, respectively. We can calculate the packet error probability given the channel state when the first bit of a packet is sent, i.e., the initial channel state of one packet. Remember that the channel state could change at the edge of every bit. After knowing the initial channel state of packet j, the channel states for all other $n - 1$ bits can be either *Good* or *Bad*. So the total number of channel state series for the packet is 2^{n-1}. Given a channel state series, whether an error will occur on a bit will then be decided by the bit error rate on each channel state. Suppose packet j is composed of bits $b_j, b_j + 1, \ldots, b_j + n - 1$. A packet is correct after FEC decoding only when the number of errors in the packet is not more than k, the capability of error correction code adopted. So the packet error probability can be calculated only when the initial channel state is given. In other words, we can calculate $P(\hat{Z}(j) = A|X(b_j) = g)$ and $P(\hat{Z}(j) = A|X(b_j) = b)$. The formula for $P(\hat{Z}(j) = A|X(b_j) = g)$ is as follows:

$$P(\hat{Z}(j) = A|X(b_j) = g) = \prod_{i=b_j}^{b_j+n-1} P(Y(i)|X(b_j) = g), \tag{6.1}$$

where $\sum_{i=b_j}^{b_j+n-1} P(Y(i)) \leq k$.

The procedure that we take to calculate this probability is described as follows. First, calculate $Sg(n,m), 0 \leq m \leq n$, the probability that while transferring the packet which is composed by n bits, there are m times that the channel state is *Good* given that the initial channel state is *Good*. Considering the final channel state of the packet, we also define two probabilities, $Sgg(n,m)$ and $Sgb(n,m)$. $Sgg(n,m)$ is the case that the final channel state is *Good* while $Sgb(n,m)$ is for the final channel state as *Bad*. Since $Sgg(n,m)$ and $Sgb(n,m)$ enumerate the final channel state for the packet, we have

$$Sg(n,m) = Sgg(n,m) + Sgb(n,m) \tag{6.2}$$

And for $Sgg(n,m)$ and $Sgb(n,m)$, we have

$$\begin{cases} Sgg(n,m) = Sgg(n-1,m-1)p_{gg} + Sgb(n-1,m-1)p_{bg} \\ Sgb(n,m) = Sgg(n-1,m)p_{gb} + Sgb(n-1,m)p_{bb} \end{cases}. \tag{6.3}$$

When $N = 1$, we have

$$\begin{cases} Sgg(1,1) = 1 \\ Sgg(1,0) = 0 \\ Sgb(1,1) = 0 \\ Sgb(1,0) = 1 \end{cases}$$

After getting $Sg(n,m)$ recursively, we can further calculate the value of $P(\hat{Z}(j) = A|X(b_j) = g)$. Denote $P(\hat{n} = i|X(b_j) = g)$ as the probability that the number of erroneous bits in the packet is i. This probability can be calculated as:

$$\begin{aligned} &P(\hat{n} = i|X(b_j) = g) \\ &= \sum_{j=0}^{n} Sg(n,j) \sum_{l=0}^{i} \binom{j}{l} BER_g^l (1-BER_g)^{j-l} \binom{n-j}{i-l} BER_b^{i-l} (1-BER_b)^{n-j-i+l} \end{aligned} \tag{6.4}$$

From $P(\hat{n} = i|X(b_j) = g)$, $P(\hat{Z}(j) = A|X(b_j) = g)$ can be calculated easily:

$$P(\hat{Z}(j) = A|X(b_j) = g) = \sum_{i=0}^{k} P(\hat{n} = i|X(b_j) = g) \tag{6.5}$$

$P(\hat{Z}(j) = A|X(b_j) = b)$ can also be calculated following the same procedure, which actually analyzes all the channel state series. For each of the series, the initial channel state for the following packet can also be analyzed. Having this information, the statistics for the following packet can be calculated using the same method. Thus, knowing the initial channel state of one packet and using this method, we can calculate the error/error-free probability for this packet as well as the probability for any combination of this packet and its following packets. After finding these values, further statistical analysis results can be calculated.

6.2.2 Gap Model and FEC Performance Analysis

Since the packet error/error-free probability can be calculated only when the initial channel state is given, and the channel state is changing following a Markov chain model just considering one packet cannot describe the underlining stochastic processes correctly. Focusing on one packet thus cannot be used to discuss FEC performance. A proper way to discuss packet error after FEC could be adopting the gap model [40]. Under any situation, two consecutive correct packets are separated by an arbitrary number of corrupted packets. The gap model uses the probability for the number of these corrupted packets to describe the block error statistics. The gap model actually describes the property of a stochastic process through analyzing the probability of different lengths of corrupted packet runs. We will use the gap model to calculate some properties of the packet error stochastic process after FEC, i.e., the packet error rate and the average packet cluster length.

In the gap model, $Q_1(j)$ is defined as the probability that there exist exactly $j-1$

corrupted packets between two consecutive correct ones. The expression for $Q_1(j)$ is given as:

$$\begin{aligned} Q_1(j) &= P_X(g) \times P(\hat{Z}_{i+1} = B, \ldots, \hat{Z}_{i+j-1} = B, \hat{Z}_{i+j} = A | (\hat{Z}_i = A, X(b_i) = g)) \\ &+ P_X(b) \times P(\hat{Z}_{i+1} = B, \ldots, \hat{Z}_{i+j-1} = B, \hat{Z}_{i+j} = A | (\hat{Z}_i = A, X(b_i) = b)), \end{aligned} \tag{6.6}$$

where $j = 1, 2, 3, \ldots$.

$P_X(g)$ and $P_X(b)$ is the stationary distribution that the channel is in the *Good* and *Bad* states, respectively. These values are thus the global probability that the channel is in the corresponding state.

In order to get the average packet error cluster length, we also define $Q_2(j)$, the probability that the packet error cluster length is j given that the cluster appears.

$$\begin{aligned} Q_2(j) &= P_X(g) \times P(\hat{Z}_{i+1} = B, \ldots, \hat{Z}_{i+j} = B, \hat{Z}_{i+j+1} = A | (\hat{Z}_i = A, \hat{Z}_{i+1} = B, X(b_i) = g)) \\ &+ P_X(b) \times P(\hat{Z}_{i+1} = B, \ldots, \hat{Z}_{i+j} = B, \hat{Z}_{i+j+1} = A | (\hat{Z}_i = A, \hat{Z}_{i+1} = B, X(b_i) = b)), \end{aligned} \tag{6.7}$$

where $j = 1, 2, 3, \ldots$.

After setting up the gap model, we can calculate the packet error rate and average packet error cluster length.

In Section 5.2.2, the packet error rate for the SR protocol has been achieved when the gap model is adopted. Even though we are discussing the packet error rate when FEC is used, we can use a similar formula since the gap model is used in this case again. Packet error rate (PER), can be calculated from $Q_1(j)$:

$$PER = 1 - \frac{1}{\sum_{j=1}^{\infty} j \times Q_1(j)} \tag{6.8}$$

The formula for average packet error cluster length is:

$$\sum_{j=1}^{\infty} j \times Q_2(j) \tag{6.9}$$

6.3 Numerical Results and Discussion

6.3.1 Parameters Used and Simulation Environment

We now give some numerical results for the FEC performance. For the bit level error model, we use the parameters of the car speed model used in [38], i.e., $p_{gb} = 0.0000535, p_{bg} = 0.000496$. In order to compare the FEC performance, we define two cases. In the first case, $BER_g = 0.00001, BER_b = 0.1$. We set the BER in the second case extremely severe, i.e., $BER_g = 0.001, BER_b = 0.5$. Moreover, assume the length for all the packets are of the same size. We will analyze the FEC performance for the packet size of 20 bytes as well as 50 bytes. Notice the packet length we select for the numerical results, 20 bytes and 50 bytes, could be proper for wireless sensor networks.

In some MAC protocols proposed for wireless sensor networks [47] [56], TDMA is adopted. When TDMA is used, the source and destination addresses can be removed from the packet header and the packet length will become shorter. In some routing protocols, clusters are formed within the wireless sensor network [26]. Thus, a sensor node, which is not serving as the clusterhead, needs to send only the information it senses to the destination, i.e., the clusterhead. In a wireless sensor network, most of the nodes are of this type and will operate in this way. Moreover, the sensor nodes are supposed to send packets that can describe the sensed data instead of sending the whole data itself, for example, an image [27]. So the packet sent by a sensor node may be short. Denote the time for transmitting a packet as one slot. The packets are assumed to be transmitted every other slot. In other words, the packets are transmitted periodically. After one packet is sent out, one slot time later, another packet will be transmitted. We can calculate the packet error probability and average packet error cluster length using the formulas described in Section 6.2. We obtain the

results while varying the capability of the error correction code, i.e., the maximum number of errors that can be recovered by the coding adopted.

In order to verify the theoretical analysis, we also provide simulation results. We simulate the connection by using Matlab as follows:

- Use the Markov chain model to generate series of the channel states.
- Use the IID model to simulate whether every bit transmitted in each state is correct or not.
- The sender generates packets periodically.
- For any packet, whether it is correct or not is determined by the number of erroneous bits it contains. The packet is correct only when the number is not larger than k.
- After collecting the packet error status series, calculating the average packet error rate as well as the average packet error cluster length.

6.3.2 Packet Error Rate

In Figure 6.1, we give the packet error rate for the wireless channel where BER for the *Good* state is 10^{-5} and the BER for the *Bad* state is 10^{-1}. The theoretical results match the simulation results closely. It is clear that the more powerful error correction code we use, the lower packet error rate we can achieve. Intuitively, the packet error rate is determined by the probability that a packet contains a certain number of errors. And this probability decreases with the number of errors. So a coding that can correct more errors will generate a lower packet error rate. In wireless sensor networks, sensors could be deployed very densely such that same event can be detected by several sensors. These sensors could send out the information through

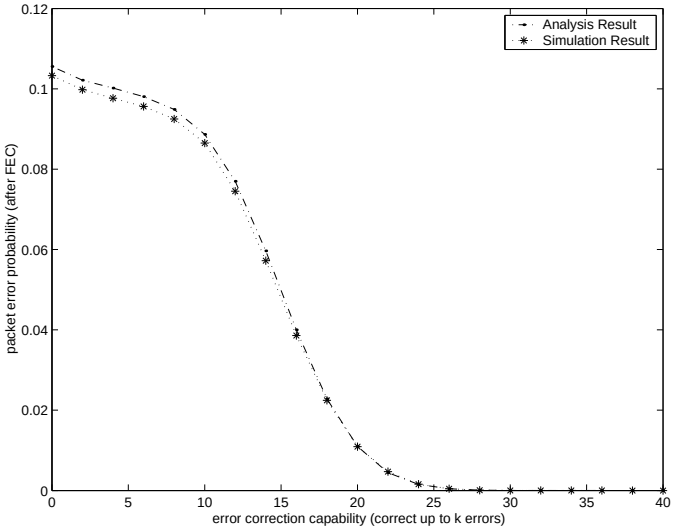


Figure 6.1: Packet Error Probability, 20 Byte Packets, case 1

different paths. So data reliability can also be achieved by the redundancy of the packets sent out by these sensors. When the paths these packets are routed through can be assumed as having independent channel statistics, the error correction code used in this case and how much sensor redundancy is allowed should be determined by the average packet error rate.

In many other cases, we are more interested in the goodput efficiency provided by adopting the error correction code. When considering FEC, goodput is defined as the throughput reaches the receiver, not including the additional bits added for error correction purpose. We assume that the error correction capability of k errors is provided by adding k extra bits to the original data. Notice that assuming k extra bits can correct up to k bits is the optimal case for error correction codes and the efficient goodput thus reached is optimal. But the results can still provide something to compare with.

The goodput efficiency normalized to the bandwidth occupied by the connection can be calculated as $(1 - PER) \times \frac{n-k}{n}$. And both the theoretical analysis and simula-

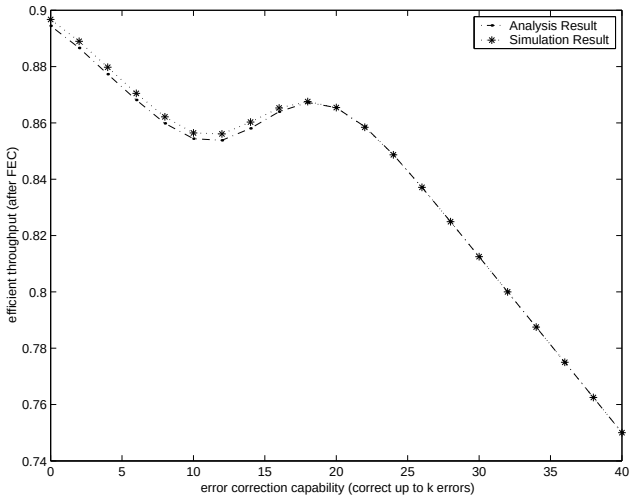


Figure 6.2: Efficient Goodput, 20 Byte Packets, case 1

tion results are plotted in Figure 6.2. The conclusion for the particular environment that we are evaluating is that using error correction codes can provide only lower goodput efficiency.

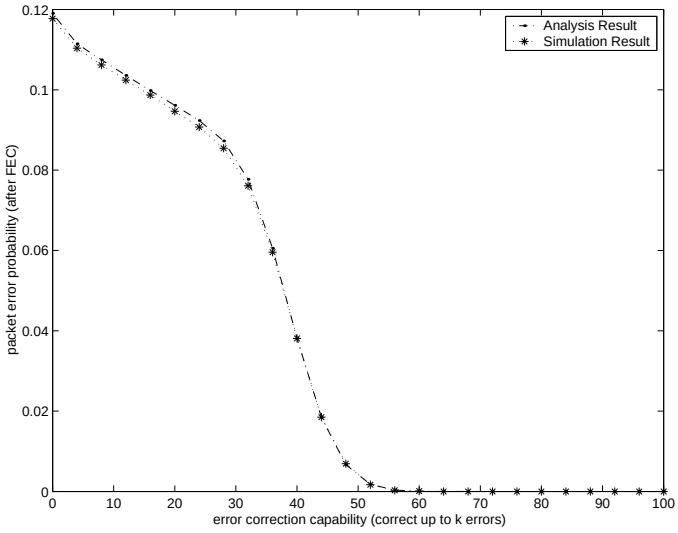


Figure 6.3: Packet Error Probability, 50 Byte Packets, case 1

We now give the packet error rate and efficient goodput results for a larger packet

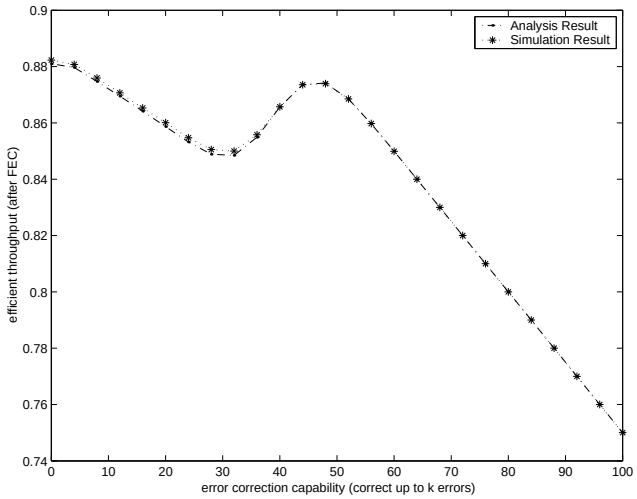


Figure 6.4: Efficient Goodput, 50 Byte Packets, case 1

length situation, i.e., 50 bytes. Again, as we expected, the more powerful error correction code we use, the lower packet error rate we can get. However, the goodput efficiency we can get by using an error correction code is still worse than that when not using any error correction code. Remember that we assume that k errors can be corrected by using only k extra bits, which is the optimal case and cannot be achieved in practice. When the actual efficiency rates of error correction codes are considered, even worse goodput efficiency will be realized.

From the aspect of goodput efficiency, error correction codes should not be used for either the 20 byte packet or the 50 byte packet. This conclusion demonstrates that without interleaving, error correction codes do not perform well on correlated errors. We should be conservative when using the error correction mechanism in sensor networks if we want to get higher goodput efficiency. We also analyze the connection performance in an even worse wireless channel. In the second case, we assume the BER for the *Good* state is 10^{-3} and the BER for the *Bad* state is 0.5. This case is much worse than what we have studied. We now give the theoretical

results as well as the simulation results to check whether error correction codes can be used in this case.

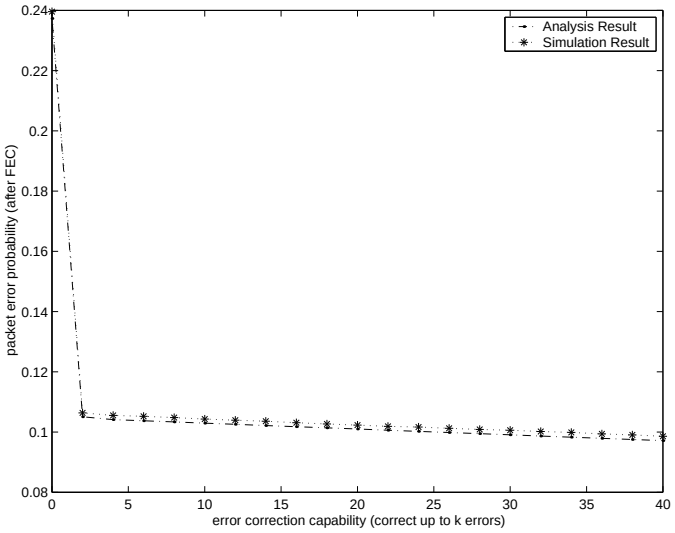


Figure 6.5: Packet Error Probability, 20 Byte Packets, case 2

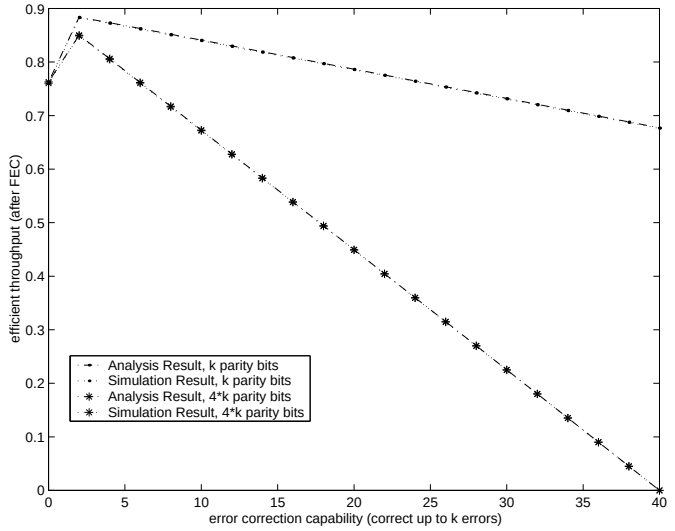


Figure 6.6: Efficient Goodput, 20 Byte Packets, case 2

From Figure 6.5 and Figure 6.7, again, we can find that the packet error rate can be lowered by adopting error correction codes. Moreover, there is a big difference

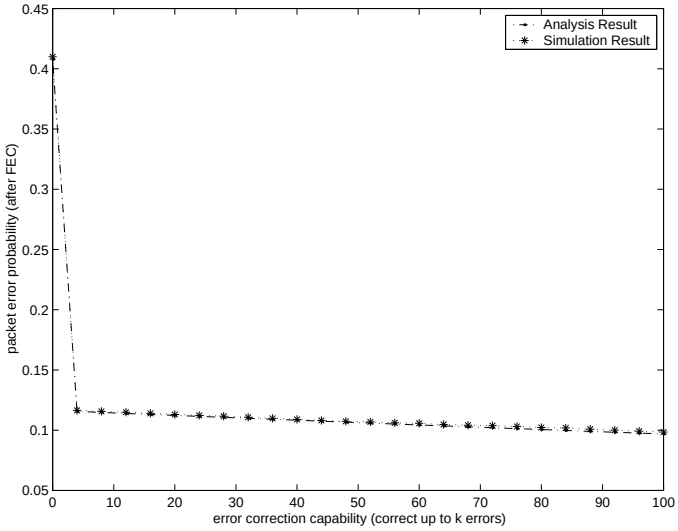


Figure 6.7: Packet Error Probability, 50 Byte Packets, case 2

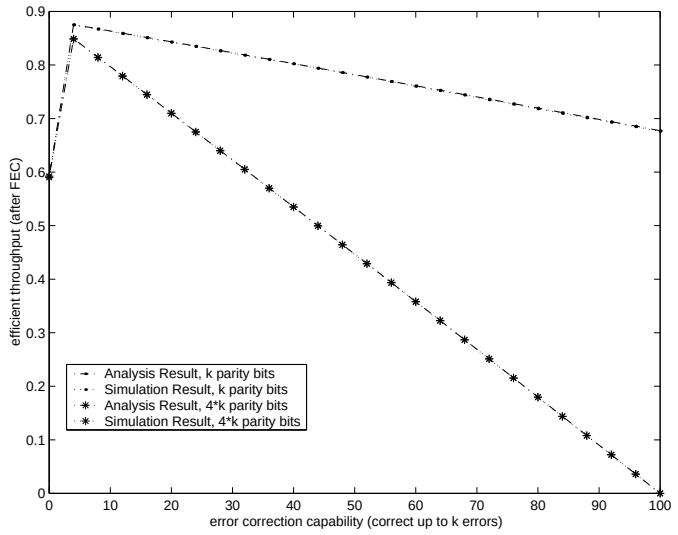


Figure 6.8: Efficient Goodput, 50 Byte Packets, case 2

in the packet error rate when a low-capability error correction code, i.e., correction up to $4-5$ bits, is used. When using the error correction code that can recover more errors, the packet error rate keeps decreasing, whereas the improvement is not significant. For the goodput efficiency, we give the results for two situations. In the

first one, we still assume k additional bits can correct up to k bit errors. In the second situation, we assume only $4 \times k$ additional bits can achieve such performance. The second case is more realistic. From Figure 6.6 and Figure 6.8, it is clear that the highest goodput efficiency can be reached with lower capability error correction codes. For the second situation, goodput efficiency decreases more quickly than the first situation we consider. The results demonstrate that error correction codes can be adopted in really bad wireless channels when no packet interleaving can be applied. But even in this case, only the codes with lower error-correction capability, which means fewer bits need to be added, can be used. Use more powerful error correction codes can only introduce lower inefficient bandwidth usage.

6.3.3 Average Packet Error Cluster Length

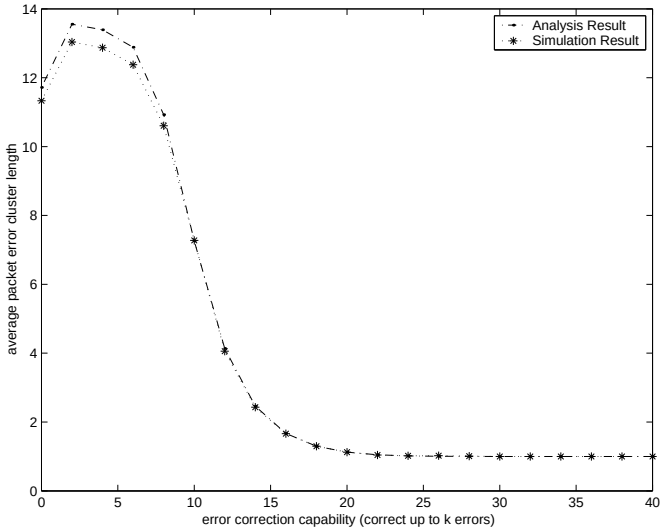


Figure 6.9: Average Packet Error Cluster Length, 20 Byte Packets, case 1

We also give the average packet error cluster length in Figure 6.9, Figure 6.10, Figure 6.11, and Figure 6.12. Again, the simulation results match the theoretical results closely.

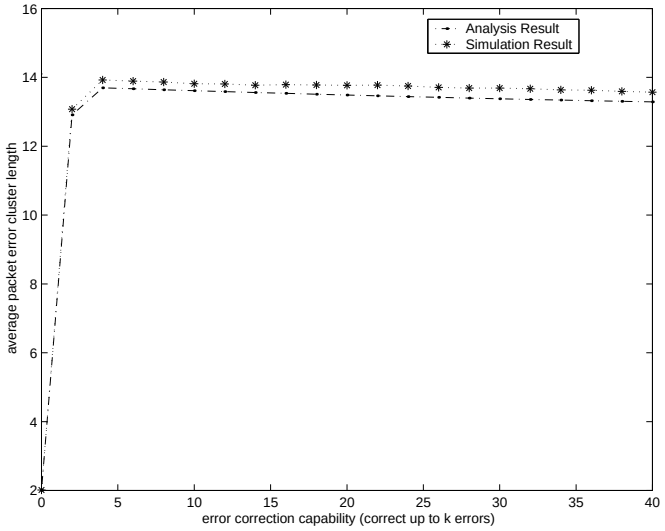


Figure 6.10: Average Packet Error Cluster Length, 20 Byte Packets, case 2

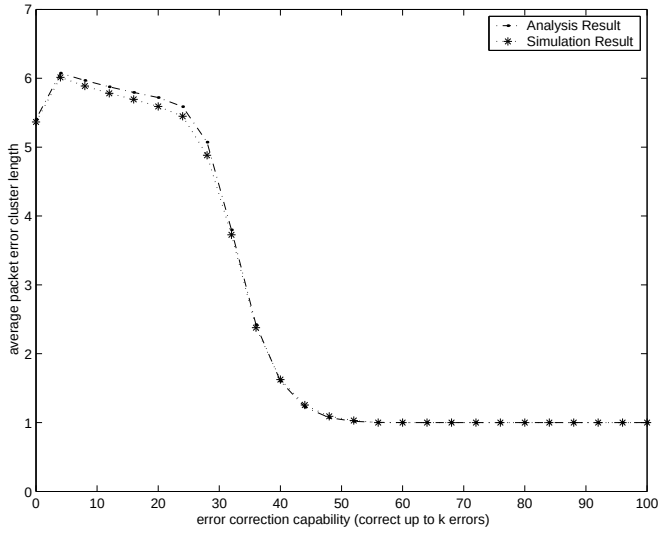


Figure 6.11: Average Packet Error Cluster Length, 50 Byte Packets, case 1

These results have some interesting features for the first wireless channel situation where the BER is lower for both channel states. In Figure 6.9, when the error correction capability is not high, i.e., $1 \leq k \leq 8$, average packet error cluster length is even larger than without FEC. The reason is that errors tend to occur in clusters

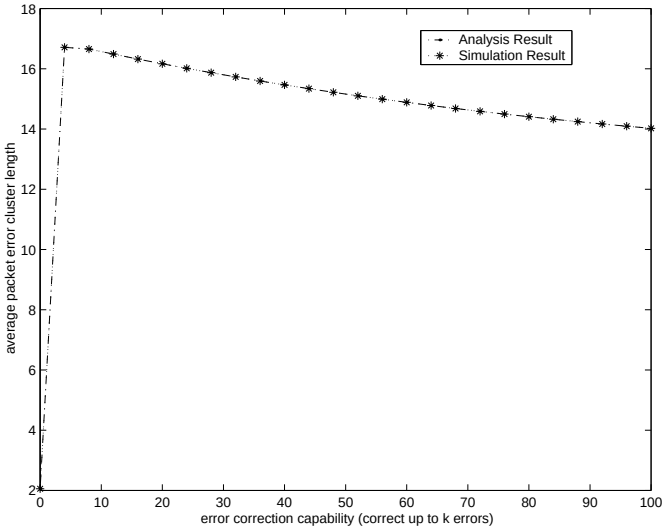


Figure 6.12: Average Packet Error Cluster Length, 50 Byte Packets, case 2

so that a large number of packet error clusters tend to have more than 8 bit errors in each packet. Although FEC decoding can shorten or even remove some other kinds of error clusters, it cannot improve these clusters. So the average packet error cluster length grows. With the increase in k, some packets with these kinds of error clusters will be corrected and the cluster will be divided into some smaller ones. So the average cluster length becomes smaller. When the code has a very high error correction capability, most erroneous packets can be corrected. Corrupted packets after FEC decoding occur so rarely that the error cluster length is almost 1. When we consider longer packet lengths, more errors are expected to occur in one packet. And a large number of packet error clusters tend to have more than 24 bit errors in each packet even though the error cluster length will become shorter. So the average packet error cluster length is even bigger than without FEC when $1 \leq k \leq 24$.

When considering the wireless channel with much higher BER in each channel state, the packet error clusters are different from the result with lower BER. The reason is that with higher BER, bit errors trend to occur in longer clusters. So the

corrupted packets tend to form long clusters with each packet having a lot of erroneous bits. The error correction codes evaluated in our plot can recover the corrupted packets with fewer errors, but not those with more errors. After error correction, the ratio of these long clusters to total packet error clusters becomes bigger. Thus as shown in Figure 6.10 and Figure 6.12, average packet error cluster length is larger than that without error correction code for packet lengths of 20 bytes and 50 bytes.

Since packet errors tend to occur consecutively, we might want to resend a group of packets when an erroneous packet occurs in the hybrid error correction mechanism. And the average packet error cluster length could be used to decide the number of packets resent.

6.4 Conclusion

So far, we have set up the gap model for packet error statistics and analyzed the FEC performance for wireless sensor networks, where interleaving is not adopted. We also give the numerical results both from theoretical analysis and simulation under a certain situation. The analysis results match the simulation results closely. These results show that the FEC error correction mechanism does not perform well in wireless sensor networks with the error characteristics we used for theoretical analysis and simulation.

Chapter 7

Conclusions and Future Work

In this dissertation, we evaluate the errors that can pass the internet checksum. We also analyze the packet errors and error control mechanisms in wireless channels. From the results we have achieved, we address and answer several essential questions related to data transmission performance. In this chapter, we will summarize the contributions of our work. We will also give some possible extensions of the work.

7.1 Work Summary

7.1.1 Performance Analysis of 1's Complement Checksum

For the internet checksum, which uses 1's complement arithmetic, people have never fully characterized the error detection possibility. Only the total number of error passing patterns [53] and the error detection performance for 2-bit and 3-bit errors [14] were found in previous work. We provide the expression to calculate the exact error passing probability and simplify the calculation to produce the exact values for normal TCP/UDP packet lengths.

Only after having achieved these values, we can compare 1's complement and 2's complement checksum. We also find that *traditional thinking on the internet checksum performance is wrong.* People usually think that the internet checksum fails to detect $\frac{1}{2^{16}-1}$ of most of the errors, so the error passing probability is also $\frac{1}{2^{16}-1}$. However, a much larger fraction of erroneous packets cannot be detected when the error bits are not high. And the probability that a packet contains fewer corrupted bits is large given that the error distribution is IID. So, for some real world conditions, the packet error passing probability is much bigger than $\frac{1}{2^{16}-1}$.

7.1.2 Header Compression

Header compression is necessary in wireless channels. There are several header compression algorithms that have been proposed. However, the recently observed likelihood of non-pathological packet reordering [6] and errors passing link-layer error detection [58] could deteriorate the performance of these algorithms. We discuss the influence of these phenomena on each existing algorithm. We also give an adaptive header compression algorithm and provide the performance analysis of this algorithm.

7.1.3 Packet Error Statistics and Error Control Performance

We also evaluate the packet error characteristics in wireless channels. Bit errors in wireless channels can be modeled using a two-state Markov chain with a specific BER in each state. The packet error statistics have also been modeled as a Markov chain with a constant transition matrix [64]. However, we prove that packet error statistics can be modeled only as a time-inhomogeneous Markov chain, i.e., a Markov chain without a constant transition matrix. Instead, a HMM model and a gap model are proposed as better models.

Using these models, we analyze the performance of ARQ protocols. We also evaluate the optimal packet length in wireless channels. All the theoretical results are verified by simulation results.

Using a similar idea as modeling the packet error statistics, we analyze the performance of FEC in wireless sensor networks, where packet interleaving may not be practical. Again, simulation results are generated and used to verify the analysis results.

7.2 Future Work

The research work we have completed so far could be extended in several directions. Our research results can also be used to solve some timely and persistent problems people have in networks. We give some examples of possible extensions to our research and the problems that can be solved.

7.2.1 Error Passing in Header Compression

A natural extension of the work on 1's complement checksum is to evaluate the errors that can be delivered from several header compression algorithms. This will help us understand these algorithms thoroughly. As analyzed in [58], errors that can pass link layer error detection might occur in certain spots, e.g., TCP or IP headers, due to hardware defects or software bugs. In this situation, our analysis allows just treating the TCP or IP header as a block and thus reaching the error passing probability.

7.2.2 Effect of Packet Reordering

Non-pathological packet reordering is hard or even impossible to be removed since this phenomenon is caused by parallelism inside a router or a logical path of a high-speed switch. One cannot expect this to be eliminated in the near future. If the packet reordering range is not large, TCP can tolerate the reordering and perform well. However, some existing header compression algorithms cannot. Our header compression algorithm can be used to perform header compression even with packet reordering. To the best of our knowledge, no model has been proposed for the packet reordering phenomenon. From our analysis on connection performance in wireless channels, the gap model can be used to evaluate flow control performance. So a proper model for packet reordering could be the gap model. One extension of our header compression work is to analyze header compression algorithms based on this

model.

7.2.3 Using the Packet Error Models Proposed

The models we proposed for packet error statistics in wireless networks can be used to analyze many other performance aspects and many other protocols. In theory, the models describe the process precisely and can be used in many cases. As long as the calculation complexity can be handled or the calculation can be simplified, we can adopt these models. There has been some previous work on modeling TCP flow control [36] [2]. From these works, we expect either the HMM model or the gap model could be used to analyze the TCP performance in wireless channels. Our work on connection performance could then be extended for TCP performance analysis. For wireless sensor networks, our work on ARQ performance and FEC performance analysis can be used to compare the performance of these two error control mechanisms. The work can also be extended to the hybrid error control mechanism (Hybrid ARQ Type I and II). From the performance analysis, we can then select the best scheme for a sensor network environment.

Appendix A:

Appendix: Proof of Lemma 4

First, for the coefficient of $f(n,x)$, we have the following lemma:

Lemma 5

Define $0 < |i| \leq 2^n - 2$,

- *the coefficient of* p^n *in* $f(n,i)$ *is* $\frac{1}{2^n}$, *if* i *is an odd value.*
- *the coefficient of* p^n *in* $f(n,i)$ *is* 0, *if* i *is an even value.*
- *the coefficient of* $p^{n-1}(1-p)$ *in* $f(n,i)$ *is less than or equal to* $\frac{2(n-1)}{2^n}$, *if* i *is an odd value.*
- *the coefficient of* $p^{n-1}(1-p)$ *in* $f(n,i)$ *is less than or equal to* $\frac{2}{2^n}$, *if* i *is an even value.*
- *the coefficient of* $p^x(1-p)^{n-x}$ *in* $f(n,i)$ *is less than or equal to* $\frac{1}{2}$, *for* $1 < x < n-1$.

Proof (by induction) From the expression of $f(n,1)$ and $(n, 2^n-1)$, the relationship holds for $i = 1$ and $i = 2^n - 1$ for any value of n. So, we need to consider only the values of i except for these two values.

We know that $f(1,1) = \frac{1}{2}p$, so the relationship holds for $n = 1$. Also, we have $f(2,1) = \frac{1}{2}p(1-p) + \frac{1}{4}p^2$, $f(2,2) = (1-p)f(1,1) = \frac{1}{2}p(1-p)$, and $f(2,3) = \frac{1}{4}p^2$. The relation then holds for $n = 2$.

Suppose that the relationship also holds for any value of n, where $n \geq 1$.

Consider the situation for $n+1$. For an odd value of i, where $i = 2 \times j + 1$, from Lemma 1, we have $f(n+1, 2j+1) = \frac{p}{2}f(n,j) + \frac{p}{2}f(n,j+1)$. The coefficient

of p^{n+1} is half the addition of that of p^n in $f(n,j)$ and $f(n,j+1)$. According to the assumption, one of the coefficients is $\frac{1}{2^n}$, the other is 0, so halving the addition results in $\frac{1}{2} \times \frac{1}{2^n} = \frac{1}{2^{n+1}}$. Similarly, the maximum value of the coefficient of $p^n(1-p)$ is $\frac{1}{2} \times (\frac{2(n-1)}{2^n} + \frac{2}{2^n}) = \frac{2n}{2^{n+1}} = \frac{2((n+1)-1)}{2^{n+1}}$. For other values of x, the coefficient is less than or equal to $\frac{1}{2} \times (\frac{1}{2} + \frac{1}{2}) = \frac{1}{2}$.

For an even value of i, where $i = 2 \times j$, from Lemma 1, we have $f(n+1, 2j) = (1-p)f(n,j)$. Since $f(n,j)$ contains terms up to only p^n, this expression will not generate a term p^{n+1}, so the coefficient of p^{n+1} is 0. The coefficient of $p^n(1-p)$ is less than or equal to $\frac{1}{2^n} = \frac{2}{2^{n+1}}$. For the term $p^{n-1}(1-p)^2$, the maximum value of the coefficient is either $\frac{2(n-1)}{2^n} \leq \frac{2^{(n+1)-1}}{2^{n+1}}$ or $\frac{2}{2^n} \leq \frac{2^{(n+1)-1}}{2^{n+1}}$. For other values of x, the maximum value of the coefficient is $\frac{1}{2}$.

So, the relationship also holds for $n+1$. Hence, the relation holds for any value of n. □

Proof of Lemma 4 Recall that y_{ab} is the probability that the 1's complement error value of one n-bit word is a if b bits get corrupted. y_{ab} is actually the coefficient of terms in $g(n,a)$. And, $\sum_{a=0}^{2^n-2} y_{ab} = \binom{n}{b}$.

By definition, $g(n,0) = (1-p)^n + \frac{2}{2^n}p^n$. For $0 < a < 2^n - 1$, $g(n,a) = f(n,a) + f(n,-(2^n-1)+a)$, which means the coefficient of $g(n,a)$ must be the summation of the two $f(n,x)$ values. One of the x values is odd and the other is even. We can analyze the coefficients according to Lemma 5.

When $b = 1$, the error value can be only $\pm 2^x$, where $0 \leq x < n$, and the coefficient is $\frac{2^{n-1}}{2^n}$. Since $2^n - 1 - 2^x$ cannot be a power of 2, the maximum coefficient of $g(n,1)$ is $\frac{2^{n-1}}{2^n}$ or 0. So, $\frac{y_{i1}}{\sum_{a=0}^{2^n-2} y_{a1}} \leq \frac{\frac{2^{n-1}}{2^n}}{\binom{n}{1}} = \frac{1}{2n}$.

When $b = n$, the coefficient of $g(n,0)$ is $\frac{2}{2^n}$;

the coefficient of $g(n,i)$, when $i \neq 0$, is $\frac{1}{2^n}$. And, $\frac{2}{2^n} = \frac{1}{2n}$ when $n = 4$, $\frac{2}{2^n} < \frac{1}{2n}$ when $n > 4$. So $\frac{y_{in}}{\sum_{a=0}^{2^n-2} y_{an}} \leq \frac{\frac{2}{2^n}}{\binom{n}{n}} = \frac{2}{2^n} \leq \frac{1}{2n}$.

When $b = n-1$, the coefficient of $g(n,0)$ is 0. The maximum value of (g,i), where $i \neq 0$, is $\frac{2(n-1)}{2^n} + \frac{2}{2^n} = \frac{2n}{2^n}$. So $\frac{y_{i(n-1)}}{\sum_{a=0}^{2^n-2} y_{a(n-1)}} \leq \frac{\frac{2n}{2^n}}{\binom{n}{n-1}} = \frac{2}{2^n} \leq \frac{1}{2n}$

For all other values of b, the coefficient of $g(n,b)$ is equal to or less than $\frac{1}{2} + \frac{1}{2} = 1$. And $\frac{y_{ib}}{\sum_{a=0}^{2^n-2} y_{ab}} \leq \frac{1}{\binom{n}{b}} \leq \frac{1}{\binom{n}{2}}$. When $n > 4$, $\binom{n}{2} \geq 2n$, so $\frac{y_{ib}}{\sum_{a=0}^{2^n-2} y_{ab}} \leq \frac{1}{2n}$. When $n = 4$, calculating $g(4,i)$ directly shows the maximum coefficient of term

$p^2(1-p)^2$ is $\frac{12}{16}$, so $\frac{y_{i2}}{\sum_{a=0}^{15} y_{a2}} \leq \frac{12/16}{\binom{4}{2}} = \frac{1}{8} = \frac{1}{2n}$. □

Bibliography

[1] Sandeep Agrawal and Suresh Singh. An Experimental Study of TCP's Energy Consumption over a Wireless Link. In 4^{th} *European Personal Mobile Communications Conference*, February 2001.

[2] E. Altman, J. Bolot, P. Nain, D. Elouadghiri, M. Erramdani, P. Brown, and D. Collange. Performance Modeling of TCP/IP in a Wide-Area Network. *INRIA-France, Research Report N=3142 (1997).*

[3] Ajay Bakre and B. R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *15th International Conference on Distributed Computing Systems*, pages 136–143, May 1995.

[4] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanisms for improvong TCP performance over wireless links. In *Proc. ACM SIGCOMM'96, Stanford*, Aug. 1996.

[5] J. S. Bendat and A. G. Piersol. Random Data: Analysis and Measurement Procedures. John Wiley & Sons, Inc. 1986.

[6] J. Bennett, C. Partridge, and N. Schectman. Packet reordering is not pathological network behavior. *IEEE Tran. Net. vol.7, no. 6*, 1999.

[7] Pravin Bhagwat, Partha Bhattacharya, Arvind Krishna, and Satish K. Tripathi. Using Channel State Dependent Packet Scheduling to Improve Throughput over Wireless LANs. In *IEEE INFOCOM*, March 1996, An extended version appears in ACM/Baltzer Journal on Wireless Networks, July 1996.

[8] Eugene Charniak. *Statistical Language Learning.* MIT Press, Cambridge, Massachusetts, 1993.

[9] C. Tzou Chen and G.S. Fang. A Closed-Form Expression for the Probability of Checksum Violation. *IEEE Trans. On Systems, Man, and Cybernetics*, Vol. SMC-10, No. 7, July 1980.

[10] G. C. Clark and J. B. Cain. Error-Correction Coding for Digital Communications. Plenum Press, 1981.

[11] M. Degermark, M. Engan, B. Nordgren, and S. Pink. Low-loss TCP/IP header compression for wireless networks. In *Second ACM/IEEE International Conference on Mobile Computing and Networking 1996 (MobiCom'96)*, 1996.

[12] M. Degermark, B. Nordgren, and S. Pink. IP header compression. RFC 2507, Internet Engineering Task Force, February 1999.

[13] S. Derring and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, Internet Engineering Task Force, December 1998.

[14] Y. Desaki, K. Iwasaki, Y. Miura, and D. Yokota. Double and triple error detecting capability of Internet checksum and estimation of probability of undetectable error. In *Pacific Rim International Symposium on Fault-Tolerant Systems*, pages 47 –52, 1997.

[15] W. Diepstraten. IEEE 802.11: Wireless Access Method and Physical Specification. Technical Report Doc:IEEE P802.11-93/70, May 1993.

[16] Jean-Pierre Ebert and Adam Wolisz. Power Saving in Wireless LANs: Analyzing the RF Transmission Power and MAC Retransmission Trade-Off. In *Proc. Euro-*

pean Wireless '99 and ITG Fachtagung Mobile Kommunikation, pages 187–192, Mnchen, Germany, October 1999.

[17] C. Bormann (ed.) *et al.* RObust Header Compression (ROHC). Internet Draft (work in progress) draft-ietf-rohc-rtp-09.txt, Internet Engineering Task Force, February 2001.

[18] E. O. Elliott. Estimates of Error Rates for Codes on Burst-error channels. *Bell Systems Tech. Journal*, 42:1977–1997, Sept. 1963.

[19] H. Liao *et al.* TCP-Aware RObust Header Compression (TAROC). Internet Draft (work in progress) draft-ietf-rohc-tcp-taroc-01.txt, Internet Engineering Task Force, March 2001.

[20] William Feller. An Introduction to Probability Theory and Its Applications. John Wiley & Sons, Inc. 1957.

[21] Pascal Frossard. FEC Performance in Multimedia Streaming. *IEEE Communications Letters*, 5(3):122–124, 2001.

[22] R. G. Gallager. Information Theory and Reliable Communication. New York, Wiely, 1968.

[23] Zoubin Ghahramani and Michael I. Jordan. Factorial Hidden Markov Models. In *Proc. Conf. Advances in Neural Information Processing Systems, NIPS*, volume 8, pages 472–478. MIT Press, 1995.

[24] E. N. Gilbert. Capacity of a Burst-noise Channel. *Bell Systems Tech. Journal*, 39:1253–1266, Sept. 1960.

[25] J. D. Hamilton. Time Series Analysis. Princeton University Press, 1994.

[26] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *IEEE Proceedings for the Hawaii International Conference on System Sciences*, pages 1–10, January 2000.

[27] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Fifth ACM/IEEE International Conference on Mobile Computing and Networking 1999 (MobiCom'99)*, pages 174–185, 1999.

[28] V. Jacobson. IP headers for low-speed serial links. RFC 1144, Internet Engineering Task Force, February 1990.

[29] C. Jiao and L. Schwiebert. Error Masking Probability of 1's Complement Checksums. In *the 10th International Conference on Computer Communications and Networks (ICCCN'01)*, October 2001.

[30] C. Jiao and L. Schwiebert. Analysis of Header Compression for Satellite Communications. In *the 5th Annual Michigan Space Grant Consortium Conference*, September 2000.

[31] C. Jiao and L. Schwiebert. Error Masking Probability of 1's Complement Checksums. submitted for publication, Information Processing Letters, June 2002.

[32] C. Jiao, L. Schwiebert, and G. Richard. Adaptive Header Compression for Wireless Networks. In *the 26th Annual IEEE Conference on Local Computer Networks (LCN'01)*, November 2001.

[33] C. Jiao, L. Schwiebert, and B. Xu. On Modeling the Packet Error Statistics

in Bursty Channels. In *the 27th Annual IEEE Conference on Local Computer Networks (LCN'02), to appear*, November 2002.

[34] S. Karlin and M. T. Taylor. A First Course in Stochastic Processes. Academic Press, New York, 1975.

[35] Almudena Konrad, Ben Y. Zhao, Anthony D. Joseph, and Reiner Ludwig. A Markov-Based Channel Model Algorithm for Wireless Networks. In *Proceedings of Fourth ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, July 2001.

[36] T. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, 1997.

[37] P. Lettieri and M. Srivastava. Adaptive Frame Length Control for Improving Wireless Link Throughput, Range, and Energy Efficiency. In *IEEE Infocom'98*, pages 307–314, 1998.

[38] Paul Lettieri, Christina Fragouli, and Mani B. Srivastava. Low Power Error Control for Wireless Links. In *Third ACM/IEEE International Conference on Mobile Computing and Networking 1997 (MobiCom'97)*, September 1997.

[39] S. Lin and D. J. Costello. Error Control Coding: Fundamentals and Application. Prentice-Hall, 1983.

[40] Deng-Lin Lu and Jin-Fu Chang. Performance of ARQ Protocols in Nonindependent Channel Errors. *IEEE Transactions on Communications*, COM-41(5):721–730, 1993.

[41] T. Mallory and A. Kullberg. Incremental Updating of the Internet Checksum. RFC 1141, Internet Engineering Task Force, January 1990.

[42] MATLAB. *http://www.mathworks.com.*

[43] Giao T. Nguyen, Randy H. Katz, Brian Noble, and Mahadev Satyanarayanan. A Trace-Based Approach for Modeling Wireless Channel Behavior. In *Proceedings of the Winter Simulation Conference*, pages 596–604, Coronado, CA, December 1996.

[44] C. Partridge, J. Hughes, and J. Stone. Performance of checksums and CRCs over real data. *SIGCOMM '95*, October vol. 25, no. 4, 1995.

[45] J. Postel. User Datagram Protocol. RFC 768, Internet Engineering Task Force, August 1980.

[46] J. Postel. Transmission Control Protocol. RFC 793, Internet Engineering Task Force, September 1981.

[47] J.G. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[48] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[49] T. S. Rappaport. Wireless Communications: Protocols and Practice. Prentice hall, 1995.

[50] A. Rijsinghani. Computation of the Internet Checksum via Incremental Update. RFC 1624, Internet Engineering Task Force, May 1994.

[51] S. Ross. Stochastic Processes. John Wiley & Sons, Inc., 1983.

[52] A. K. Salkintzis. A Survey of Mobile Data Networks. *IEEE Communications Surveys*, Vol 2, No.3, Third Quarter, 1999.

[53] Nirmal R. Saxena and Edward J. McCluskey. Analysis of Checksums, Extended-Precision Checksums, and Cyclic Redundancy Checks. *IEEE Trans. on Computers*, Vol. 39, No. 7, July 1990.

[54] E. Shih, B. Calhoun, S. Cho, and A. Chandrakasan. Energy-Efficient Link Layer for Wireless Microsensor Networks. In *Proceedings of IEEE Computer Society Workshop on VLSI*, pages 16–21, April 2001.

[55] C. K. Siew and D. J. Goodman. Packet data transmission over mobile radio channels. *IEEE Transactions on Vehicular Technology*, 38(2):95–101, 1989.

[56] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, 2000.

[57] J. D. Spragins, J. Hammond, and K. Pawlikowski. Telecommunications: Protocols and Design. Addison-Weslay, 1991.

[58] J. Stone and C. Partridge. When The CRC and TCP Checksum Disagree. *ACM SIGCOMM*, September 2000.

[59] F. Swarts and H.C. Ferreira. Markov Characterization of Digital Fading Mobile VHF Channels. *IEEE Transactions on Vehicular Technology*, 43(4):977–985, 1994.

[60] J. F. Wakerly. Error Detecting Codes, Self-checking Circuits, and Applications. New York, North-Holland, 1978.

[61] H. Wang and P. Chang. On verifying the first-order Markovian assumption for a Rayleigh fading channel model. *IEEE Trans. on Veh. Tech.*, 45(2):353–357, 1996.

[62] H. Wang and N. Moayeri. Finite State Markov Channel - a Useful Model for Radio Communication Channels. *IEEE Trans. on Veh. Tech.*, 44(1):163–171, 1995.

[63] M. Zorzi and R. Rao. ARQ Error Control for Delay-Constrained Communications on Short-Range Burst-Error Channels. In *47th IEEE Vehicular Technology Conference*, pages 1528–32, May 1997.

[64] M. Zorzi and R. Rao. On the Statistics of Block Errors in Bursty Channels. *IEEE Transactions on Communications*, COM-45(6):660–667, 1997.

[65] M. Zorzi, R. Rao, and L. Milstein. On the accuracy of a first-order Markov Model for data transmission on fading channels. In *IEEE ICUPC'95*, pages 211–215, November 1995.

Abstract

WIRELESS DATA TRANSMISSION:
ERROR CHARACTERIZATION AND PERFORMANCE ANALYSIS

by
CHANGLI JIAO
December 2002

Advisor: Dr. Loren Schwiebert

Major: Computer Engineering

Degree: Doctor of Philosophy

This dissertation addresses the accuracy of the data delivered from the transport layer and the connection performance in wireless channels. Our main purpose is to study these issues under network phenomenon/problems noticed recently and for wireless sensor networks which have become feasible only after the recent advances in IC fabrication technology, digital electronics, and wireless communications.

First, we analyze the error detection performance of 1's complement checksum. Internet checksum uses 1's complement arithmetic to detect errors in the content delivered by the data-link layer. Previous work on this topic determined the number of error passing patterns and the probability for only 2 and 3 bit errors, and the method used for determining the probability is hard to extend to more bit errors. We present a method to generate the formula of error passing probability. We then summarize some features of the probability. We also compare the performance of 1's complement checksum and 2's complement checksum.

Second, we address the problems with existing header compression algorithms and propose an adaptive header compression algorithm. Performing TCP/IP header

compression on wireless links is necessary due large IPv6 header as well as the limited resources of wireless channels. However, people recently noticed high frequency packet reordering, and packet errors that avoid link layer error detection. We analyze the influence of these problems on existing header compression algorithms. We also propose a new algorithm, which is adaptive to the wireless channel as well as the packet size.

Third, we study the connection performance of wireless channels. Wireless channels usually face bursty errors, i.e., errors are prone to occur in clusters. These bit errors can be modeled as a discrete time Markov chain. Packet error statistics have also been modeled as a DTMC in previous work. However, whether this Markov chain is time-homogeneous has never been addressed. We prove that the packet errors can be modeled only as a Markov chain without constant transition probabilities, which means the Markov chain is not time-homogeneous. Thus finding a constant transition matrix and then discussing the performance is not accurate. Instead, some other models are proposed and the packet error/error-free length distributions are thus analyzed. We then use these models to analyze the ARQ performance for wireless channels. We also set up a gap model for FEC used in wireless sensor networks, where packet interleaving may not be adopted. The FEC performance is then analyzed based on this model.

Autobiographical Statement

CHANGLI JIAO

Personal

Born on March, 14, 1972 in Kaifeng, People's Republic of China

Education

- M.S. in Electrical and Computer Engineering, August 1998
  Wayne State University, Detroit, Michigan
- M.S. in Computer Engineering, April 1996
  Beijing University of Posts and Telecommunications, Beijing, China
- B.S. in Computer Engineering, July 1993
  Beijing University of Posts and Telecommunications, Beijing, China

Publications

- C. Jiao, L. Schwiebert, and B. Xu, "On Modeling the Packet Error Statistics in Bursty Channels", 27^{th} *Annual IEEE Conference on Local Computer Networks (LCN)*, to appear.
- M. Kochhal, L. Schwiebert, S. K. S. Gupta, and C. Jiao, "An Efficient Core Migration Protocol for QoS in Mobile Ad Hoc Networks", 21^{st} *IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 387–391, April 2002.
- C. Jiao, L. Schwiebert, and G. Richard, "Adaptive Header Compression for Wireless Networks", 26^{th} *Annual IEEE Conference on Local Computer Networks (LCN)*, pages 377–378, November 2001.
- C. Jiao and L. Schwiebert, "Error Masking Probability of 1's Complement Checksums", 10^{th} *IEEE International Conference on Computer Communications and Networks*, pages 505–510, October 2001.
- Poster Presentations
  C. Jiao and L. Schwiebert, "Analysis of Header Compression for Satellite Communications", 5^{th} *Annual Michigan Space Grant Consortium Conference*, September 2000.

Research Interests

- Wireless Communication
- Mobile Computing
- High-Speed Interconnection Networks