Coding Performance on the AX.25 Radio Packet

by

Robert P. Wilson

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

Dr. T. Pratt, Chairman

Dr. F. G. Gray

Dr. A. Safaai-Jazi

May, 1994

Blacksburg, Virginia

CODING PERFORMANCE ON THE AX.25 RADIO PACKET

by

Robert P. Wilson

Committee Chairman: Dr. T. Pratt
Electrical Engineering

(ABSTRACT)

AX.25 packet radio is a popular means of data communications which has found many wide spread applications. For error control, AX.25 packet radio currently uses the go-back-N ARQ scheme which makes no attempt to correct errors, but only re-transmits packets which have been received with errors. This can lead to a very inefficient system when the channel error rate becomes substantially large. It has been found that by adding forward error correction (FEC) to the packet, the system performance can be substantially improved.

This thesis studies the performance of various BCH codes, the (23,12) Golay code, Reed-Solomon codes, and different rate convolutional codes with varying constraint lengths when used in conjunction with the go-back-N ARQ. Code combining and concatenation are also studied. The performance of these codes is based on throughput performance, code rate, and system complexity.

If is found that the (255,187) Reed-Solomon and the 1/2 rate v=9 convolutional codes can greatly enhance the performance of the AX.25 packet radio system. These codes provide good throughput performance over a large range of bit error rates and both are readily implemented. In conclusion, error control codes should be included with the AX.25 packet radio in order to improve its performance over noisy channels.

ACKNOWLEDGMENTS

I would like to thank Dr. T. Pratt for taking the time to help me along with this thesis. Without his guidance and patience, this would not have been possible. I also have to thank my committee, Dr. F. G. Gray and Dr. A. Safaai-Jazi, for reading my thesis and attending my defense.

CONTENTS

CHAPTER 1 INTRODUCTION

1.1 AX.25 Packet Radio
- 1.1.1 Automatic Repeat Request (ARQ)
- 1.1.2 Forward Error Correction (FEC)

1.2 Purpose
- 1.2.1 Project Criteria

1.3 Contents

CHAPTER 2 AX.25 PACKET RADIO

2.1 Structure
- 2.1.1 Frames
- 2.1.2 Fields

2.2 How AX.25 Works

2.3 AX.25 and Space Communications

CHAPTER 3 ERROR CONTROL CODES

3.1 Block Codes
- 3.1.1 Encoding
- 3.1.2 Decoding
- 3.1.3 Classes of Block Codes
  - 3.1.3.1 The Golay Code
  - 3.1.3.2 BCH Codes
  - 3.1.3.3 Reed-Solomon Codes

3.2 Convolutional Codes
- 3.2.1 Encoding
- 3.2.2 Decoding

3.3 Error Control in AX.25 Packet Radio
- 3.3.1 ARQ
  - 3.3.1.1 Mixed Modes of Re-transmission
  - 3.3.1.2 Hybrid ARQ
  - 3.3.1.3 AX.25 Packet Radio

CHAPTER 4 INTERLEAVING

4.1 Transmission Line Noise

4.2 Burst Error Correction

4.3 Cyclic Codes

CHAPTER 5 CODING THE AX.25 RADIO PACKET

5.1 Hybrid ARQ Performance

5.1.1 Probability of Packet Error
5.1.2 Throughput Efficiency
5.2 Code Combining
5.2.1 AX.25 Frame Hierarchy
5.2.2 Probability of Packet Error
5.2.3 Throughput Efficiency
5.3 Concatenated Codes
5.3.1 Probability of Packet Error
5.3.2 Throughput Efficiency
5.4 The Golay Code
5.4.1 Performance

CHAPTER 6 DATA ANALYSIS

6.1 Throughput Efficiency
6.1.1 Golay Code
6.1.2 Throughput Performance Selection
6.2 System Complexity
6.3 Code Selection

CHAPTER 7 IMPROVING SYSTEM PERFORMANCE

7.1 Selective Repeat ARQ
7.1.1 Throughput Efficiency
7.2 ARQ with Mixed Modes of Re-transmission
7.2.1 Throughput Efficiency

CHAPTER 8 CONCLUSION

8.1 Summary
8.2 Further Work

Appendix A: Golay Code

Appendix B: List of Symbols

Appendix C: References

CHAPTER 1

INTRODUCTION

It is common practice to incorporate error control to detect or correct errors which have occurred during transmission when designing a communication system. Many methods have been developed to carry out these functions, but their performance is as widely varied as the number of different techniques available. The purpose of this thesis is to study AX.25 packet radio and to recommend coding methods which improve performance under noisy channel conditions.

1.1 AX.25 Packet Radio

Amateur X.25, or simply AX.25, Packet Radio has been developed as a means of providing error free communications for the purpose of transmitting information packets. It has found many wide spread applications and in the recent past has even begun expanding into outer space with the launch of Amateur-Radio satellites such as AMSAT OSCAR 10.[1]

1.1.1 Automatic-Repeat-Request (ARQ)

AX.25 Packet Radio uses a method referred to as automatic-repeat-request (ARQ) for error control. ARQ is widely used for error control in data

[1] Stan Horzepa, p 12-1

systems because it is simple and provides high system reliability This type of scheme does not attempt to correct errors, but only detects them. When errors are detected in a transmitted packet, it is re-transmitted until it is received error free. This ensures that all data can be received virtually free of errors by the user.

There are three basic types of ARQ systems: stop-and-wait ARQ, go-back-N ARQ and selective-repeat ARQ. AX.25 packet radio employs the go-back-N ARQ. This method works well for systems with low round-trip delays and small data transmission rates, but otherwise the throughput performance becomes insufficient.

The throughput efficiency, or throughput, is defined as the ratio of the average number of information digits successfully accepted by the receiver per unit of time to the total number of digits that could be transmitted per unit of time.[2] One problem with ARQ is that as the channel bit error rate increases, many packets may have to be re-transmitted several times before they are correctly received and acknowledged which causes a reduction in system throughput.

1.1.2 Forward-Error-Correction (FEC)

Forward-Error-Correction (FEC) provides an alternative to using ARQ for error control. In an FEC system, redundant bits are added to the data at the transmission end so the receiver can correct as well as detect errors in the

[2] Lin and Costello, p 462.

packet. The throughput for a system which uses FEC is constant and is equal to the rate of the code used by the system.

1.2 Purpose

The purpose of this thesis is to attempt to improve the error control of AX.25 packet radio. To achieve this, a combination of FEC and ARQ, referred to as hybrid ARQ, will be used. The FEC will be applied to the radio packet so that when the receiver detects the presence of errors in a received packet, it will attempt to correct them. If correction is achieved, the packet will not have to be re-transmitted which should cause the system's throughput performance to be enhanced.

There are two main types of FEC which will be tested: block codes and convolutional codes. Some of the block codes to be used will be the (23,12) Golay Code and various length BCH and Reed-Solomon codes. Convolutional codes will be used which have different rates as well as different constraint lengths.

1.2.1 Project Criteria

When considering which type of FEC to use with the ARQ, there will be several considerations or criteria. The first of these is the throughput. This is an important measure of performance because it is desired to find a system which will maintain a constant throughput as the bit error rate increases. The magnitude of the throughput is also a factor, because this is a measure of how much information is being passed through the system compared to the number

of redundancy bits. It is desirable to maintain as large a magnitude as possible, which means that more information than redundancy bits are being passed. The maximum throughput is equal to the rate of the code used, so the code rate will be another consideration. One last factor will be the complexity and ease of implementation for the system. A system which is too complex or hard to implement will not be desirable.

1.3 Contents

This thesis begins with a summary of AX.25 packet radio specifying its format, how it works, and some of its current uses. Chapter 3 looks at some different block and convolutional error control codes detailing their format and performance. This chapter also reviews the different types of Automatic-Repeat-Request (ARQ) error control schemes. Chapter 4 describes interleaving which is a technique used by systems to correct errors which occur in bursts. Chapter 5 looks at the performance of some different codes when used in combination with ARQ. Specifically, the probability of packet error and throughput are plotted vs. varying bit error rates. Chapter 6 analyzes the results found in Chapter 5 and compares them to determine which error control coding performs the best with packet radio. The comparison is based on criteria previously specified. Chapter 7 takes some of the results from Chapter 5 and looks at the throughput performance when different ARQ methods are used. Chapter 8 includes a summary of the results and looks at future areas of research.

CHAPTER 2

AX.25 Packet Radio

The AX.25 Amateur Packet Radio Link-Layer Protocol referred to as AX.25 is a means of error free communications that in recent years has found its way into many wide spread applications. Besides its use by radio hackers, AX.25 packet radio has found applications in amateur television, traffic handling, public-service applications such as communications in disasters, emergencies, or public events, and in the early 1980's has even found its way into space communications. The AX.25 protocol is used to specify the content and format of an amateur packet radio frame. It also details how the frame is to be handled at the link layer by the packet radio station.

2.1 Structure

AX.25 packets are made up of small blocks of information called frames of which there are three different types. These are the *Information* frame, the *Supervisory* frame, and the *Unnumbered* frame referred to as the I, S, and U frames respectively. Each of these frames is subdivided into smaller blocks of information called *fields*. The fields can all be measured in octets, where an octet is a byte or eight bits. The various types of fields are the *flag* field, *address* field, *control* field, *frame-check sequence* field, and a *final flag* field. In addition to these, the I and UI frames also include a *protocol identifier* field and an *information* field. Figure 2.1 shows the basic structure of the information, supervisory, and unnumbered A.25 frames.

FLAG	ADDRESS	CONTROL	PID	INFORMATION	FCS
8 bits	112-560 bits	8 bits	8 bits	N × 8 bits	16 bits
01111110	call signs & SSIDS of destination, source, & optionally, digipeaters	Frame Type	Layer 3 Protocol Type	User Data	Calculated Value

Information Frame

FLAG	ADDRESS	CONTROL	FCS
8 bits	112-560 bits	8 bits	16 bits
01111110	call signs & SSIDS of destination, source, & optionally digipeaters	Frame Type	Calculated Value

Unnumbered and Supervisory Frames

Figure 2.1 AX.25 unnumbered, supervisory and information frame formats[1]

2.1.1 Frames

Information Frame

The *Information (I)* frame is used to transfer data from one station to another. This frame contains an information field.

Supervisory Frames

The *Supervisory (S)* frame is used to control the communication link. It can be used to acknowledge I-frames, request re-transmission or call for a suspension of transmission. There are three different types of the S-frame. These include the *Receive Not Ready (RNR)* frame, *Receive Ready (RR)* frame, and the *Reject (REJ)* frame. The Receive Not Ready frame indicates that the destination station is not able to accept any more information frames because it is presently "busy" or there is a buffer shortage. The Receive Ready supervisory frame indicates that the destination station is ready to receive information frames. It is also used to clear an RNR condition. The RR and RNR frames may also be used to acknowledge that an information frame has been

[1] Stan Horzepa, 3-6

properly received. The Reject supervisory frame is used by the destination station to request a re-transmission when an out-of-sequence frame is received.

Unnumbered Frames

There are six different types of *Unnumbered (U)* frames. Of these six, five are used to perform supervisory functions while the sixth is used to make unconnected transmissions. The *Set Asynchronous Balanced Mode (SABM)* unnumbered frame indicates a connection between two stations. To terminate a connection between two stations a *Disconnect (DISC)* frame is used. The *Unnumbered Acknowledge (UA)* frame is used to acknowledge the receipt and acceptance of either the SABM or DISC frames. When a station is busy and unable to make a connection with another station, it can reject the SABM frame by transmitting a *Disconnected Mode (DM)* frame.

A *Frame Reject (FRMR)* is used by either end to indicate an error condition which cannot be recovered by the re-transmission of identical frames. The last unnumbered frame is the *Unnumbered Information (UI)* frame. This frame allows the transmission of data from one station to another without a connection to the destination station.

2.1.2 Fields

Flag Field

The first field to appear in the frame is the flag field. This field is used to indicate the beginning and end of a frame. It is possible for two frames to share a flag field, so in this case the flag represents the end of one frame while also indicating the beginning of another. The flag is one octet long and always has the contents [01111110] so no other octet will be interpreted as a flag field. To insure that the other octets between the flag fields do not contain the same unique contents as the flag field a process called *zero bit insertion* or *bit stuffing*

is used. In this process whenever five consecutive one bits are encountered in the other frame octets, the transmitting packet-radio station inserts a zero bit after the fifth one bit. On the receiving end, whenever five one bits are received, the following zero bit is discarded.

Address Field

The second field to appear in the packet is the address field which is composed of the source and destination of the frame. It may also contain the call signs of up to eight other digipeaters which are devices that receive, store, and then re-transmit packet-radio transmissions. The address field can be from 14 to 70 octets long depending upon the number of digipeaters included in the field.

Control Field

The next field encountered is the control field. This field indicates the frame type and is only one octet long.

Protocol Identifier Field

The protocol identifier (PID) field will be present in I or UI frames and is used to indicate the type of network layer protocol that is in use. Like the control field, the PID is one octet long.

Information Field

The information field contains the user data that is transferred in an I, UI, or frame reject frame. This field has a maximum length of 256 octets prior to bit stuffing.

Frame Check Sequence Field

The frame check sequence field or FCS field is used for frame error checking. It consists of a 16-bit number that is calculated from the transmitted data by the transmitting station. When the frame is received, the receiving station recalculates the FCS from the received data. If the contents of the FCS

are equal to the FCS calculated by the receiving station, then no error has occurred in the frame. If the two are not equal then an error has occurred and the station will discard the frame.

The 16-bit sequence is defined to be the ones complement of the sum (modulo 2) of: The remainder of $x^k(x^{15} + x^{14} + x^{13} + \ldots + x^2 + x + 1)$ divided (modulo 2) by $x^{16} + x^{12} + x^5 + 1$, where k is the number of bits in the frame whose FCS is being generated, and the remainder after multiplication by x^{16} and division by $x^{16} + x^{12} + x^5 + 1$ of the contents of the frame between the last bit of the initial flag and the first bit of the FCS, but including neither.[2]

2.2 How AX.25 Works

This section illustrates the basic operating procedure for transmitting AX.25 radio packets. To demonstrate this, two fictitious stations will be used: VA1ROB will be the transmitting station identifier while VATECH will be the receiving station. The first step is for the VA1ROB terminal node controller to construct a frame with "VATECH0" in the destination address field and "VA1ROB0" in the source address field. The zero attached to the end of the identifier means that no secondary station identifier (SSID) is specified. The SSID allows one call sign to be used by more than one station and can be a number from 0 to 15.

After the source and destination stations have been specified, the control field is set as an SABM frame. The SABM frame is transmitted to VATECH-0 and an acknowledgment timer (T1) is started at VA1ROB. The value for T1 is at least twice the amount of time involved to transmit the longest possible frame plus the amount of time it takes for the destination station to transmit a proper

[2] R.J. Deasington, p 33.

response frame. The value for T1 will be increased depending upon how many, if any, digipeaters are included in the transmission route.

If VATECH-0 is monitoring the channel and is able to accept a connection with VA1ROB-0, then the terminal node controller at VATECH-0 will respond to the SABM frame with a UA frame. If VATECH-0 is unable to make a connection or is presently busy, the controller with respond with a DM frame.

In response to the frame returned from VATECH-0, the T1 timer at VA1ROB-0 will be canceled. If the returned frame was a UA frame, then VA1ROB-0 will begin data transmission. If the T1 timer elapsed before a response, VA1ROB-0 will continue transmitting SABM frames until a response is received or until a maximum number of retries is reached by the controller at VA1ROB-0.

When transmitting data, VA1ROB-0 is sending I frames. At the transmission of a new frame, the T1 timer is re-started. A maximum of seven I frames can be outstanding or unacknowledged at one time.

If the I frame is received in the proper sequence and without error, the destination station will send an acknowledgment of reception. The receiving station then has some different options. If VATECH-0 has I frames to send, it can use the I frame to indicate the acknowledgment, or it can send an RR frame followed by the I frame. If VATECH-0 does not have any I frames it will send an RR frame or possibly a RNR frame if it doesn't want to receive any more I frames at that moment.

When an I frame is received out of sequence, the frame is discarded and a REJ frame is returned by VATECH-0. A frame received out of sequence means that it arrived at the destination station in a different order than it was transmitted by the source station. The order can be determined by a difference between the received I frame's sequence number and the destination station's

received state variable. If the sending station receives a REJ frame, it will re-transmit the frame which was rejected.

An I frame will also be discarded at the destination of the FCS if the frame does not match the calculated FCS. If this happens the destination station will not make any acknowledgment so the T1 timer at the source station will run out which will cause a re-transmission of the faulty frame.

Once the data transfer between stations has occurred, either one can initiate a disconnection. The station's controller will simply send a DISC frame to the other and start its T1 timer. The receiving station will respond with a UA frame and enter the disconnected mode. When the station that initiated the DISC receives the UA frame it will cancel its T1 timer and then it will be in the disconnected mode. If the T1 timer runs out before a response, the DISC frame will be re-transmitted until a response is received, or until the maximum number of retries has been reached.

2.3 AX.25 and Space Communications

There are two different categories for packet radio communications in space: real-time communications and store-and-forward communications. In real-time satellite communications, the satellite is used to relay packets immediately from one station to another. Used in this capacity the satellite can be thought of as a digipeater. In order to perform real-time communications, the stations that wish to exchange packets must be able to see the satellite at the same time.

In store-and-forward communications, the satellite is used to store messages from a transmitting station and relay them later when it passes over a receiving station. When this type of communication is used, the satellite can be in a low earth orbit or as in the previous case a higher orbit could be used.

Before packet radio could be implemented into space, experiments had to be performed to see if it had any practical uses. Packet-radio was tested and proven on March 11, 1984 when AMSAT OSCAR 10 was used as a repeater to connect packet-radio stations on the East and West coasts of the United States.[3] Store-and-forward packet radio was first accomplished by radio amateurs on January 16, 1985 by a low earth orbit satellite, UoSAT0-OSCAR 11, and now this type of system is fully active in such countries as Australia, England, New Zealand, South Africa, the Soviet Union and the United States.[4] In the early 1990's, many such store-and-forward packet radio satellites have been scheduled to be launched for uses world wide.

[3] Stan Horzepa p. 12-1
[4] Stan Horzepa p. 12-2

CHAPTER 3

ERROR CONTROL CODES

Error-correcting codes were introduced in the 1940's in an attempt to implement a theorem of Shannon's which guarantees that virtually error-free communication can be obtained over a noisy channel.[1] In an ideal system the binary symbol output by the decoder should match the symbol that entered the encoder. In realistic systems there are occasional errors and hence the need for some type of detection and possible correction of these errors.

Many error-control codes or schemes have been developed to combat errors that occur in the communication process. The two common features of all error-correction codes are structured redundancy and noise averaging.[2] Structured redundancy is a method of inserting extra or redundant symbols into the information message. Noise averaging is accomplished by making the redundant symbols depend on a span of several information symbols. It is these features which provide a means for detecting and correcting errors.

There are two main objectives in using error correction codes. The first of these is to increase the efficiency of the communication system and the second is to reduce errors in transmission. For example, if a system is capable of correcting errors, it will not have to ask for the information to be re-transmitted upon every occurrence of an error, if that error is correctable. This in return will increase the throughput efficiency for that system.

For binary error-control coding there are two different types of codes that are commonly used today, block codes and convolutional codes.

---

[1] Assmus & Key, p. 25
[2] Tri T. Ha, p. 457

3.1 Block Codes

A *block code* is one in which a particular message from a source is always coded into the same fixed sequence of code symbols. As the name implies, block codes process information in blocks. The encoder divides the information sequence into message blocks of *k* information bits each. A message block is represented by the binary *k*-tuple $\mathbf{u}=(u_1,u_2,...,u_k)$ called a *message*. There are a total of $M=2^k$ different messages possible. Each message, **u**, is transformed independently by the encoder into an *n*-tuple ($n \geq k$) $\mathbf{v}=(v_1,v_2,...,v_n)$ called a *code word* or *code block* where n is referred to as the *block length* of the code. Figure 3.1 shows a simplified diagram of a coded system and the locations of **u** and **v**.

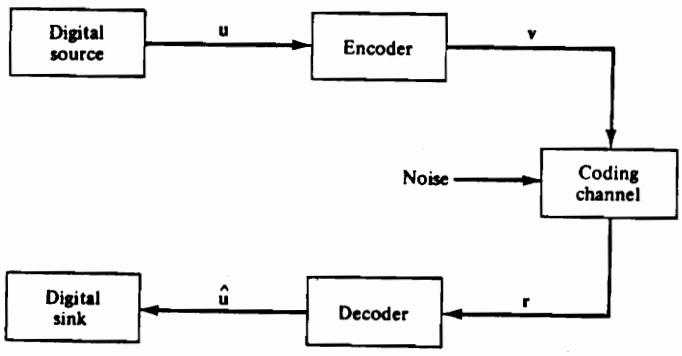

Figure 3-1 Simplified model of a coded system. From Lin and Costello, *Error Control Coding: Fundamentals and Applications* 1983, p. 3.

As there are $M=2^k$ different possible messages; there are also $M=2^k$ possible code words. The set of 2^k code words of length n is referred to as an *(n,k) block code*.

An important term used in the description of codes is the *code rate (R)* which is defined as the ratio of the information bits to the total bits in a code word. It is seen that

$$R = \frac{k}{n} \tag{3-1}$$

If the information bit rate at the input of the encoder is R_b, then the coded bit rate R_c at the output of the encoder is given by Equation 3-2.

$$R_c = \frac{R_b}{R} = \frac{nR_b}{k} \tag{3-2}$$

Practical code rates range from 1/4 to 7/8 with *k* ranging from 3 to several hundreds. Note that since ($n \geq k$) the maximum value for R is one which occurs when the system is uncoded. When $k < n$, $n - k$ redundant bits are added to each message to form a code word. It is these redundant bits that give the system the ability to correct errors caused by channel noise or interference.

3.1.1 Encoding

Encoding is the process of adding redundant digits to the information digits. There are many different ways to encode an (n,k) code. Some of these methods produce *systematic codes* which means that the message part of the code word consists of k unaltered information symbols and the redundant checking part is made of $n - k$ redundant or check symbols. If the information symbols are not explicitly visible in the code word the code is said to be a *non-systematic code.*

In general, to encode a block code, the information sequence or message of length k is multiplied by a generator matrix. Depending upon the generator matrix, the encoding of the information will be different. For example the (7,4) Hamming Code has the following generator matrix:

$$\mathbf{G} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{vmatrix}$$

If one wanted to encode the message **u**=(1101), then performing the modulo-2 matrix multiplication of **uG** would give an encoded message of **v**=(1101001). Note that this is an example of a systematic code because the generator matrix has the form **G** = [**I**|**B**], where **I** is a k x k identity matrix, and **B** is a k by n - k matrix. Using this type of generator matrix produces a code word whose first k symbols are identical to the k bit information word.

An alternative to continuously performing the multiplication would be to simply use a look-up table, since a given same information word always produces the same code word.

3.1.2 Decoding

The basic idea of error correction by a block code is that the code words must be as different as possible from one another. This means that two distinct code words must have a large *Hamming distance*, which is the number of symbols in which the words differ. The *Minimum Hamming distance* of a code is defined as the minimum number of symbols in which any two encoded words differ. This is an important concept because it is possible to decode in such a way as to correct all patterns of t or fewer errors if and only if the minimum distance between code words is at least (2t + 1). So in order to correct two or

more errors, the code must have a minimum distance which is greater than or equal to five.

Several methods' for decoding block codes have been developed. One of these methods called *Maximum Likelihood decoding*, also referred to as *Minimum Distance decoding*, simply corrects the received code word to the nearest actual code word. This type of correction will be considered successful if the number of errors created by the noise is less than one-half the minimum Hamming distance of the code words. One problem with this method is that if too many errors occur, the decoder could mistakenly change the transmitted code word to another code word. Hence the importance of a code with a large minimum distance so this type of mistake will be infrequent.

Another method for decoding block codes is *Syndrome decoding* or *table-lookup decoding*. A syndrome is a binary word computed by the decoder and used in making the decision as to which code word was transmitted. This method results in minimum decoding delay and minimum error probability, but for large n - k, the implementation becomes impractical. Also, either large storage or complicated logic circuitry is needed. For a detailed description of syndrome decoding, Lin and Costello's *Error Control Coding: Fundamental and Applications*, 1983 or Michelson and Levesque's *Error-Control Techniques for Digital Communication*, 1985 provide good coverage on the subject.

3.1.3 Classes of Block Codes

There are many important classes of block codes. Among these are *Hamming Codes, Golay Codes, Bose-Chaudhuri-Hocquenghem* (BCH) *codes, Reed-Solomon codes,* and *maximal-length codes.*

3.1.3.1 The Golay Code. The Golay code was discovered in 1949 be M.J.E. Golay[3] and has the following parameters:

Code length: n=23
Information block length: k=12
Minimum distance: d_{min}=7
Error-correcting capability: t=3

An extended version of the Golay code can be used making it a (24,12) code. When this is done the minimum distance becomes 8 and the code rate, R, becomes 1/2. This code is an important one because of its significant error correcting capability as well as the fact that it is one of the few known "perfect" codes. A perfect code is one for which all error patterns with Hamming weight t or less and no error patterns with weight greater then t are correctable using a minimum-distance maximum-likelihood decoder. If the (24,12) extended Golay code is used it should be noted that this is not a perfect code, but it is often used because its rate is exactly 1/2. The generator polynomial for the (23,12) code is given by one of two different equations:

$$g_1(x) = 1 + x^2 + x^4 + x^5 + x^6 + x^{10} + x^{11} \tag{3-3}$$

or

$$g_2(x) = 1 + x + x^5 + x^6 + x^7 + x^9 + x^{11} \tag{3-4}$$

The same code words will be generated by either generator, although they will have different message associations.

[3] Ziemer and Peterson, p. 397

Decoding the Golay Code

There are several different methods available for decoding the Golay code. Table look-up is possible, but due to the great number of potential code words this could be a time consuming process. One method in particular that can be used is a decoder that has been developed by Kasami. This decoder has been described in great detail in Lin and Costello's *Error Control Coding: Fundamentals and Applications,* 1983. A diagram of the Kasami decoder is shown in Figure 3.2. This decoder begins decoding by shifting the received vector into the syndrome register from the rightmost stage. The syndrome register contents are then tested for correctable error patterns by comparing certain sums of the syndrome to fixed thresholds. There are three syndrome sums which are calculated to estimate the error patterns from which corrections to the received code word can be made. The decoding process continues until the buffer register has been cyclically shifted 46 times, checking the syndrome register contents upon each shift.

Another method of decoding involves a variation of the Kasami error-trapping decoder. This method uses a systematic search decoder. It operates on the principle that if a code word is received with at most 3 errors, and if all the errors are confined to 11 consecutive positions, they can be trapped. That means some cyclic shift of the received word has a syndrome of Hamming weight at most 3, and the syndrome is then equal to the error pattern. If the errors are not confined to 11 consecutive positions, then one suitable bit can be changed such that the remaining errors are confined to 11 consecutive positions. There is an algorithm based on this procedure which will be discussed in detail in subsequent chapters.

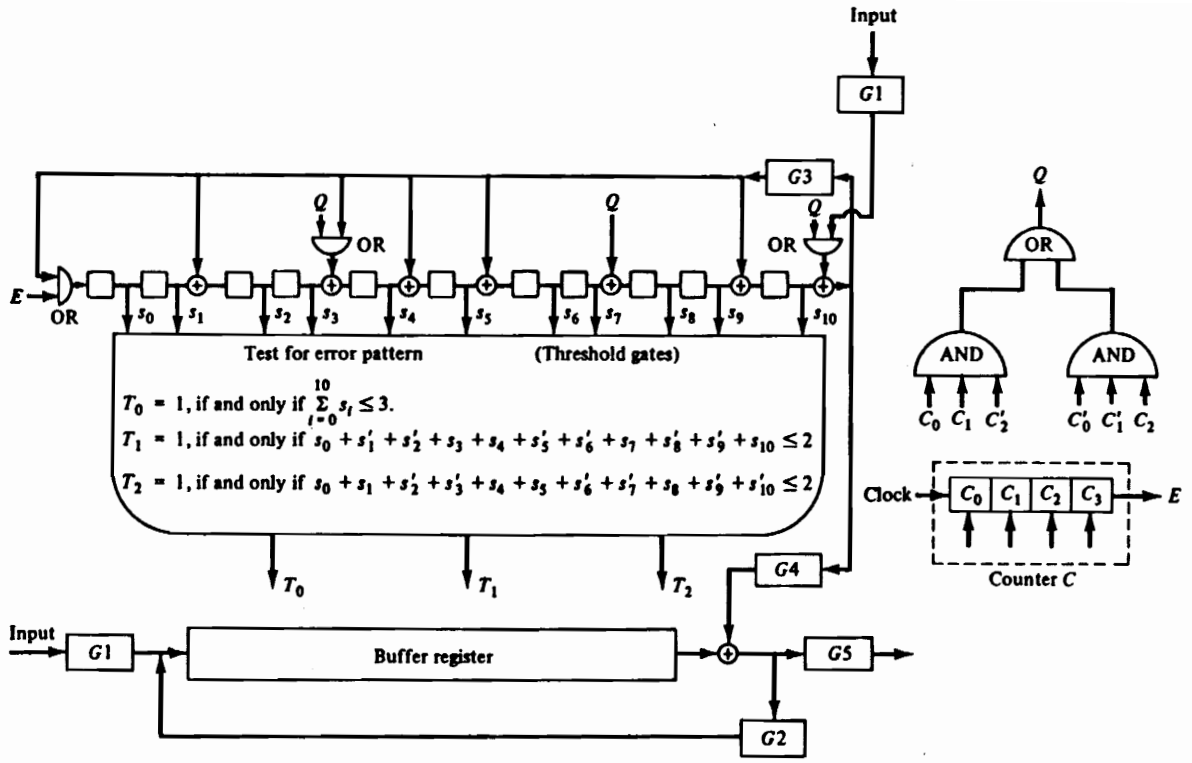

Figure 3-2 Error-trapping decoder for the Golay Code. From Lin and Costello, *Error Control Coding: Fundamentals and Applications*, 1983 p. 136.

It is important to note that both the Kasami and systematic search decoders are capable of correcting all combinations of up to three errors located in the received code word. Some decoders which have been developed, such as the error tapping decoder, do not correct all of the double-error patterns and triple-error patterns which can occur. So, if the Golay code is to be decoded up to its error-correcting capability t=3, then either the Kasami or systematic search decoders can be used.

The bit error probability for a bit in a (23,12) Golay code word is given by Equation 3.5:

$$P_b \leq \frac{1}{12}\sum_{i=4}^{23}\min[12, i+3]\binom{23}{i}p^i(1-p)^{23-i} \qquad (3\text{-}5)$$

where p is the channel bit error rate. The Golay codes have been studied extensively and have been used in modern communication systems, including recent deep space missions.[4]

3.1.3.2 BCH Codes. Bose-Chaudhuri-Hocquenghem (BCH) codes are an important case of block codes since they exist for a wide range of rates and they are also capable of achieving significant coding gains. Another reason for which BCH codes are useful is that due to their decoder design, the code can be implemented at high speeds. BCH codes are linear cyclic codes which are always defined by their code generator polynomial and they have the following properties with $m \geq 3$:

Block length: $n = 2^m - 1$
Information block length: $k \geq n - mt$
Error correcting capability: $t < (2^m - 1)/2$

[4] Yuen 1983.

Not all values which satisfy the above inequalities are possible, but instead specific values for t and k can be found using algebraic techniques for determining code polynomials. One well know application for the BCH codes is in cellular mobile telephony where a shortened BCH code is used for the signaling messages which tell the mobile what power and channel to use.[5]

The BCH codes are cyclic codes so that the encoding process can be performed as follows:

(1) multiply the message polynomial $\mathbf{r}(x)$ by x^{n-k}

(2) calculate the remainder $\rho(x) = R_{g(x)}[x^{n-k}\mathbf{r}(x)]$

(3) generate the code polynomial $\mathbf{u}(x) = \rho(x) + [x^{n-k}\mathbf{r}(x)]$

The steps listed above are easily performed using feedback shift register implementation.

Decoding the BCH Code

Decoding of BCH codes can be performed by syndrome decoding. For a more complete description of the exact algorithm, see Ziemer and Peterson, *Introduction to Digital Communications*, 1992 pp 390-392. The performance of the BCH codes is given by the following equations. The bit error probability can be calculated as follows:

[5] Ziemer and Peterson, p. 390

$$P_b \leq \frac{1}{t}\sum_{i=t+1}^{n}\min[k, i+t]\binom{n}{i}p^i(1-p)^{n-i} \qquad (3\text{-}6)$$

The probability of code word error is found to be:

$$P_{cw} = \sum_{i=t+1}^{n}\binom{n}{i}p^i(1-p)^{n-i} \qquad (3\text{-}7)$$

The best performance (lowest E_b/N_o for a given P_b) for the BCH codes occurs for the 1/2 rate codes.[6] It has also been found that the performance improves with increasing block length n.

3.1.3.3 Reed-Solomon Codes Reed-Solomon codes are an important class of nonbinary BCH codes with the following parameters:

Symbol length: m bits per symbol
Code length: $n = 2^m - 1$
Information block length: $k = n - 2t$
Minimum distance: $d_{min} = 2t + 1$
Error-correcting capability: t symbols

Reed-Solomon codes provide correction for 2^m symbols. They are important codes because of their ability to correct errors which occur in bursts as opposed to independent random errors. The Reed-Solomon (RS) codes achieve the largest possible d_{min} of any linear code with the same encoder input and output lengths. These codes are classified as *maximum-distance-separable* (MDS) codes, or simply *maximum codes* because they have the maximum possible minimum distance for their length and dimension.

The RS code can be generated by generator polynomials or matrices as in the case with other binary BCH codes. The difference in the code generation

[6] Ziemer and Peterson, p. 393

is that the code generator polynomial has coefficients from the nonbinary alphabet {0 ,1, 2, ..., 2^m $^{-1}$} instead of from the binary alphabet {0,1}. Implementation of the RS encoder can be done by using a feedback shift register which uses modulo-2^m addition. Figure 3-3 shows an encoding circuit for the nonbinary RS code.

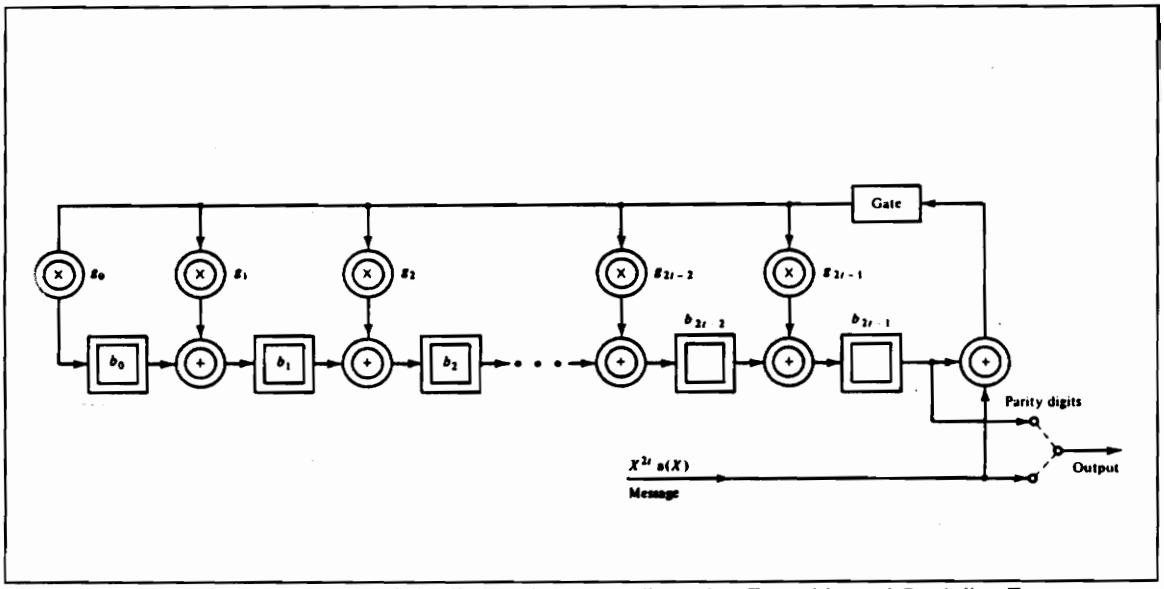

Figure 3-3 Encoding circuit for the nonbinary cyclic code. From Lin and Costello, *Error Control Coding: Fundamentals and Applications*, 1983 p. 172.

Decoding the Reed-Solomon Code

Reed-Solomon codes can be decoded using the same concepts which are used to decode BCH codes. There are four steps needed to decode a t-error correcting Reed-Solomon code. These steps involve computing 2t syndrome components, finding the error-locations polynomial and the error-location numbers, and calculation of the error values. The decoding first begins by calculating a syndrome from the received block and the known structure of

the code. This syndrome is used to determine an error locator polynomial which is then solved to determine the specific error estimates. Errors can then be corrected in the received sequence.[7] This decoding can be accomplished with hardware or software implementation. Hardware implementation has the advantage of speed, while software implementation is less expensive. For further details see Lin and Costello [17], or Assamus and Key [3].

The following equation can be used to measure the performance of the RS code:

$$P_b \le \sum_{i=t+1}^{2^m-1} \frac{i}{2(2^m-2)} \binom{2^m-1}{i} p_s^{\,i} (1-p_s)^{2^m-1-i} \qquad (3\text{-}8)$$

In Equation 3-8 above, the value for p_s is given by Equation (3-9):

$$p_s = 1-(1-p)^m \qquad (3\text{-}9)$$

where p is the bit error rate (BER).

The Reed-Solomon codes have been used in many of today's communication systems. The best known use for the RS code is in the audio recording industry where it is used in recording compact discs. NASA has proposed the use of a RS code concatenated with a convolutional code on the space station.[8]

[7] Ziemer and Peterson, p. 395
[8] Ziemer and Peterson, p. 396

3.2 Convolutional Codes

A *convolutional code* is one in which the encoder accepts information bits as a continuous stream and, for a binary code, generates a continuous stream of encoded bits at a higher rate. In this type of code the information sequence is not grouped into distinct blocks as in block coding. The encoder will break the input message into frames of length k and encode them into code frames of length n.

3.2.1 Encoding

In convolutional coding, the information sequence is passed through a linear shift register which contains M stages and shifts k bits at a time. There will be n linear logic circuits, for every M information bits stored in the shift register, which operate on the shift register contents to produce n coded bits at the encoder output. Like block coding, the code rate R of a convolutional code will be R=k/n. However, unlike block coding, convolutional codes contain memory. This is because a particular information bit remains in the shift register for a total of M/k shifts, thus influencing the value of nM/k coded bits. Typical values for n and k are in the range of 1 to 8, while R is typically 1/4 to 7/8. The value M usually ranges from 5 to 70. Figure 3-4 shows a sample encoder with M=3, k=1, and n=2.

For the encoder of Figure 3-4, when one bit is shifted into the shift register, there will be two bits at the output. The code tree of Figure 3-5 shows all the possible inputs into the shift register and the resulting output. For each

branch on the tree, as you move to the right, there are two different ways to continue. Moving up the branch represents a 0 input, and moving down means that a 1 has been shifted into the register. By following a path along the tree from left to right, the resulting convolutional code can be quickly found. For example, if the data stream x=1010 was fed into the encoder of Figure 3-4, the output code stream would be y=11010001. This can be found by following path A in Figure 3-5. The first data

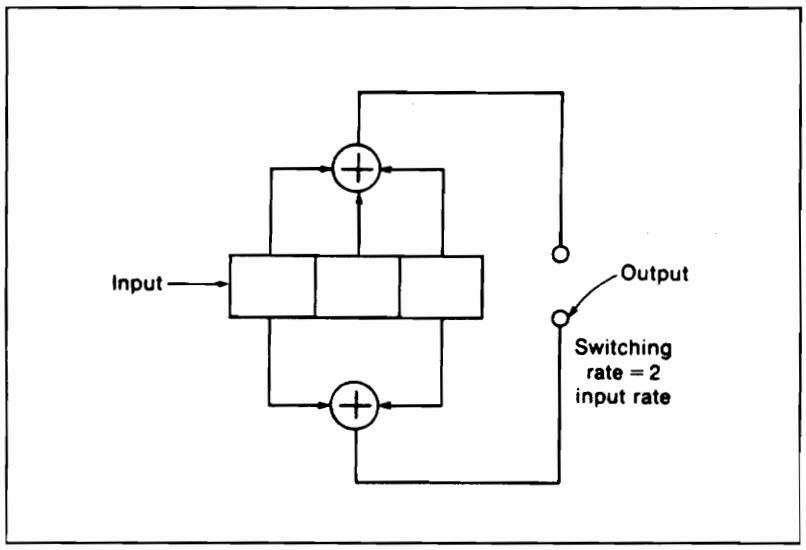

Figure 3-4 Convolutional encoder with M=3, k=1, and n=2. From Tri T. Ha, *Digital Satellite Communications*, McGraw-Hill Inc. 1990 p. 458.

bit fed into the register is a 1, so starting at the far left of the tree and moving down yields an output of (11). The next input bit is a 0. Moving up the branch shows an output of (01). The third input bit is a 1 which means you have to move down the next branch giving (00) at the output. The final bit of the data stream is a zero, and when moving up the next branch the final two output bits will be (01). Putting the four output combinations together yields a coded data

stream of (11010001). The code tree can be extended infinitely depending upon the size of the input data stream.

3.2.1 Decoding

There are several different techniques or algorithms available for decoding convolutional codes, namely, the Viterbi decoding, sequential decoding, and syndrome decoding.[9] The most common of these three, is the Viterbi algorithm because theoretically it gives the minimum error probability. This algorithm attempts to choose a path through a trellis diagram that matches the received sequence **r** as closely as possible. It then processes **r** in an iterative manner by comparing the paths entering each node of the trellis. The optimum Viterbi procedure uses a process which involves making soft-decisions. A soft-decision algorithm first decides the result based on the test statistic (the test statistic is a value that is computed at the receiver based on the receiver input during some specified time interval) being above or below a decision threshold and then gives a "confidence" number that specifies how close the test statistic was to the threshold value. In hard decisions only the decision output is known.

When viewing the performance of a convolutional code, the probability of bit error is overbounded by:

$$P_b < \frac{T(D)}{k} \tag{3-10}$$

The parameter, k, in Equation 3-10 is the number of information bits being encoded. The polynomial T(D) is described by the following:

[9] Tri T. Ha p. 469

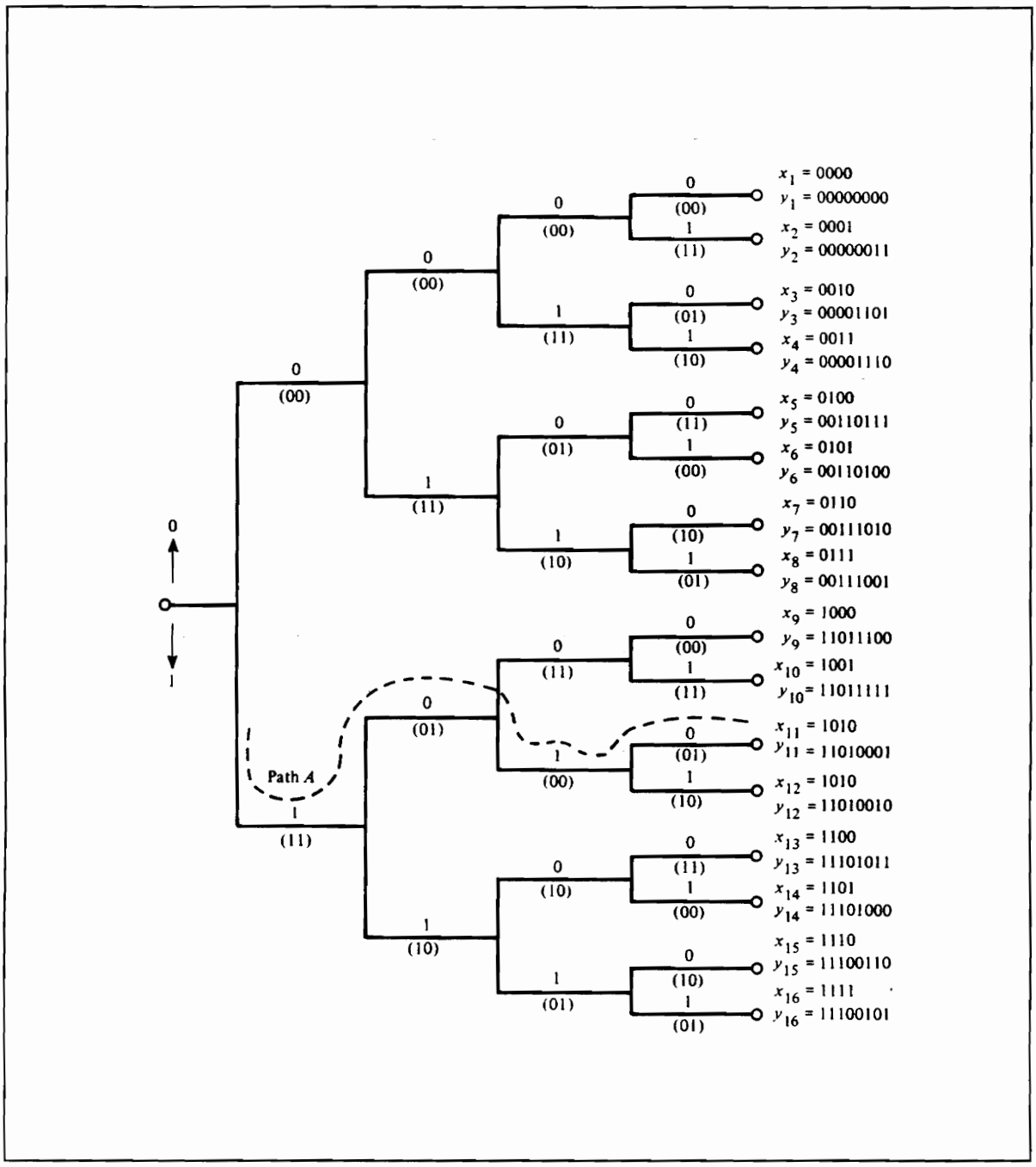

Figure 3-5 Code tree for convolutional encoder of Figure 3-4. From Leon W. Counch II, Digital and Analog Communication Systems, 1990 p. 27.

$$T(D) = \sum_{k=d_{free}}^{\infty} c_k D^k \qquad (3\text{-}11)$$

where $D = 2\sqrt{(p(1-p))}$ and p is the probability of bit error or simply the bit error rate. The values for c_k are derived from the total number of bit errors associated with all paths which are Hamming distance k from the all-zero path of the trellis diagram associated with the convolutional code. The variable d_{free} is called the free distance and is defined as the minimum Hamming distance between two arbitrarily long encoded sequences, one corresponding to an initial information bit $i_o = 0$ and the other beginning with $i_o = 1$.[10] The values of c_k are given in Tables 3-1 through 3-4. In these tables, a value known as the constraint length, ν, is shown. This value has two different definitions, but the one used here is that it is equal to one plus the number of past inputs affecting the current outputs. Independent of the definition used, the constraint length is a measure of the encoder memory. Note that only a few values for the polynomial T(D) were given. This is because the higher order terms become too small to make a significant contribution to the bit error probability.

[10] Djimitri Wiggert p 105

Table 3-1 1/2 Rate Convolutional Codes

Constraint Length, v	Free Distance, d_f	c_k for d = d_f	d_f+1	d_f+2	d_f+3	d_f+4	d_f+5
6	8	2	36	32	62	332	701
7	10	36	0	211	0	1404	0
8	10	2	22	60	148	340	1008
9	12	33	0	281	0	2179	0

Table 3-2 1/3 Rate Convolutional Codes

Constraint Length, v	Free Distance, d_f	c_k for d = d_f	d_f+1	d_f+2	d_f+3	d_f+4	d_f+5
5	12	12	0	12	0	56	0
6	13	1	8	26	20	19	62
7	14	1	0	20	0	53	0
8	16	1	0	24	0	113	0

Table 3-3 2/3 Rate Convolutional Codes

Constraint Length, v	Free Distance, d_f	c_k for d = d_f	d_f+1	d_f+2	d_f+3	d_f+4
6	6	75	0	1571	0	31474
7	6	1	81	402	1487	6793
8	8	395	0	6695	0	235288
9	8	97	0	2863	0	56633

Table 3-4 3/4 Rate Convolutional Codes

Constraint Length, v	Free Distance, d_f	c_k for d =				
		d_f	d_f+1	d_f+2	d_f+3	d_f+4
6	5	78	572	3831	24790	152108
7	6	919	0	31137	0	1142571
8	6	117	0	8365	0	319782
9	6	12	342	1996	12296	78145

3.3 Error Control in AX.25 Packet Radio

A major concern in data communications is how to control transmission errors caused by the channel noise so that the user can receive error-free data. One approach to this problem is to use error-detecting or error-correcting codes. For controlling errors in data communications, two basic categories have evolved: Automatic-Repeat-Request (ARQ) schemes and forward-error-correction (FEC) schemes.[11] Forward-error-correction involves employing error-correcting codes, some of which have been previously discussed.

3.3.1 ARQ

In an ARQ error-control scheme a code with good error-detecting capability is used. If the receiver detects an error in the received packet, it will discard it, and request a re-transmission of that packet. Re-transmission will continue until the packet is successfully received or the system gives up because it cannot get a clean packet.

[11] Lin, Costello, and Miller p. 5

In this type of system, the user will receive erroneous data only if the detector fails to detect the presence of errors. If a good error-detecting code is used the probability of this happening will be very small. ARQ systems are widely used because they provide high system reliability, but one problem is that the throughput of the decoder is not constant and it falls rapidly as the channel error rate increases. There are three basic types of ARQ schemes: stop-and-wait ARQ, go-back-N ARQ, and selective-repeat ARQ.

Stop-and-Wait ARQ

The stop-and-wait ARQ scheme is the simplest of the three different types. The transmitter sends a packet to a receiver and waits for an acknowledgment. If a positive acknowledgment (ACK)

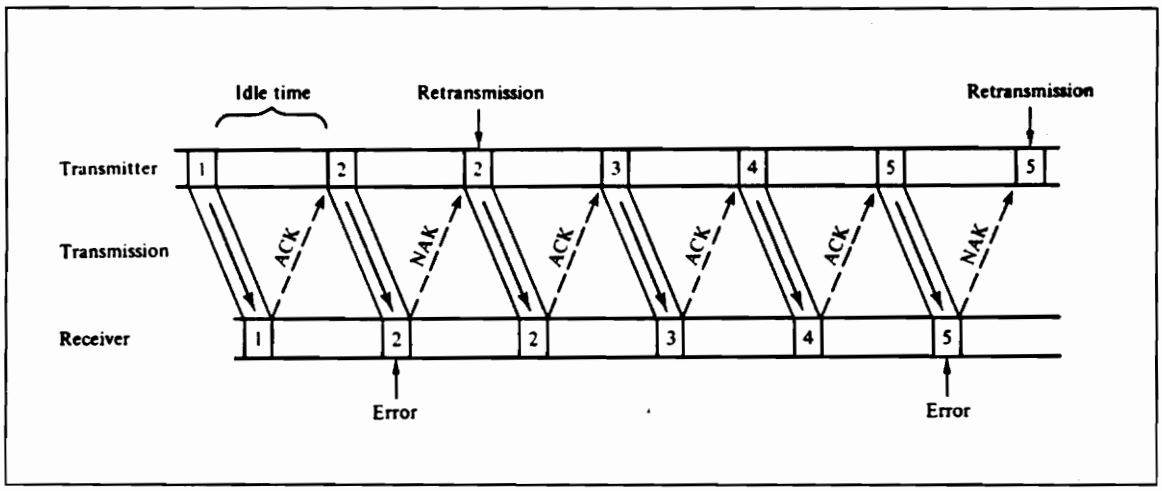

Figure 3-6 Stop-and-wait ARQ. From Lin and Costello, *Error Control Coding: Fundamentals and Applications*, 1983 p. 459.

is received then the packet has been received without error and the next packet can be transmitted. If a negative acknowledgment (NAK) is received then this indicates the receiver has detected an error in transmission. The transmitter then re-transmits the packet and again waits for an acknowledgment. Re-transmission will continue until an ACK is received at the transmitter. Figure 3-6 demonstrates the stop-and-wait ARQ system.

One measure of the performance of an ARQ system is its *throughput efficiency* or simply *throughput*. The throughput is defined as the ratio of the average number of information digits successfully accepted by the receiver per unit of time to the total number of digits that could be transmitted per unit of time.[12] For the stop-and-wait ARQ system, the throughput is given by:

$$\eta_{sw} = \frac{(1-P_p)}{1+D\tau/n}\left(\frac{k}{n}\right) \tag{3-12}$$

In Equation 3.12, the parameters are as follows:

η_{sw} = throughput for stop-and-wait ARQ
P_p = probability that a received packet has an error
n = total number of bits in the received packet
k = total number of information bits in the received packet
D = delay between transmission of packets
τ = transmitter data rate

The stop-and-wait ARQ scheme is not very efficient because of the idle time spent waiting for an ACK of each transmitted packet. Because higher data rates and satellites with long round-trip delays are widely used, a different, more efficient, ARQ system is needed. One of these is the go-back-N ARQ.

Go-Back-N ARQ

Figure 3-7 illustrates the basic go-back-N ARQ scheme. The transmitter continuously transmits packets in order and then stores them pending receipt of an ACK or NAK for each. The acknowledgment for a packet will arrive after a round-trip delay, which is defined as the time interval between the transmission of a packet and the receipt of an acknowledgment for that packet. During this

[12] Lin, Costello, and Miller, pp 7-8

interval, N - 1 other packets are also transmitted. When the transmitter receives a NAK it will stop transmitting new packets. It will then go back to the erroneous packet indicated by the NAK and re-transmit that packet and the N - 1 succeeding packets which were transmitted during one round-trip delay. The receiver will discard the erroneously received packet and all of the N - 1 following packets, whether they contained an error or not. Re-transmission will occur until the packet which was originally received in error has been positively acknowledged. The transmitter will not send new packets until that positive acknowledgment has been received.

The throughput for the go-back-N ARQ is given by Equation 3.13:

$$\eta_{GBN} = \frac{(1-P_p)}{(1-P_p)+P_pN}\left(\frac{k}{n}\right) \tag{3-13}$$

The parameters are the same in this equation as in Equation 3.3, except that there is a new parameter, N, which is the number of packets which must be re-transmitted.

The main problem with this type of ARQ method is that when an error is detected, the receiver will also reject the next N - 1 received packets. This means that they must be re-transmitted which can cause a deterioration in the throughput of the system when large round-trip delays are involved. The go-back-N ARQ system can become inefficient for communication systems with high data rates and large round-trip delays, due to the re-transmission of many error-free packets following a packet in which an error was detected. In order to overcome this, a scheme called selective-repeat ARQ can be implemented.

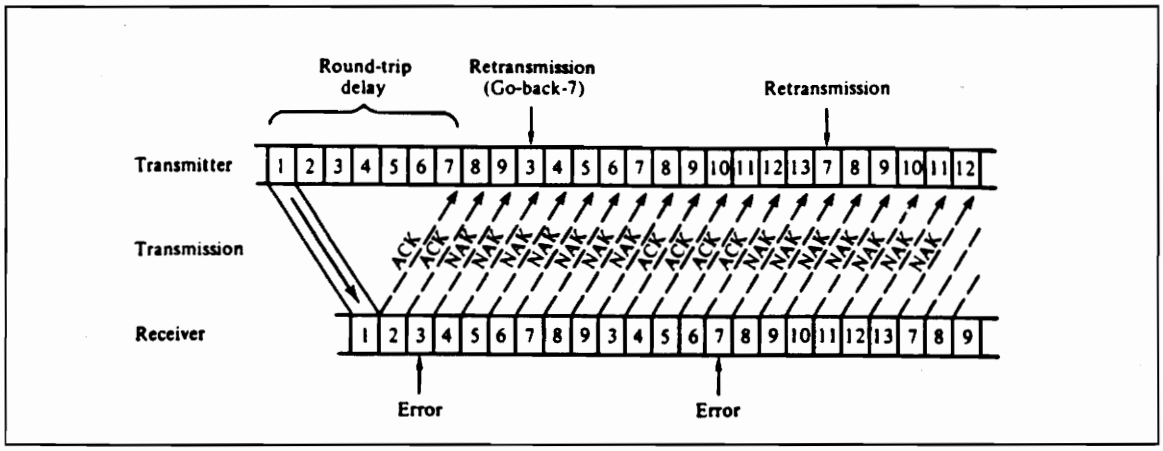

Figure 3-7 Go-Back-N ARQ with N=7. From Lin and Costello, *Error Control Coding: Fundamentals and Applications*, 1983 p. 460.

Selective-Repeat ARQ

In selective-repeat ARQ, shown in Figure 3-8, the transmitter will only re-transmit the packets for which a NAK has been received. The transmitter will then continue sending new packets. If this type of ARQ is used, a buffer is needed at the receiver in order to store error-free packets following a received packet in which an error is detected. This is because, generally, packets must be delivered in the correct order to the user. When the first negatively acknowledged packet is successfully received, the receiver will release the error-free received packets in consecutive order until the next erroneously received packet is encountered. The selective-repeat ARQ is the most efficient of the three types of ARQ, but is the most complex to implement. This is because messages must be numbered in sequence, and message buffering is required in both the transmitter and receiver. The messages must be stored in the transmitter until they are acknowledged as correctly received by the receiver. The receiver must also buffer messages so that all messages can be delivered to the user in the correct order.

The throughput for the selective-repeat ARQ is given by:

$$\eta_{SR} = (1 - P_p)\left(\frac{k}{n}\right) \tag{3-14}$$

The parameters for this equation are the same as those used by Equation 3.12.

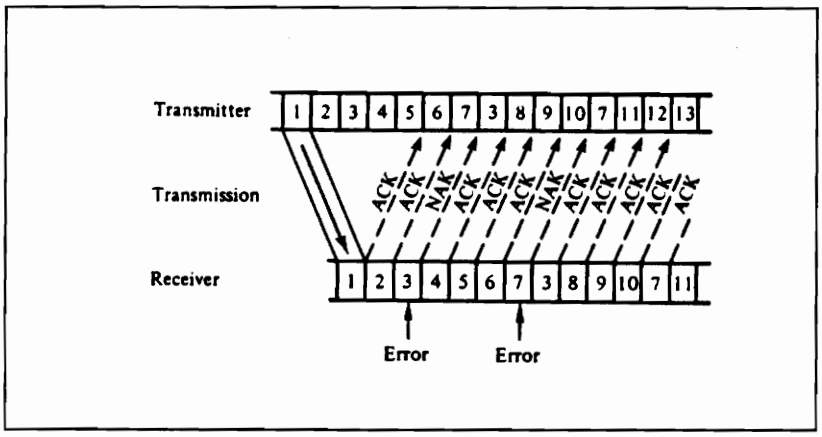

Figure 3-8 Selective-Repeat ARQ. From Lin and Costello, *Error Control Coding: Fundamentals and Applications*, 1983 p. 460.

3.3.1.1 Mixed Modes of Re-transmission. When examining the above ARQ schemes it is seen that the throughput efficiency could be increased if mixed modes of re-transmission were used. One such possible combination is the *selective-repeat plus go-back-N* (SR + GBN) shown in Figure 3-9.
When a packet x in the re-transmission buffer becomes the earliest negatively acknowledged packet, the transmitter will re-transmit x and other packets in the SR mode. While in this mode the receiver will store packets which have been successfully received. At this point two different events can occur. The transmitter will stay in the SR mode until the packet x is positively acknowledged or ν re-transmissions of packet x have been made but the transmitter has failed to receive an ACK for that packet. If the packet has been positively acknowledged then the transmitter will send a new packet. If the second event occurs, the transmitter will switch to the go-back-N mode. It will then stop

sending new packets, back up to packet x and re-transmit that packet along with the N - 1 succeeding packets that were transmitted following the νth SR re-transmission of packet x. The transmitter will remain in

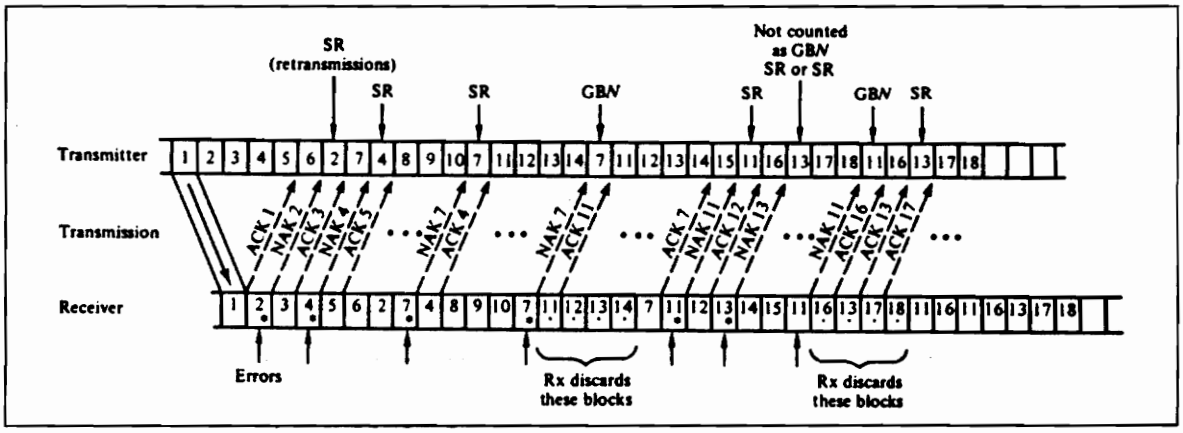

Figure 3-9 SR + GBN ARQ for v=1 and N=5. From Lin and Costello, *Error Control Coding: Fundamentals and Applications*, 1983 p.476.

the GBN mode until x has been positively acknowledged. At the receiver, when the second transmission attempt of packet x is detected in error, the following N - 1 received packets will be discarded.

This scheme will have better performance than the GBN ARQ. The throughput for this type of system is given by 3.15:

$$\eta_{SR+GBN} = \frac{(1-P_p)}{1+(N-1)(P_p)^{\nu+1}}\left(\frac{k}{n}\right) \qquad (3\text{-}15)$$

In above equation, ν is the number of re-transmissions in the SR mode allowed before the transmitter switches to the GBN mode.

3.3.1.2 Hybrid ARQ. Hybrid ARQ is the term given to systems which employ a combination of ARQ and FEC. This type of system consists of an FEC subsystem contained in an ARQ system. The FEC serves to reduce the

frequency of re-transmission by correcting some of the errors which have occurred in transmission. This can greatly improve the system throughput efficiency since the packets would not have to be re-transmitted upon every occurrence of an error if some or all of the errors could be corrected upon reception.

3.3.1.3 AX.25 Packet Radio. The AX.25 packet radio system uses the go-back-N ARQ scheme for system error control. For detection of errors, there is a *frame check sequence (FCS)* field included in each transmitted packet. This is a 16-bit number that is calculated from the transmitted data by the transmitting station. When the packet is received, the receiving station will recalculate the FCS and if it matches the transmitted FCS then no errors have occurred in transmission. If the two values differ then an error in transmission has occurred and the receiving station will discard that packet.

The value for N, which is the number of packets that must be re-transmitted in the go-back-N scheme, will depend upon several different factors. Some of these include the transmitting bit rate, the round trip delay, and size of the packet being transmitted. In order to make any throughput calculations of the system throughput, a value for N must be found. In order to do this the calculations will be based upon packets which are 2200 bits in length. In the AX.25 transmission procedure, the maximum number of frames that can be outstanding is 7 which is equivalent to 15.4 kbits. The time allowed for the T1 timer to expire is at least twice the amount of time it takes to transmit the longest possible frame plus the time required for a proper acknowledgment. The longest possible frame is 2656 bits long prior to bit stuffing, so twice this amount gives 5312 bits. Typical delays run on the order of about 3 ms. The time required for the receiver to transmit a proper response or RR frame, which contains 144 bits,

will be 15 ms at a 9.6 kbps transmission rate plus the delay time. Using the above times and bits which must be transmitted, gives a total of nearly 22.2 kbits to be re-transmitted. This works out to be about 10 packets. Therefore, the value for N in the go-back-N scheme will be 10 for future calculations.

If a satellite system is being used, then the value of the acknowledgment timer should be set to 6 seconds.[13] This would increase N to nearly 35. At such a value, the throughput would dramatically fall off with increasing bit error rate since many packets will have to be retransmitted when a packet error occurs.

[13] Horzepa, p 12-6

CHAPTER 4

INTERLEAVING

When designing a packet transmission system one of the key factors that must be taken into account is how that system will perform in the presence of noise. When sending packets through the atmosphere, transmission errors tend to occur in clusters or bursts. This means that the disturbance that causes the error lasts longer than the space required for one symbol and consequently several adjoining symbols or bits will be affected. The noise disturbance is usually a stroke of lightning or a man-made electrical disturbance such as that produced by electric motors or high-tension transmission lines. In general the codes developed for correcting random errors are not efficient for correcting burst errors. Therefore, it is desirable to design codes specifically for correcting burst errors or else to develop techniques to handle the problem.

4.1 Transmission Line Noise

When studying the noise generated from transmission lines it can be seen that this type of disturbance is periodic. In the United States the standard transmission lines are 60 Hz AC lines. This means that the line voltage will reach its peak absolute value every 8.33 msec. Research has shown that at each of these 8.33 msec intervals a burst of noise will occur that may last for 1 msec.

In looking to see how this will affect a transmitted packet it can be seen that the length of a burst of errors will be dependent upon the rate of

transmission of the packet and the length of the disturbance. For example, if the disturbance lasts for 1 msec and the transmission rate is 9.6 kbps, then 9.6 or approximately 10 adjoining bits will be corrupted. It can be seen that as the transmission rate is increased then the length of the burst of errors in the transmitted packet will increase for the same disturbance length. Below is a table illustrating some sample disturbance lengths, packet transmission rates and the length of the resulting burst error:

Transmission Rate	Disturbance Length	Number of Corrupted Bits
4.8 kbps	1.0 msec	5 bits
9.6 kbps	1.0 msec	10 bits
9.6 kbps	1.5 msec	15 bits
19.2 kbps	1.0 msec	20 bits
19.2 kbps	1.5 msec	29 bits
56 kbps	1.0 msec	56 bits

Table 4-1 Number of corrupted bits for different transmission rates and disturbance lengths.

The results in table 4-1 should make it clear that as the transmission rate is increased then the problem of burst errors can become quite significant. When designing a system to combat this problem it has to be remembered that the noise bursts from the transmission lines are periodic and that with a 9.6 kbps transmission system after approximately every 80 transmitted bits a 1 msec burst of noise may occur that corrupts nearly 10 bits.

4.2 Burst Error Correction

The general principle behind burst error correcting codes is that the parity check bits are spaced so widely apart that several successive bits, obliterated by noise, can be reconstituted correctly. There have been many various codes developed to combat error-bursts such as Hagelbarger's, Fire,

and Reed-Solomon Codes. However; instead of developing new codes there is one process called interleaving which can be used on existing codes to protect against error-bursts.

One method of interleaving is accomplished by separating the data stream into a number of parallel streams, each of which is encoded separately. The encoded sequences are then commutated for serial transmission. At the receiving end the data stream is split again, where each stream is decoded, and then the decoder outputs recombined to give the original data. This technique can be used for either block codes or convolutional codes. In this type of system if the number of encoders (m) is sufficiently large, bursts of errors are spread out over the m encoded data streams. This is beneficial because of the need for a less powerful error-correction scheme on any of the m streams.

The method presented above can be implemented with only one encoder and one decoder when using convolutional encoding. In this design the interleaver is placed between the encoder and a multiplexer, and separates the n encoded bits in a block by λ-1 intervening bits prior to transmission. In addition, the delay units required in the encoder are replaced by a string of λ delay units. The encoder becomes equivalent to λ separate encoders whose n-bit encoded blocks are formed in succession. It also ensures that there will be λ -1 intervening bits between the last bit in one block and the first bit in the next block corresponding to the same encoder. By using this procedure an interleaving degree of λ can be achieved for the original convolutional code.

4.3 Cyclic Codes

When dealing with cyclic codes, such as the Golay, Hamming, and BCH cyclic codes, if an (n,k) code is given, than a (λn,λk) interlaced code can be produced by arranging λ code vectors in the original code into λ rows of a

rectangular array and then transmitting them column by column. The parameter λ is referred to as the interlacing (or interleaving) degree. A pattern of errors can be corrected for the whole array if and only if the pattern of errors in each row is a correctable pattern for the original code. No matter where it starts, a burst of errors of length λ will affect no more than one digit in each row. This means that if the original code corrects single errors, then the interlaced code corrects single bursts of length λ or less. Similarly, if the original code corrects t or fewer errors, the interlaced code will correct any combination of t bursts of length λ or less. If the original code is capable of correcting any single burst of length ℓ or less, the interlaced code will correct any single burst of length $\lambda\ell$ or less. Thus, instead of searching for long efficient burst-error-correcting codes one only needs to search for good short codes.

It should be noted that there are many different ways to achieve interleaving. For example, interleaving could be done on the packets themselves or in the case of block codes on the individual code words. Interleaving could also be performed on each bit. It is also possible to interleave the data before it is coded just as well as after it has been encoded.

When deciding on whether or not to use interleaving one has to consider the length requirements and the time delays involved. When interleaving is performed, the coded bits are shuffled over a time span of several block lengths (for block codes) or constraint lengths (for convolutional codes). The required span length has to be several times the duration of the noise burst. For example, with the periodic noise bursts from the transmission lines which last approximately 1 msec, a span length of 30-40 bits would be sufficient to protect against these error-bursts when dealing with a 9.6 kbps transmission rate.

One consideration is the delay introduced into the system by interleaving. In order to form an array consisting of λ code words in λ rows at the receiver, the

system has to wait until λ code words are received before the de-interleaving process can begin.

To illustrate this let's look at the following example. Given a transmission rate of 9.6 kbps and knowing that noise bursts are periodic and occur every 8.33 msec with a duration of 1 msec a system needs to be designed to protect against this. Using a (7,3) cyclic code with an interleaving degree λ=5 should be sufficient for this case. The (7,3)

31	30	•	•	11	10	1
32	29	•	•	12	9	2
33	•	•	18	13	8	3
34	•	•	17	14	7	4
35	•	•	16	15	6	5

Figure 4-1 Interleaving Array and location of random burst errors (•)

cyclic code has a burst-correcting capability of 2 and when interleaving is used with λ=5, the burst-correcting capability rises to 10. For the rate given above the noise bursts will corrupt approximately 10 bits, therefore with the (7,3) code there is no extra margin built into the system. If needed, λ could be raised to six or seven in order to account for any other random errors or bursts that last longer than 1 msec, but here λ=5 will be used just for illustration. Figure 4.1 shows the interleaving array that will be produced by this system. Once the bits have been arranged in this manner, transmission will be carried out column by column starting with bit '1' and ending with bit '35' in Figure 4-1. Then at the receiving end, the data will be arranged into the array of Figure 4-1. When this happens it is seen that for a noise burst which corrupts 10 bits, only a maximum

of two bit-errors could occur in any given code word which is represented by a row in Figure 4-1. Since the (7,3) cyclic code can correct bursts of up to 2 bits, then it is capable of correcting each of the code words received.

The delay in the previous example can be found in the following manner. In a non-interleaving system the code word (which has a length of 7 bits) can immediately enter the decoder as it is received. In the above example, the receiver has to wait until 35 bits (the length of five code words) are received so the incoming bits can be de-interleaved. This means the system will have a delay equal to the amount of time it takes 28 bits to reach the receiver plus the delay of one code word for a total of 35 bits. In a 9.6 kbps transmission system this results in a 2.92 msec delay introduced for every 35 bits received. In the AX.25 packet whose maximum length is 3184 bits(after bit stuffing) this would cause a maximum packet delay of 0.618 seconds at the receiver with the (7,3) cyclic code.

In conclusion, it has been seen that interleaving can be a very useful technique in combating burst errors. Instead of looking for long efficient burst-error-correcting codes it is only necessary to find good short codes because, in general, bursts of length $t\lambda$(λ is the interleaving degree) on the channel will look like bursts of errors of length t to each of the separate decoders. Therefore the decoder has to be capable of correcting bursts of length t.

CHAPTER 5

CODING THE AX.25 RADIO PACKET

The AX.25 protocol, described in chapter 3, uses the Go-Back-N automatic-repeat-request (ARQ) scheme for error control. This type of system only detects errors with no attempt to correct them. In hybrid ARQ, a forward error correction (FEC) scheme is employed in conjunction with the ARQ error detection system in order to correct possible transmission errors. Using hybrid ARQ can improve the system throughput as shown by Figure 5-1. The improvement in performance arises because the FEC is able to correct some transmission errors. If all the errors occurring in a packet are corrected, then re-transmission of that packet will not be necessary, thereby increasing the system throughput.

5.1 Hybrid ARQ Performance

ARQ error control schemes provide error free communications, but some of these systems prove to be inefficient. The probability of packet error greatly increases with the bit error rate. The average probability of packet error without coding is simply 1 minus the probability that all q bits will be received correctly,

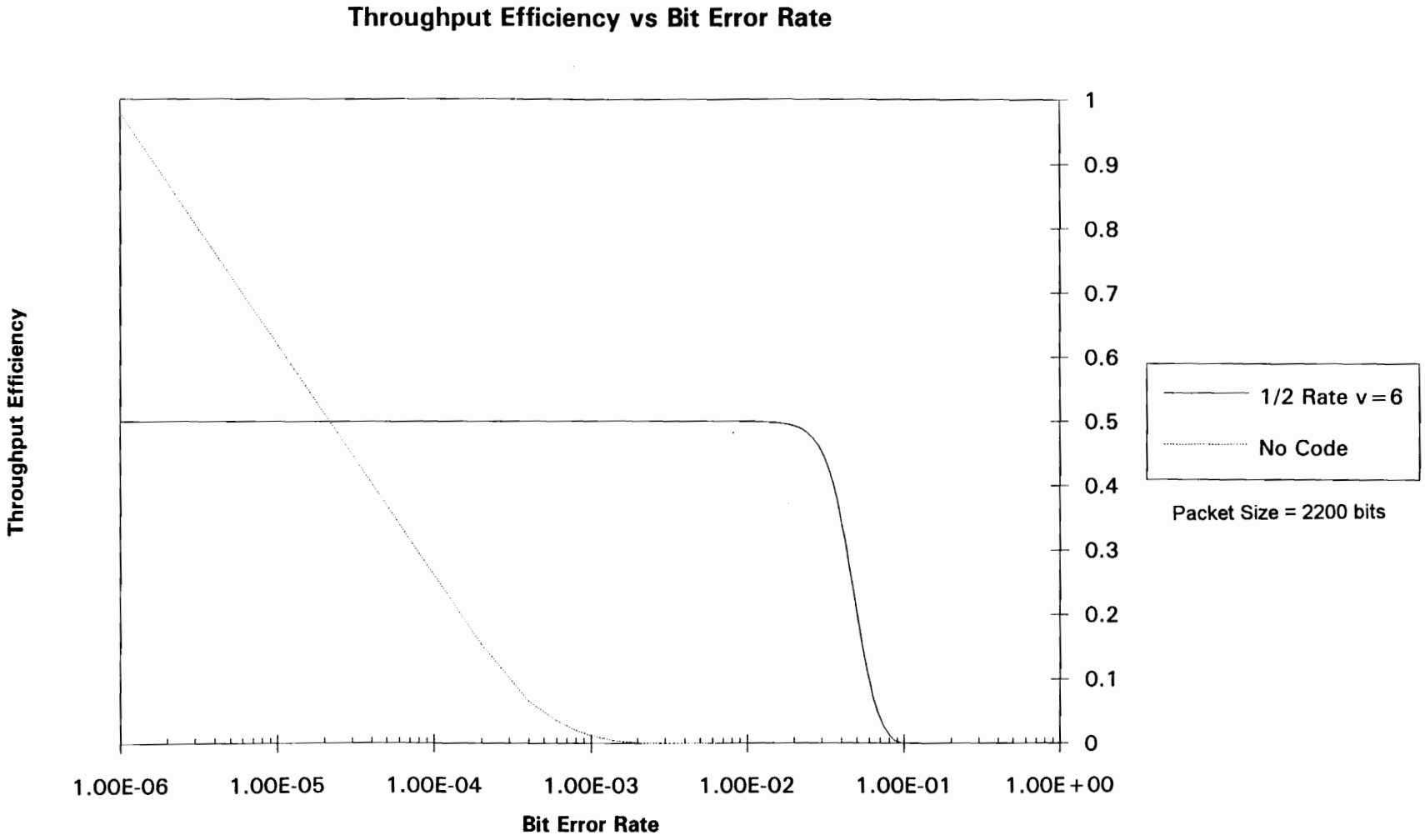

Figure 5-1 Plot of the throughput efficiency vs bit error rate for a packet with no coding and a packet with convolutional 1/2 rate v=6. The system uses Go-Back-N ARQ.

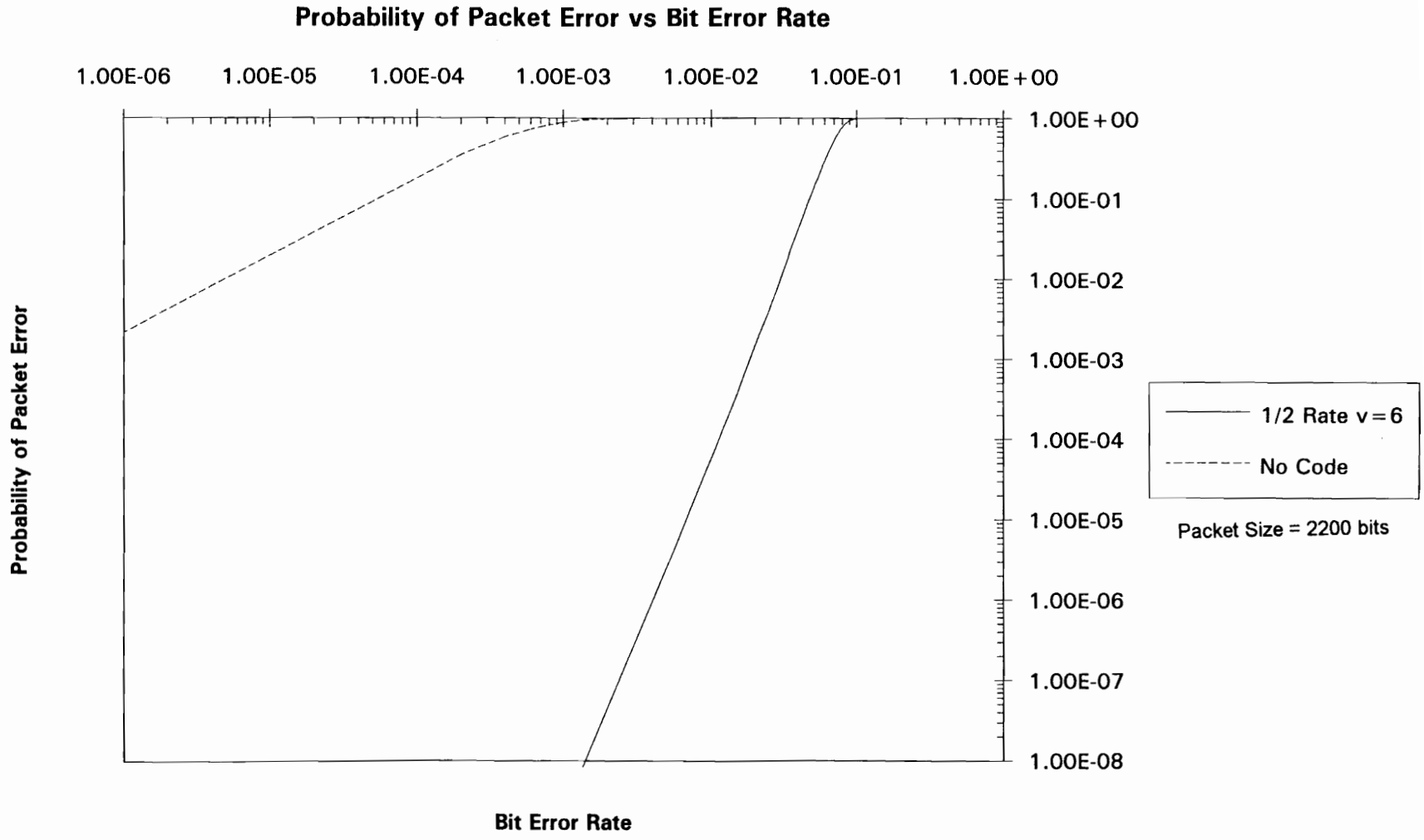

Figure 5-2 Plot of the probability of packet error vs bit error rate for a packet with no coding and a packet with convolutional 1/2 rate v=6.

which is given by:

$$P_{pe} = 1-(1-p)^{q} \tag{5-1}$$

where p is the bit error rate. Figure 5-2 shows a plot of equation 5-1 vs. the performance of a 1/2 rate convolutional code. For the uncoded system, the probability of packet error is fairly high, meaning that many of the received packets will contain errors and therefore have to be re-transmitted more often then when the hybrid ARQ is used.

5.1.1 Probability of Packet Error

The probability of packet error for a hybrid ARQ system is dependent upon the type of error correction code used. For a block code, the probability of packet error is given by:

$$P_{pe} = 1-(1-P_c)^{q} \tag{5-2}$$

In this equation, P_c is the probability of code word error, which is dependent upon the type of code used, and q is the number of code words in the packet. For a convolutional code the probability of packet error is:

$$P_{pe} = 1-(1-P_b)^{k} \tag{5-3}$$

P_b is given by equation 3-10, ($P_b < T(D)/k$) and k is the number of bits in the packet divided by the rate of the code. For example, if the packet is 2200 bits, and a 1/2 rate convolutional code is used, then k would be 4400.

The Figures 5.3 through 5.10 show the probability of packet error vs. the channel bit error rate for various codes. In each case an information packet of 2200 bits was used.

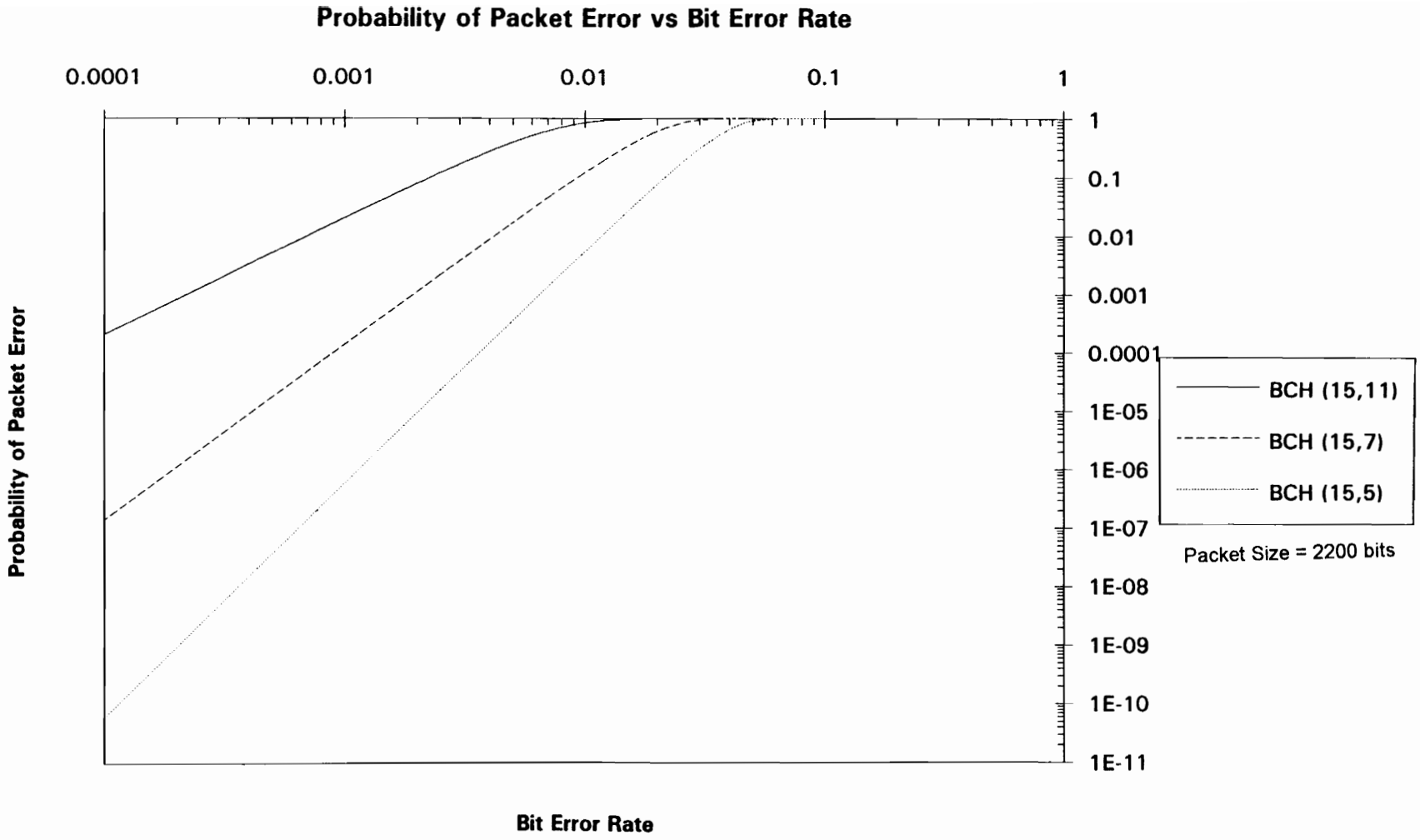

Figure 5-3 Plot of the probability of packet error vs bit error rate for various length BCH codes.

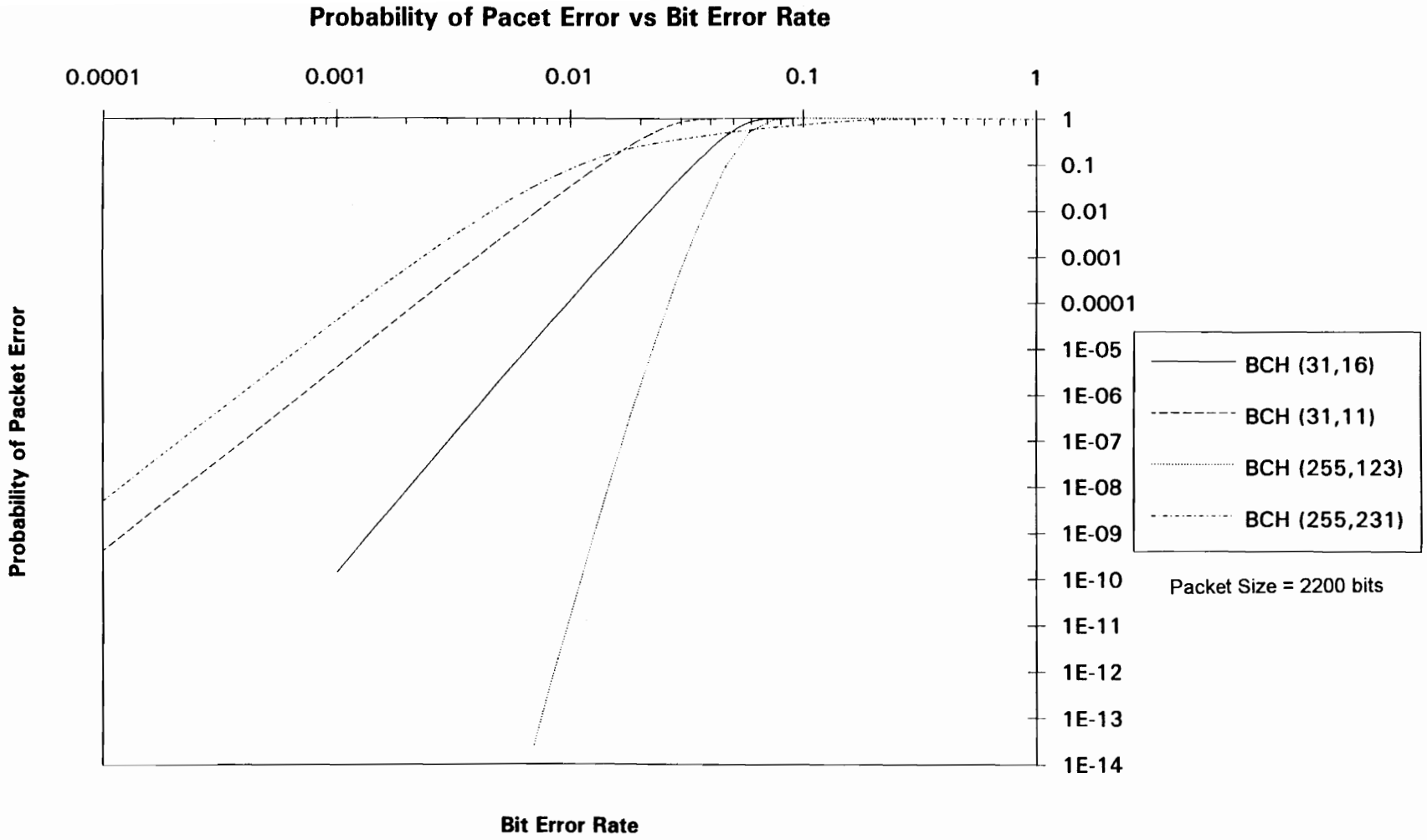

Figure 5-4 Plot of the probability of packet error vs bit error rate for various length BCH codes.

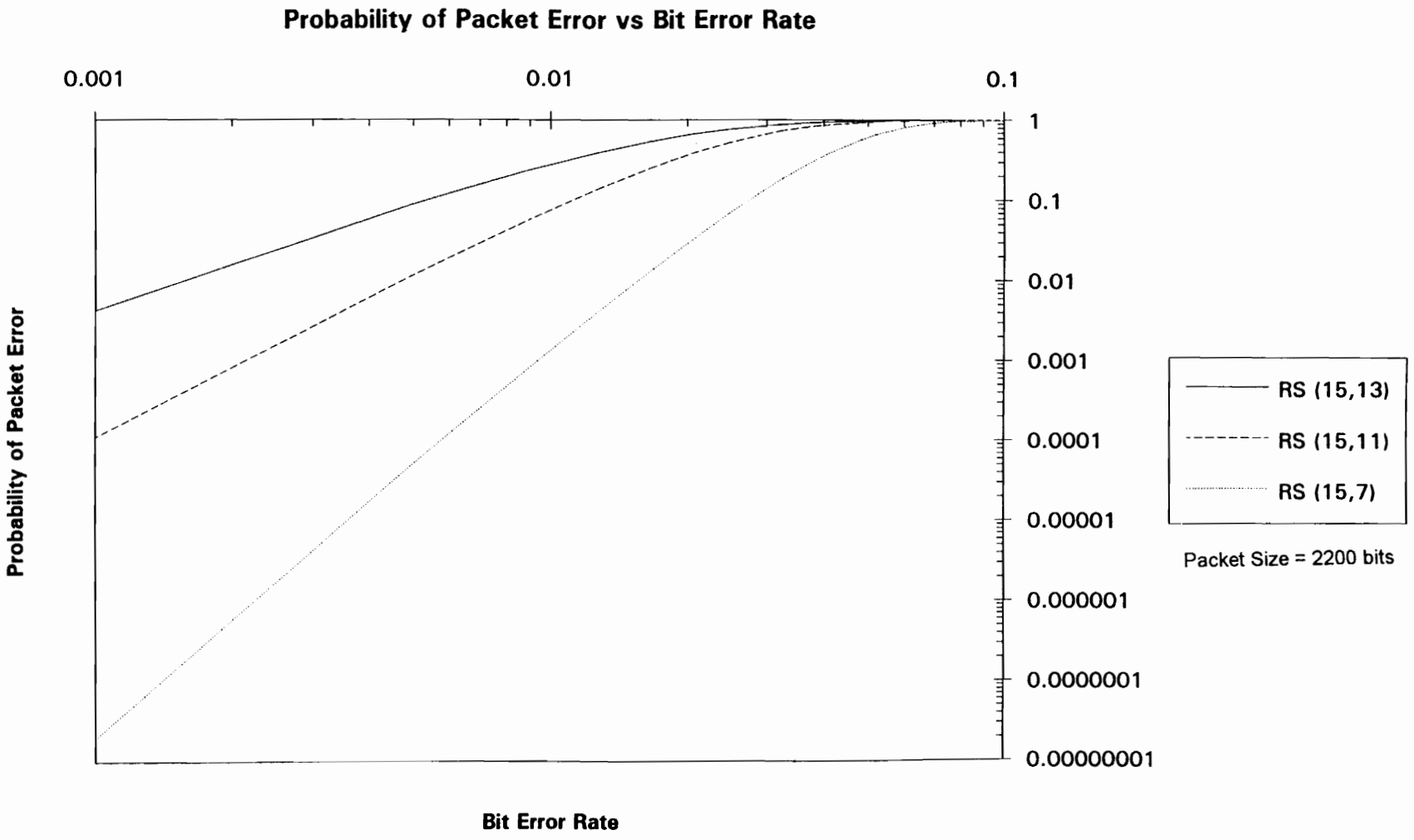

Figure 5-5 Plot of the probability of packet error vs bit error rate for various Reed Solomon codes.

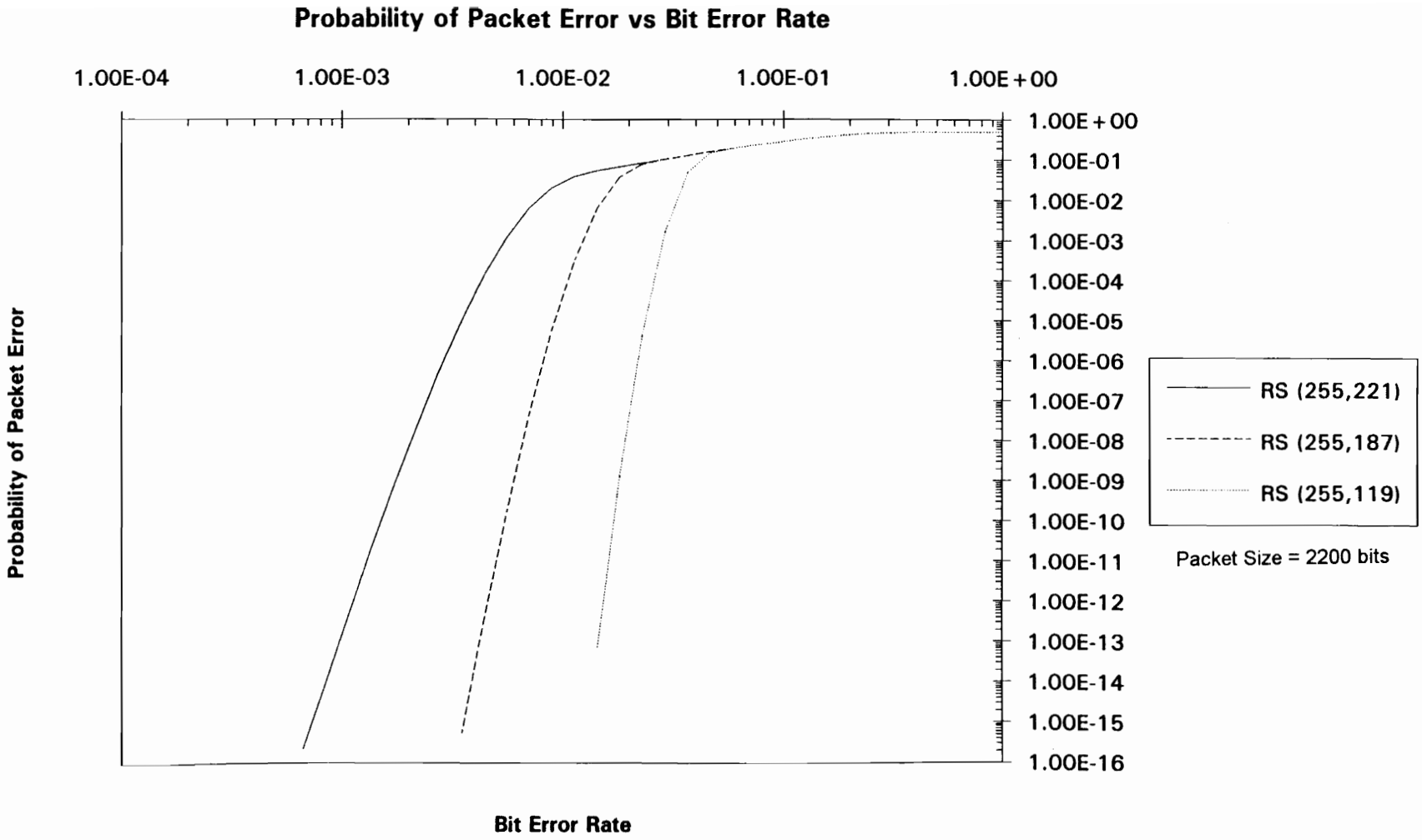

Figure 5-6 Plot of the probability of packet error vs bit error rate for various length Reed Solomon codes.

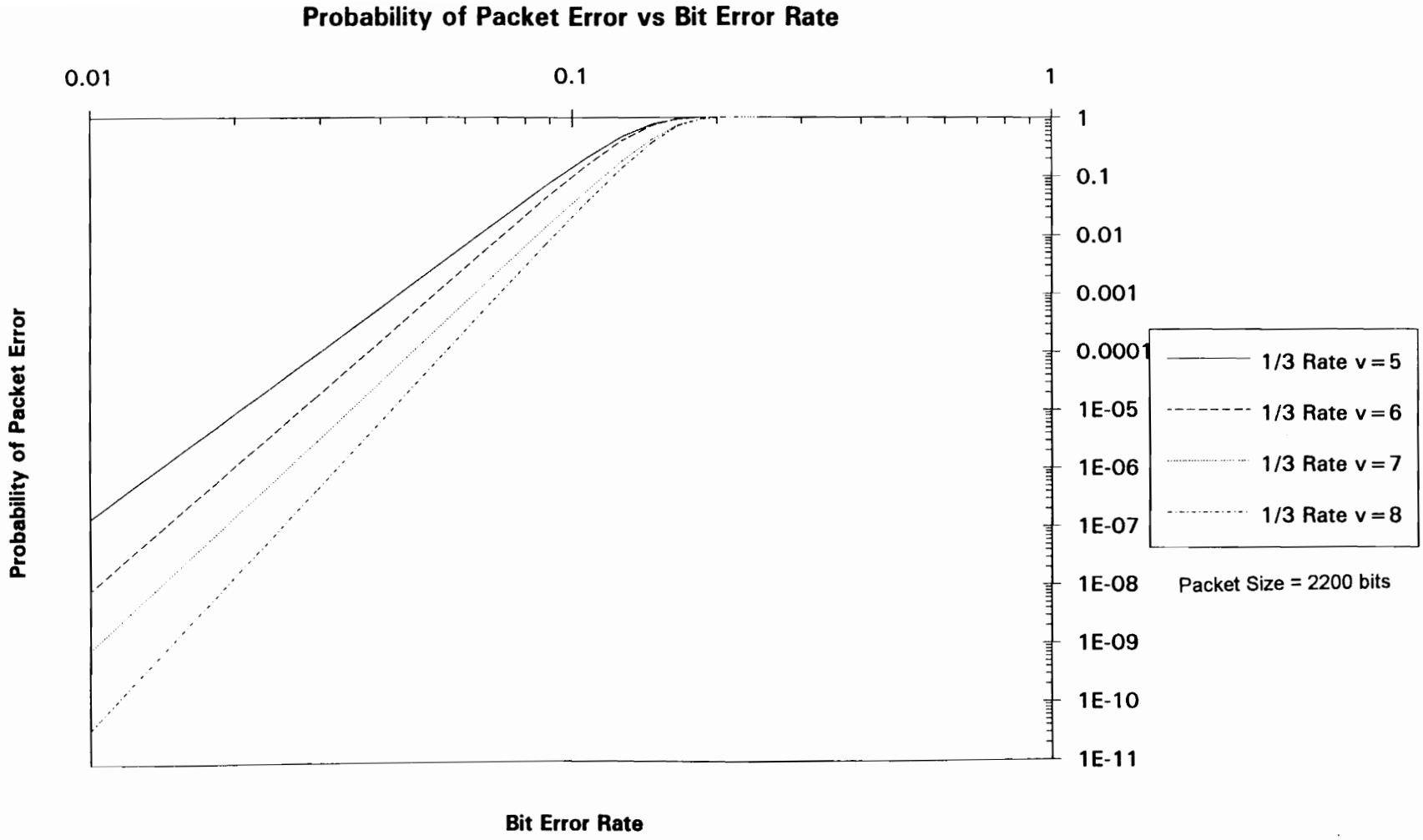

Figure 5-7 Plot of the probability of packet error vs bit error rate for 1/3 rate convolutonal codes with v=5-8.

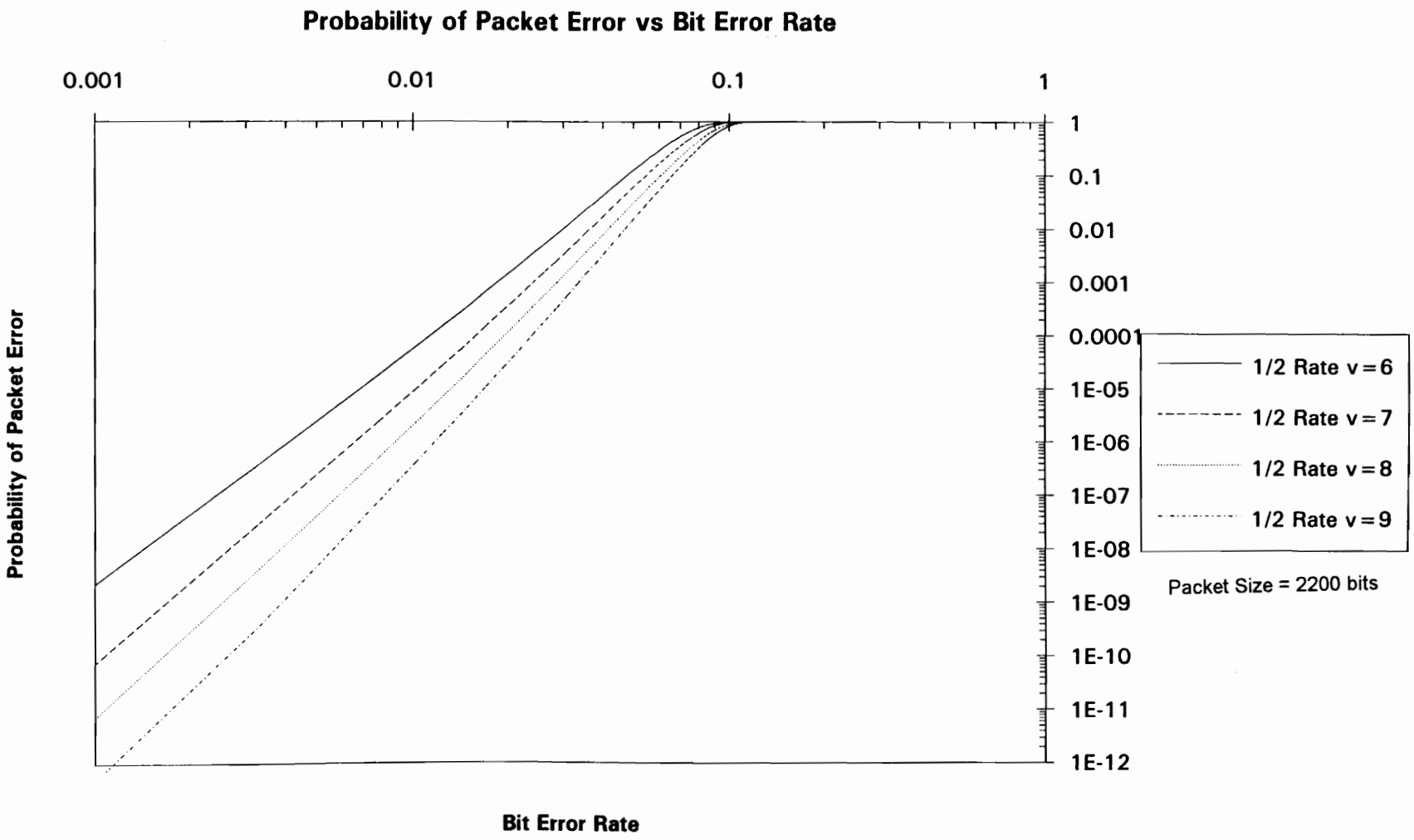

Figure 5-8 Plot of the probability of packet error vs bit error rate for 1/2 rate convolutional codes with v=6-9.

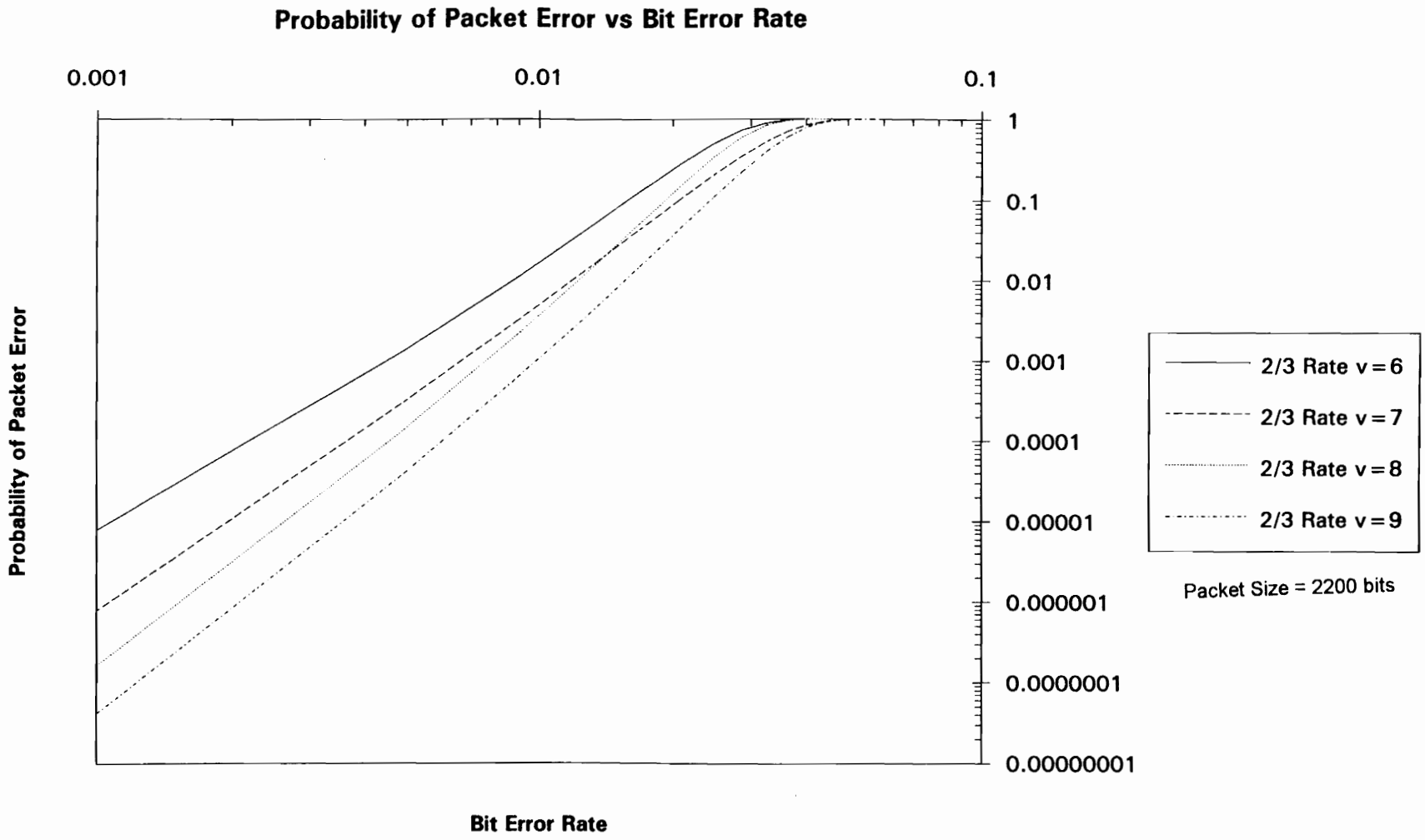

Figure 5-9 Plot of the probability of packet error vs bit error rate for 2/3 rate convolutional codes with v=6-9.

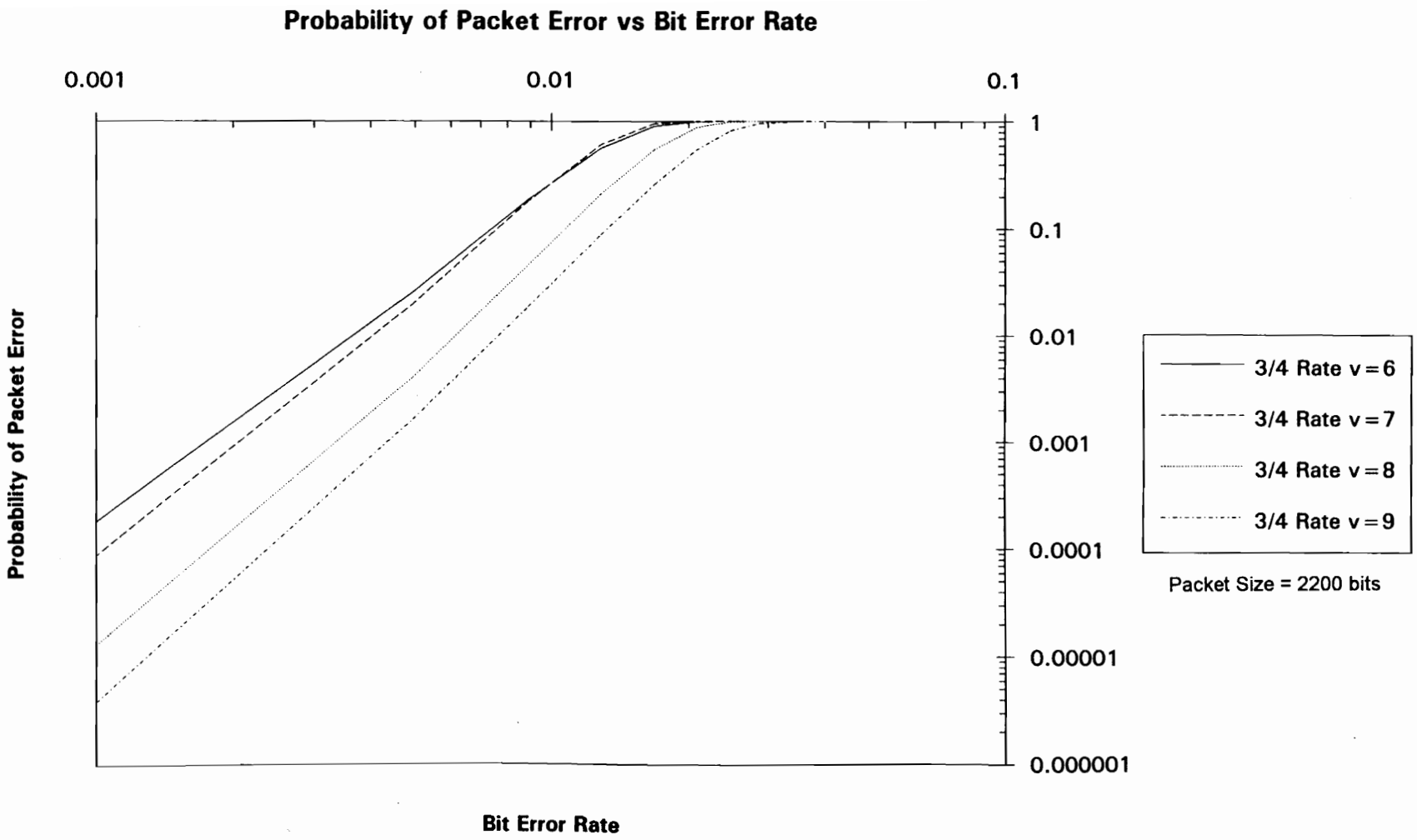

Figure 5-10 Plot of the probability of packet error vs bit error rate for 3/4 rate convoutional codes with v=6-9.

5.1.2 Throughput Efficiency

The equations to calculate the throughput efficiency of a hybrid ARQ system are the same as the ones used by the ARQ system(see Chapter 3). The only difference between the two is that the probability of packet error, which is used in the calculations, has improved for the hybrid ARQ. For equivalent bit error rates, the probability of packet error is much less for the hybrid ARQ than the ARQ as shown in Figure 5-2. When looking at the equation for the throughput efficiency of the Go-Back-N ARQ:

$$\eta_{GBN} = \frac{(1-P_p)}{(1-P_p)+P_pN}\left(\frac{k}{n}\right) \tag{5-4}$$

it can be shown that as the probability of packet error (P_p) approaches zero, η_{GBN} goes to (k/n). When P_p is equal to one, the throughput equals zero. This shows that the smaller the probability of packet error, the larger the throughput efficiency. Therefore, since the hybrid ARQ has a lower probability of packet error for the same bit error rate as the normal ARQ method, the hybrid ARQ system should demonstrate better throughput performance than when no coding is used. Figures 5-11 through 5-18 show the throughput vs. bit error rate for some error correcting codes when used with the Go-Back-N ARQ.

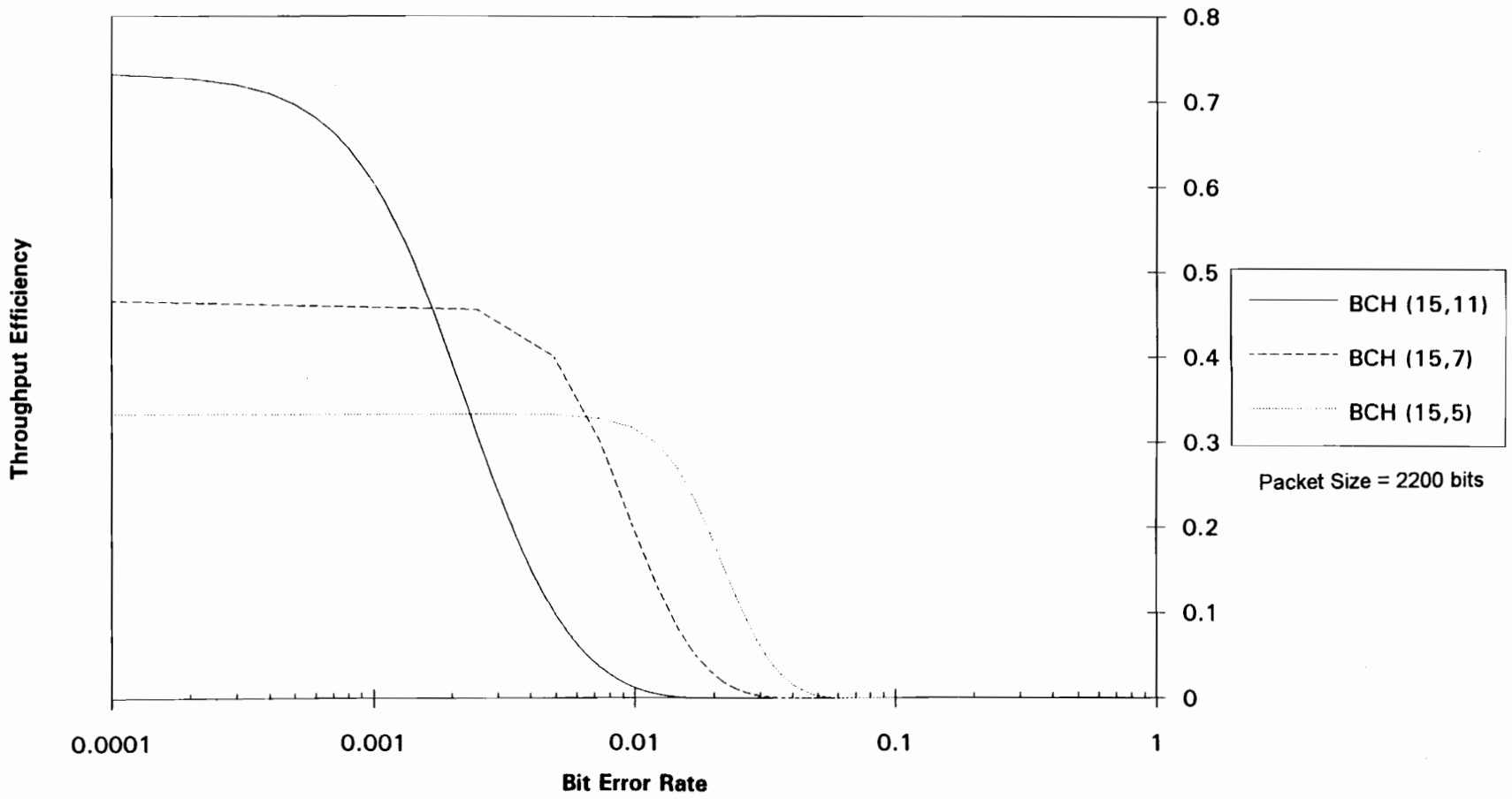

Figure 5-11 Plot of the throughput efficiency vs bit error rate for various length BCH codes. The system uses Go-Back-N ARQ.

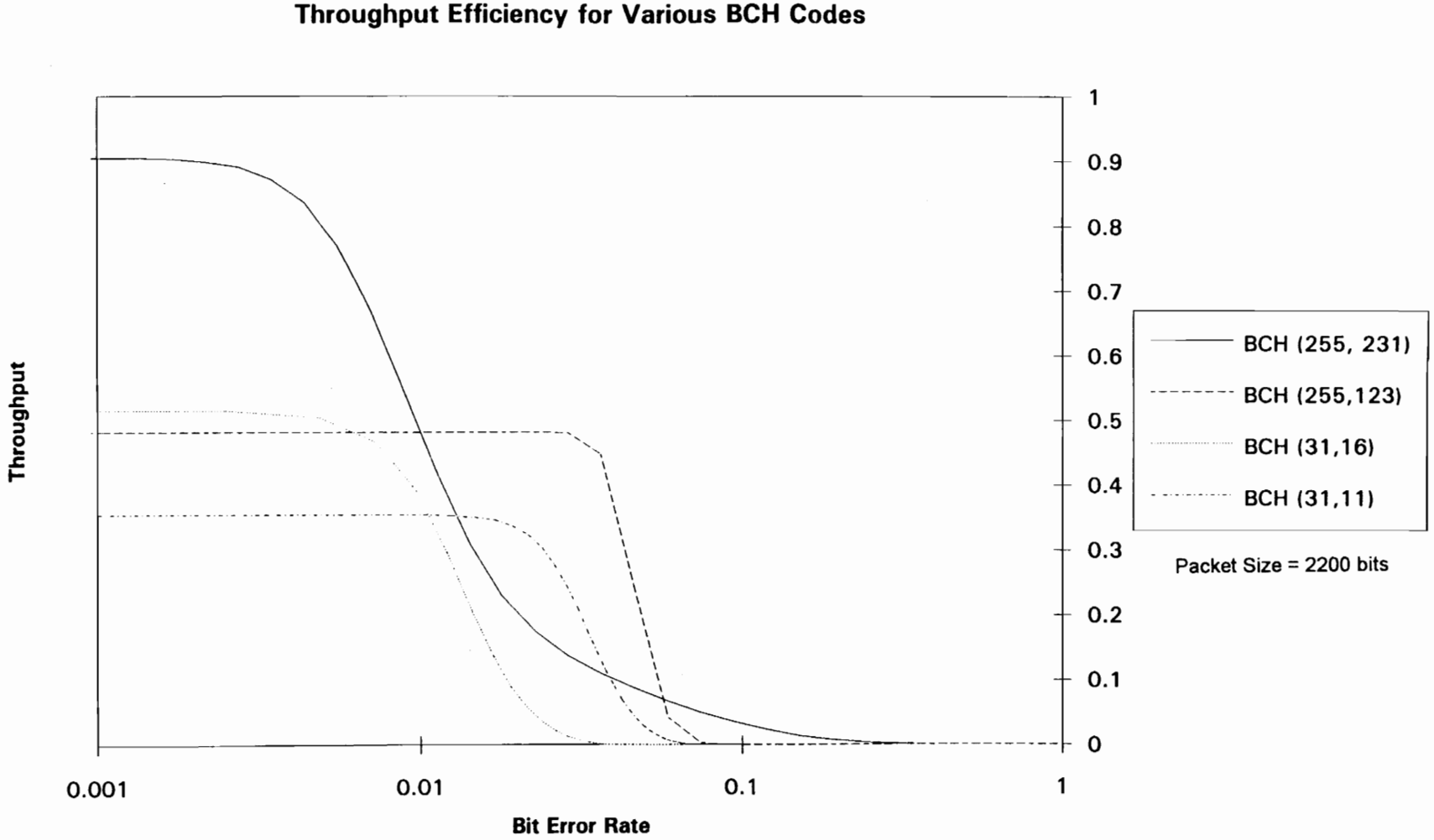

Figure 5-12 Plot of the throughput efficiency vs bit error rate for various length BCH codes. The system uses Go-Back-N ARQ.

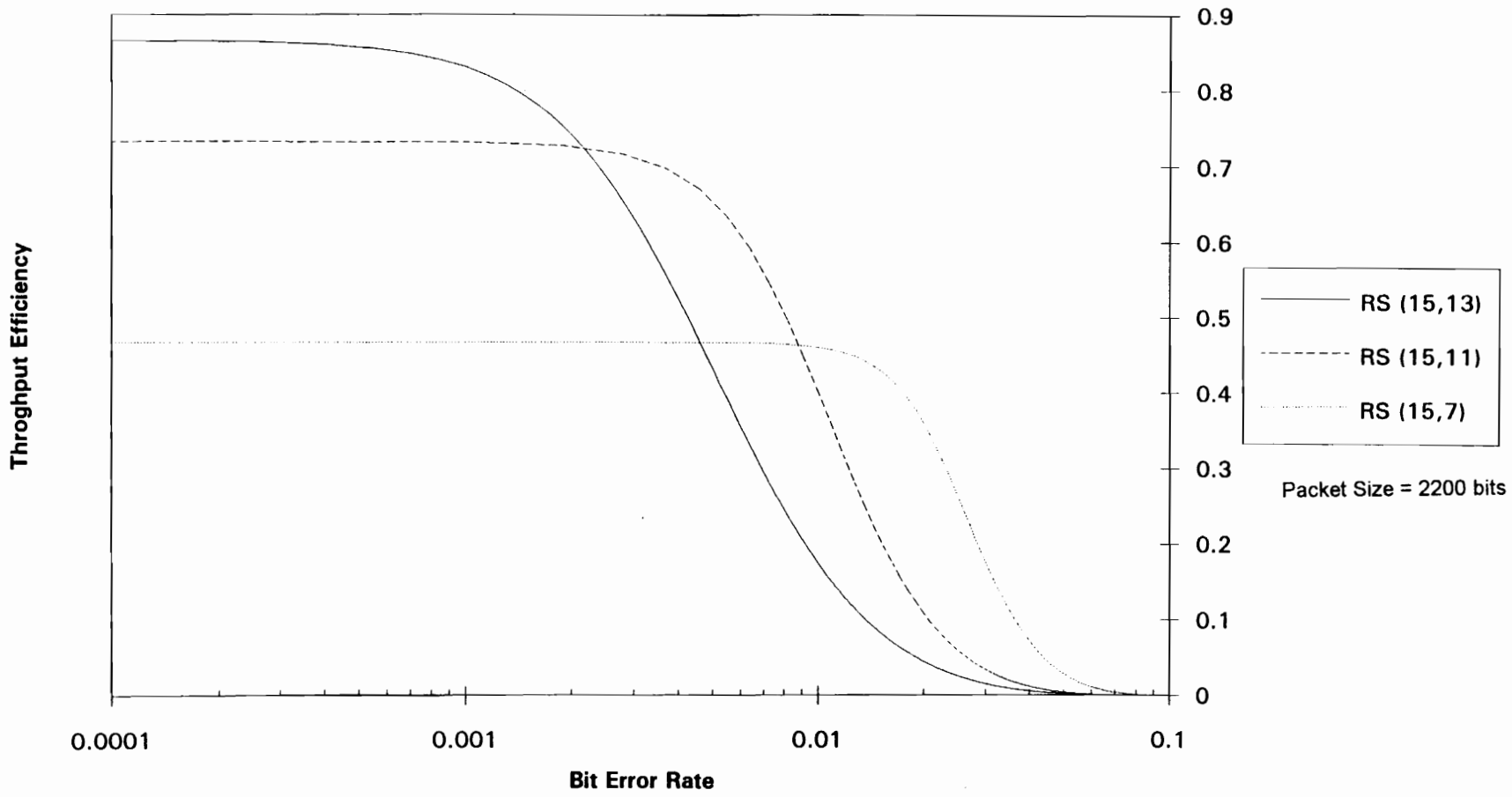

Figure 5-13 Plot of the throughput efficiency vs bit error rate for various length Reed Solomon codes. The system uses Go-Back-N ARQ.

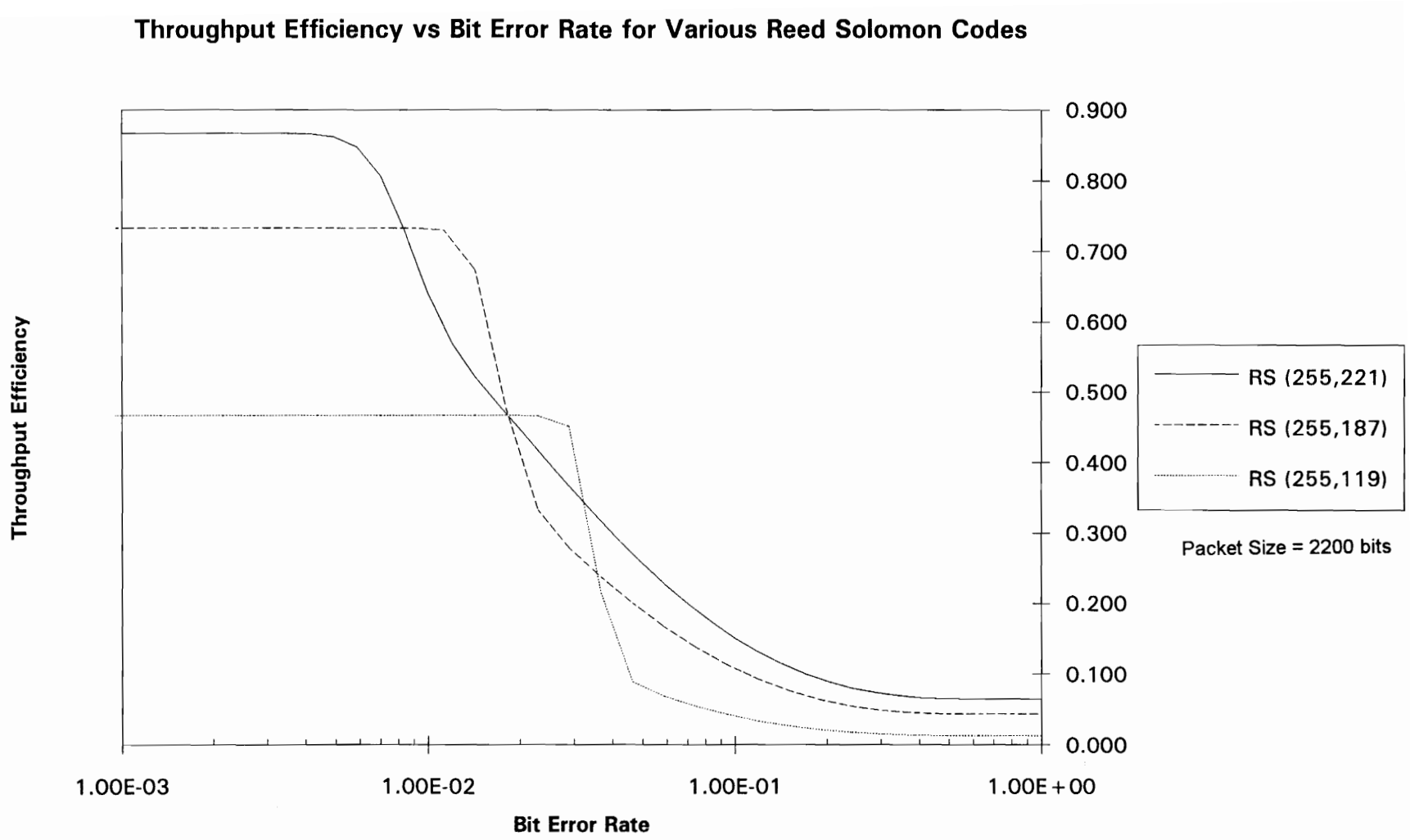

Figure 5-14 Plot of the throughput efficiency vs bit error rate for various length Reed Solomon codes. The system uses Go-Baack-N ARQ.

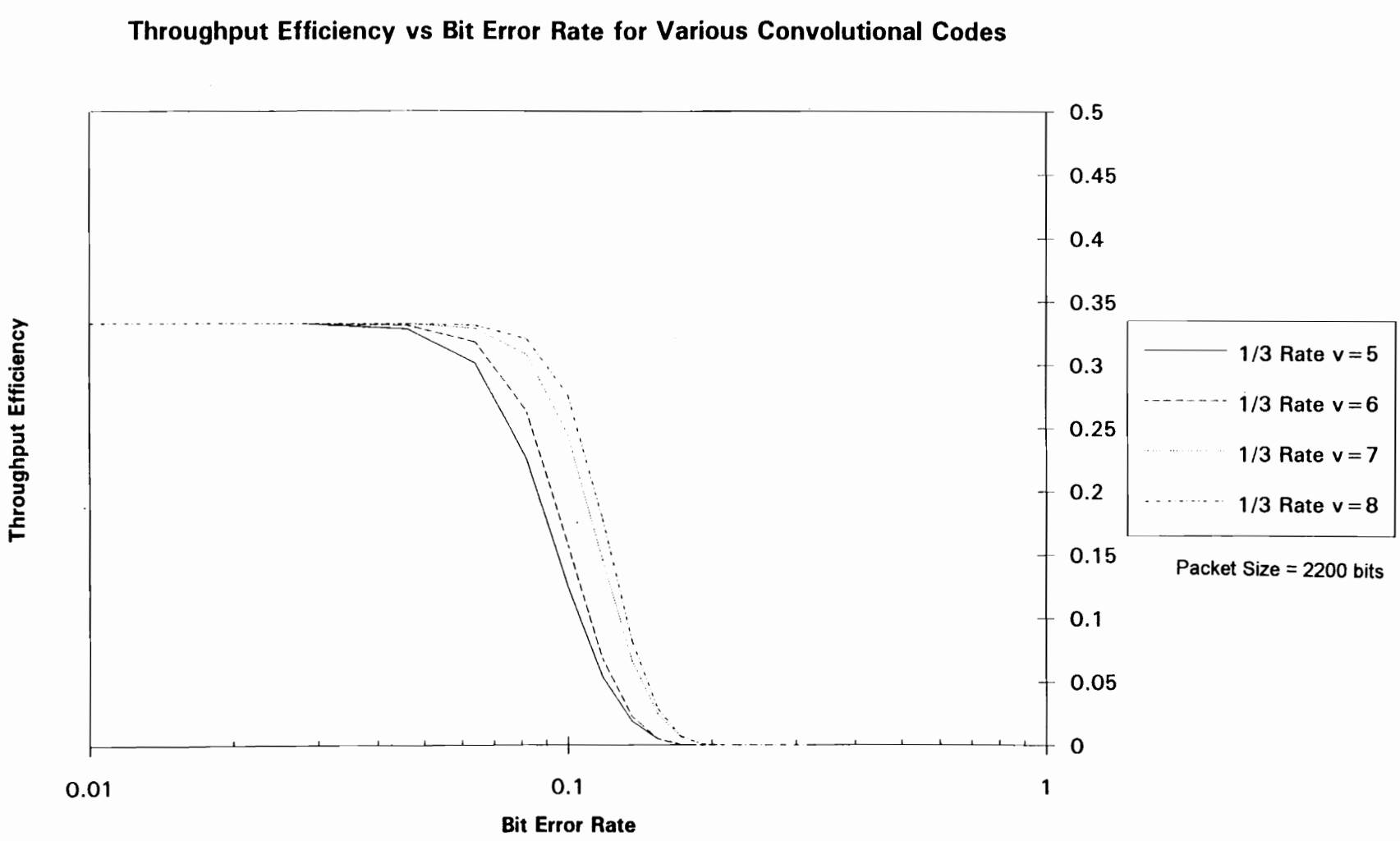

Figure 5-15 Plot of the throughput efficiency vs bit error rate for 1/3 rate convolutional codes with v=5-8. The system uses Go-Back-N ARQ.

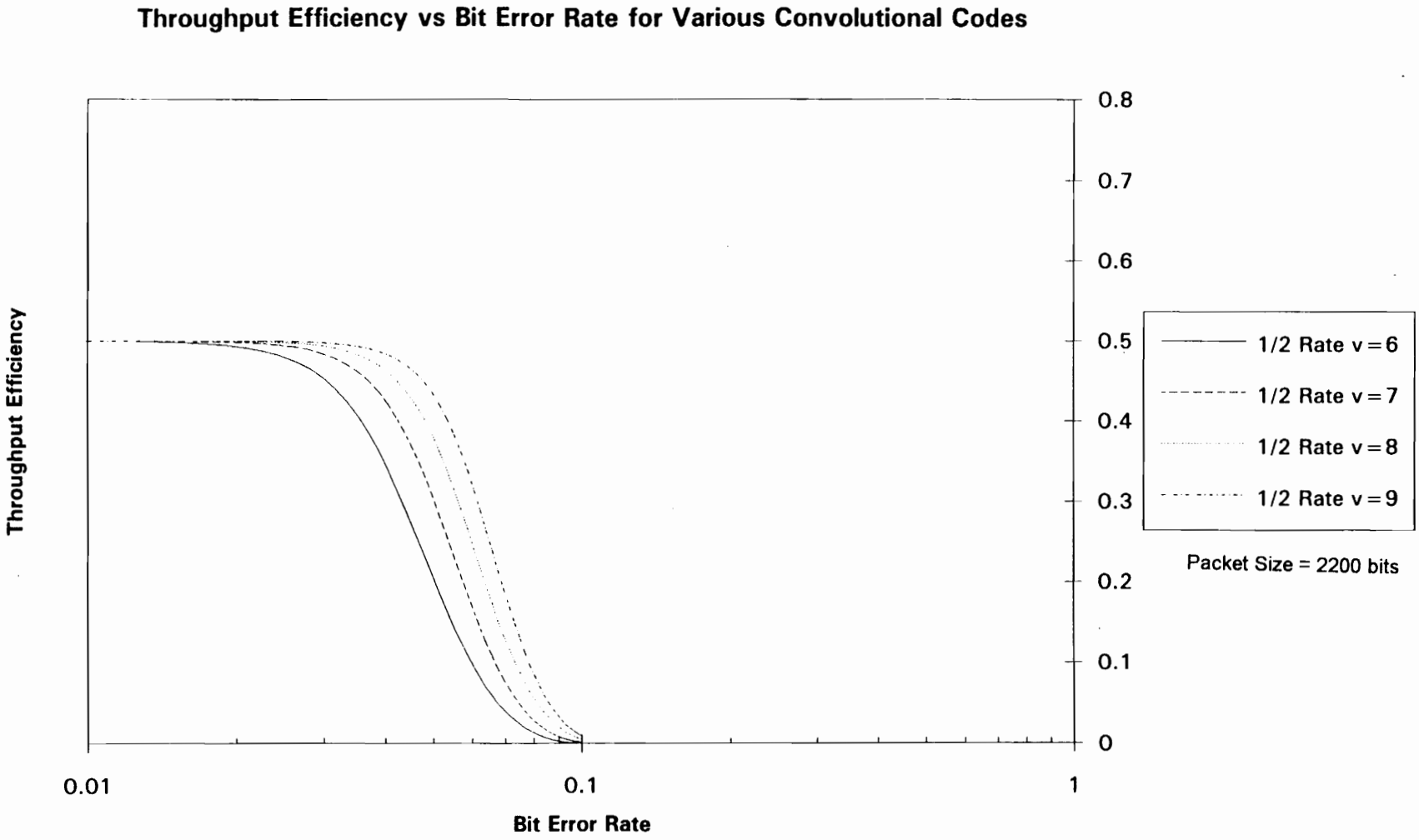

Figure 5-16 Plot of the throughput efficiency vs bit error rate for 1/2 rate convolutional codes with v=6-9. The system uses Go-Back-N ARQ.

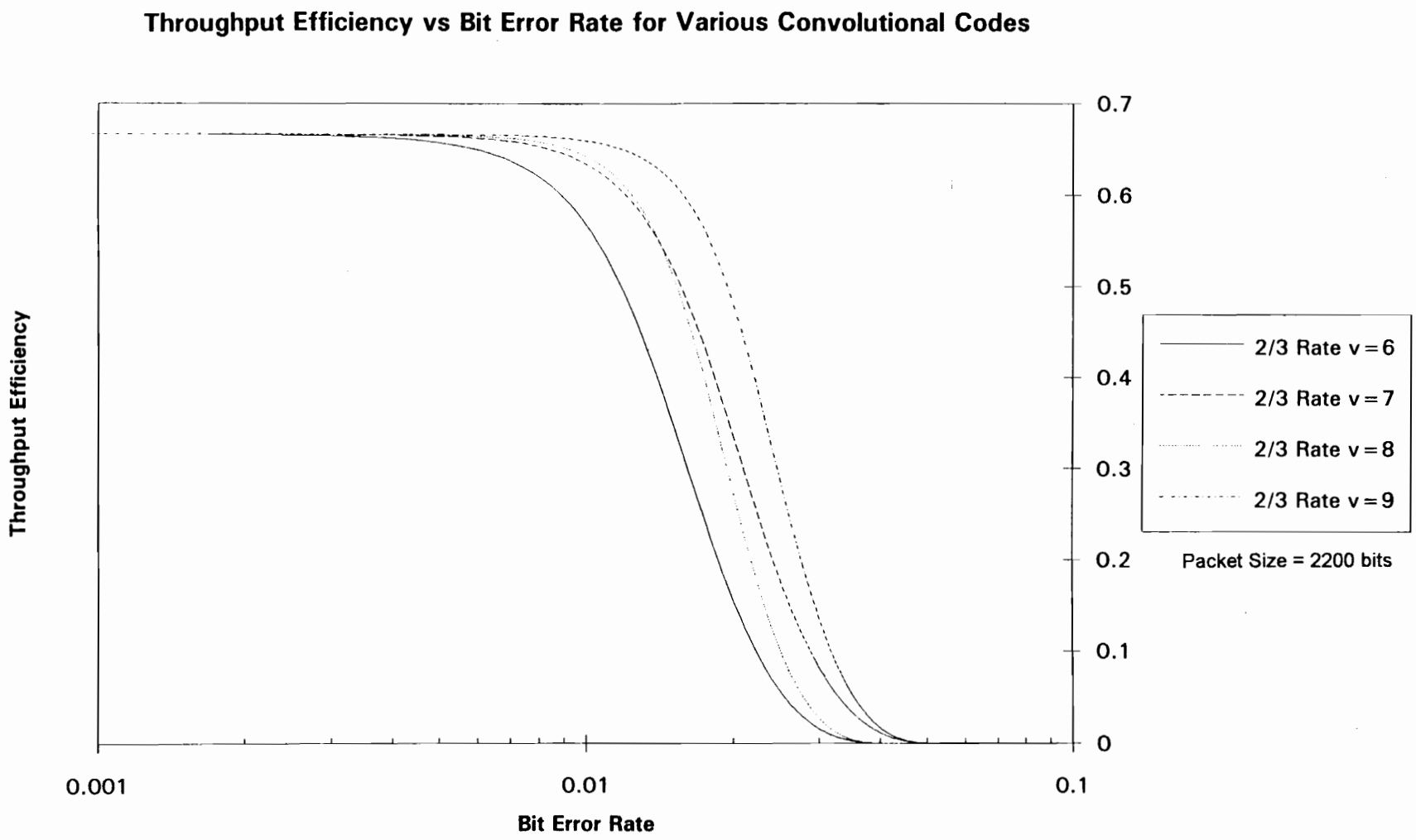

Figure 5-17 Plot of the throughput efficiency vs bit error rate for 2/3 rate convolutional codes with v=6-9. The system uses Go-Back-N ARQ.

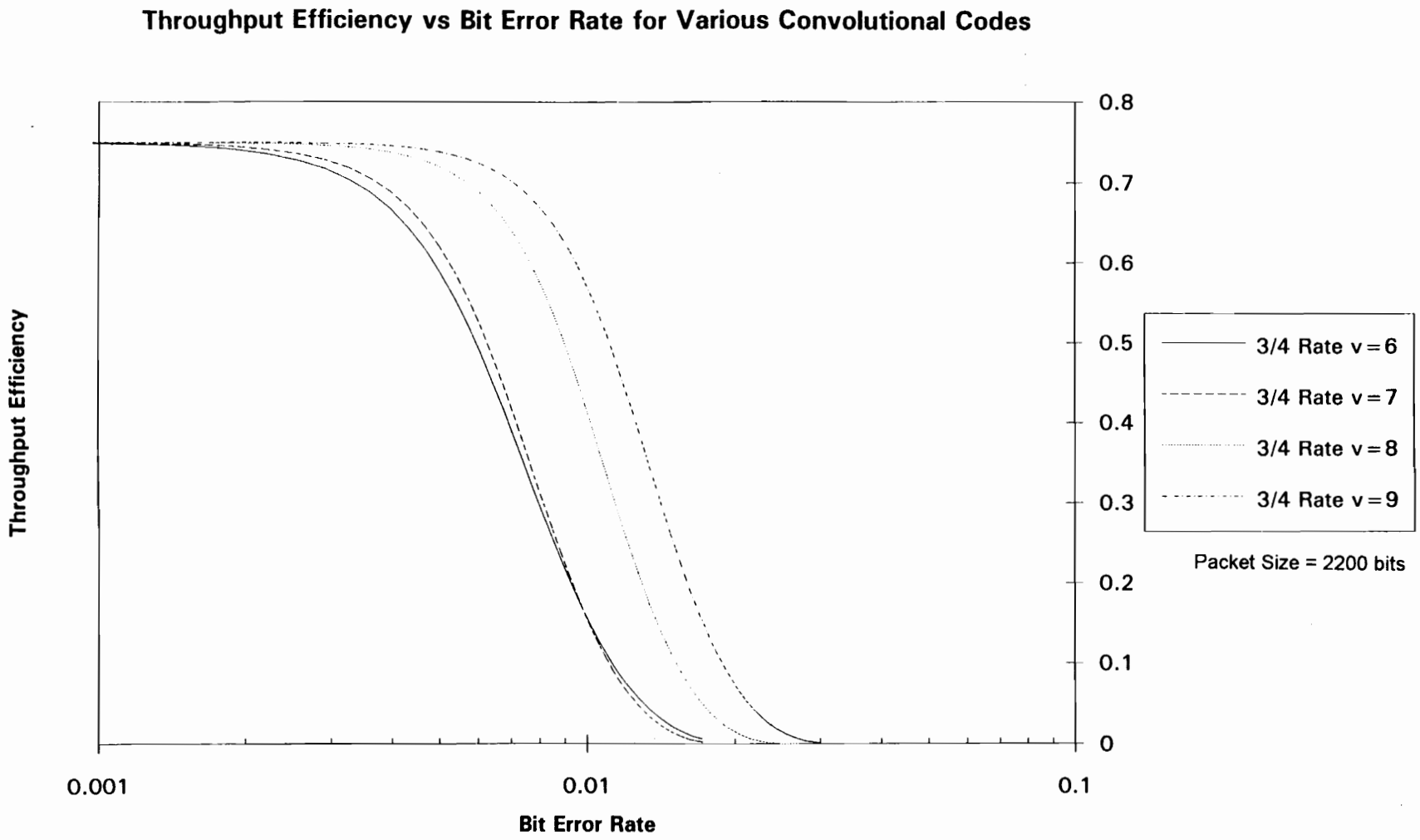

Figure 5-18 Plot of the throughput efficiency vs bit error rate for 3/4 rate convolutional codes with v=6-9. The system uses Go-Back-N ARQ.

5.2 Code Combining

When looking at the AX.25 frame, or packet, we see that it is composed of several different fields. For a better description of these fields and the AX.25 frame refer to Chapter 2. The occurrence of an error in a packet is purely random, so it is possible for an error to be located anywhere in that packet, affecting any one of the different fields. If the error occurs in a burst, it is possible that more than one field could be corrupted. The location of errors in a packet is important, because depending upon where they occur, there could be several different effects.

5.2.1 AX.25 Frame Hierarchy

With all the various fields in a given frame it is important to understand what happens to that frame when an individual field is altered by an error during transmission. The first field that occurs in a frame is the flag which is an 8 bit unique word that does not appear anywhere else in the frame. The flag indicates the beginning and/or end of frame so if an error occurs in this field, the receiver will not recognize that a new packet has arrived and it will not properly process that packet.

The second field to appear in the frame is the address field. This is an important field because it contains the address of the source, destination, and any possible digipeaters. When an error occurs in this field, the packet may not reach its intended destination. If the error affects the call sign of the source then the packet will reach its destination, but the receiver will not know the correct

identity of the transmitter so it cannot ask for any re-transmissions or send any new information to the source. If the error affects the call sign of any repeaters then the packet will not be routed correctly and will be lost before reaching its destination. Finally if the call sign of the destination is affected, the packet will never reach its intended target.

The third field occurring in the frame is the 8 bit control field which indicates the frame type. If errors occur in this field then it is possible for the receiving station to misinterpret the function of the received frame. For example, if the transmitted frame was a receive-not-ready (RNR) frame, and an error occurs causing it be received as a receive-ready (RR) frame, then the source station will begin transmitting information frames which the receiving station will not be ready to accept. Another example that poses a problem is if the received packet is mistaken for a disconnect frame. In this case the receiver will assume that the sender is ceasing operation, even though this is not the case.

When the frame is an information frame, the next field after the control field would be the PID which is an 8 bit field indicating the type of network layer protocol in use. If an error occurs here, the receiving end will not know the proper protocol to use when dealing with the received packet, causing the data to be unpacked or disassembled incorrectly.

The longest field which can occur in a frame is the information field and can have a maximum length of 256 bytes. This means that the probability for errors occurring in this field is greater than in any of the others. When errors

occur here, and if they are few in number then only a small amount of data will be lost at the receiver. Depending upon the information sent, this may not be that severe of a problem and might be acceptable.

After the information field comes the frame check sequence (FCS) field which is a 16 bit word used for frame error checking. It is important that errors do not occur here because this could cause the receiving station to believe that an error has occurred in the packet when in fact there may not be any other errors. The receiver will therefore ask for a re-transmission of a packet which could have been accepted.

The last field that could appear in the frame is the final flag which may be omitted. If the last flag is included, then any resulting errors in this field would cause effects similar to the ones for the opening flag.

5.2.2 Probability of Packet Error

It has been noted that some of the fields occurring in the radio packet carry more weight than others. It would therefore seem prudent to provide a higher level of protection for some fields.

The throughput is an important measure of performance, but as the number of redundant bits used for encoding the packet are increased, the throughput will begin to decrease. Therefore, a high rate code is desired in order to keep up the throughput efficiency. So perhaps if a high rate code is used in the "less important" fields, and a lower rate, more powerful code used on the "more important" fields, some balance can be met between system

throughput and the ability to provide more protection for the more important fields.

The performance of such a system can be compared to that of a series connection. This means that whenever any one of the components fails, the entire system will fail. In other words, if either code fails to correct any errors, the packet will be discarded and will have to be re-transmitted. For a series system the reliability is given by the following equation:

$$R_s = \prod_{i=1}^{k} R_i \tag{5-5}$$

where k is the number of components in the system and R is the reliability. Using this formula to find the probability of packet error gives:

$$P_{pe} = 1-\left(1-P_{c1}\right)^{q1}\left(1-P_{c2}\right)^{q2}\cdots\left(1-P_{ck}\right)^{qk} \tag{5-6}$$

In this equation, there are k different codes being used. P_{ck} is equal to the probability of code word error for the given code and q_k is the number of corresponding code words for that code.

The following graphs show the probability of packet error vs. the bit error rate when a series of codes are used to encode the packet. In each case two different codes are used for the encoding process. One code is used to encode only the information field which is 2048 bits, while another is used to encode the flag, address, control, PID, and FCS fields' which when combined give 152 bits.

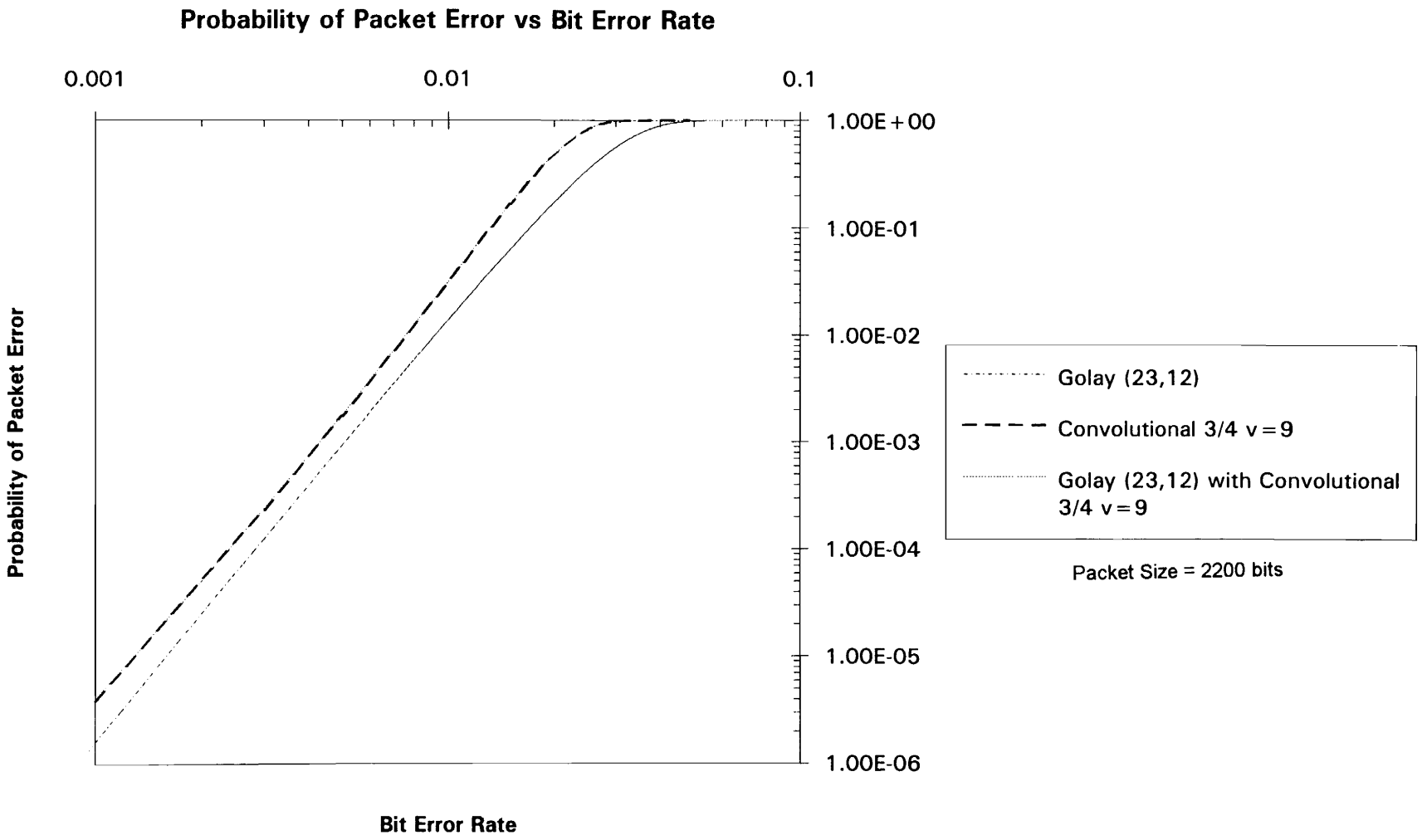

Figure 5-19 Plot of the probability of packet error vs bit error rate for a packet using both the Golay (23,12) and the convolutional 3/4 rate v=9 codes. The convolutional code is used on the information field.

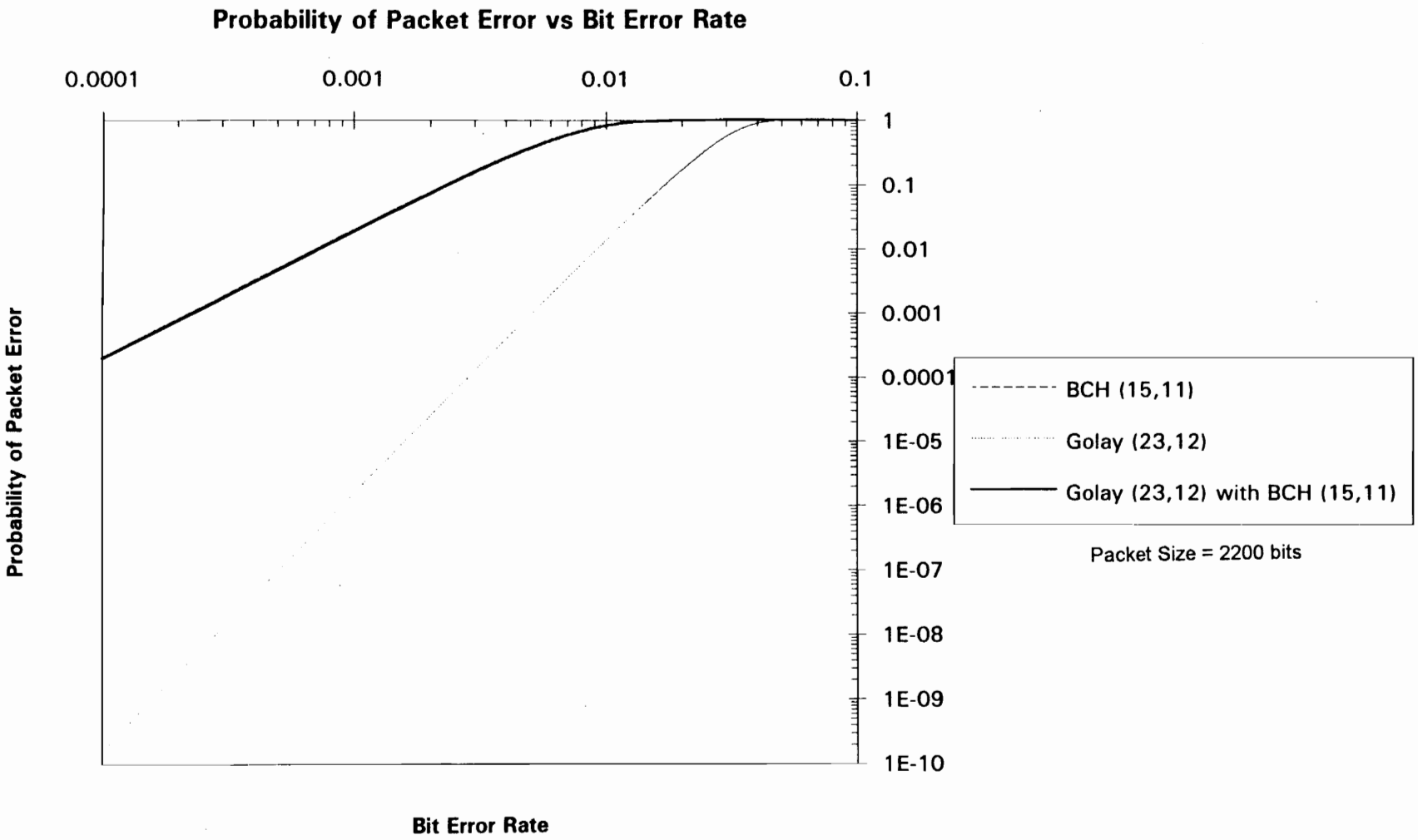

Figure 5-20 Plot of the probability of packet error vs bit error rate for a packet using both the Golay (23,12) and the BCH (15,11)codes. The BCH code is used on the information field.

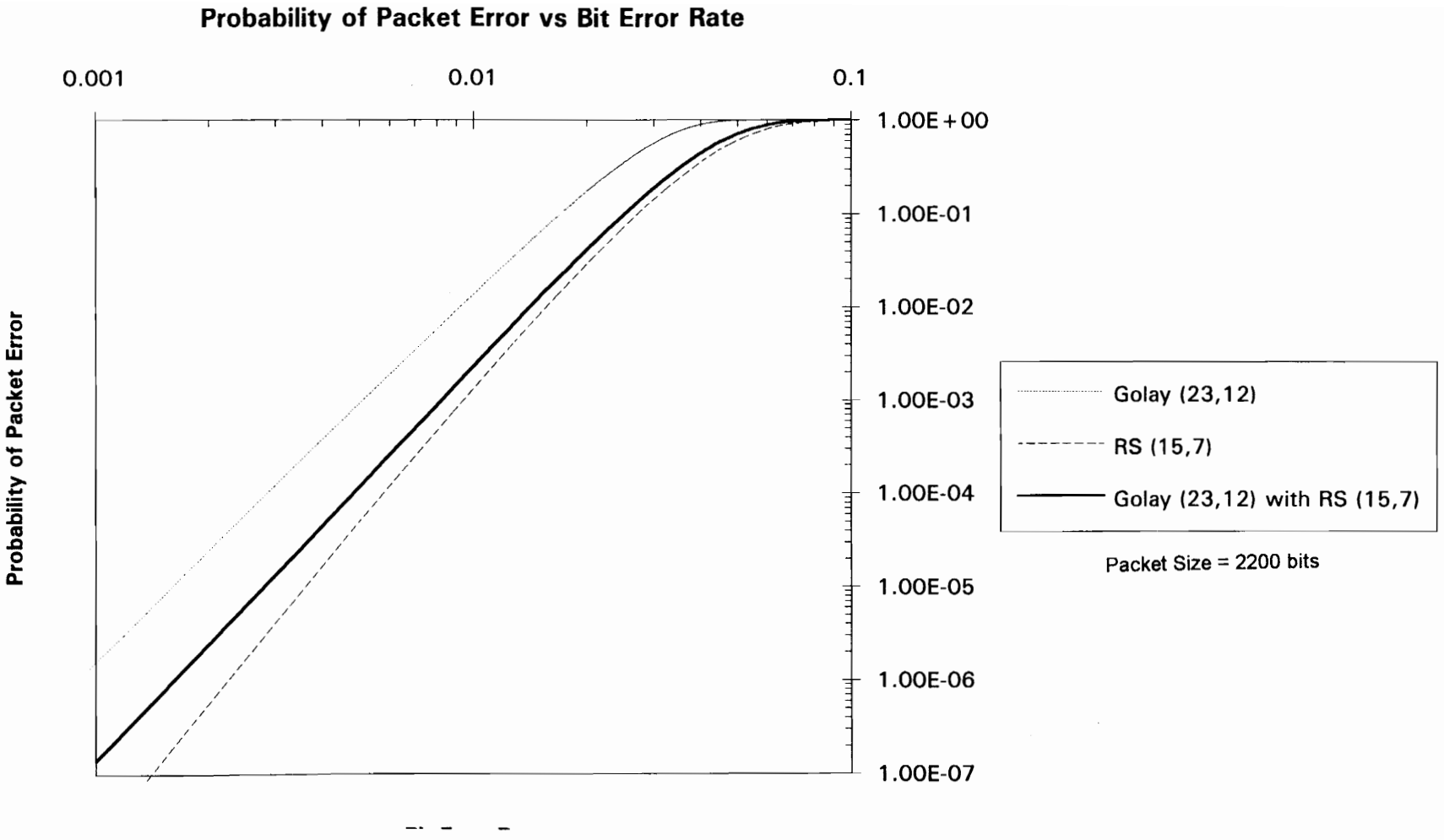

Figure 5-21 Plot of the probability of packet error vs bit error rate for a packet using both the Golay (23,12) and the RS(15,7) codes. The RS code is used on the information field.

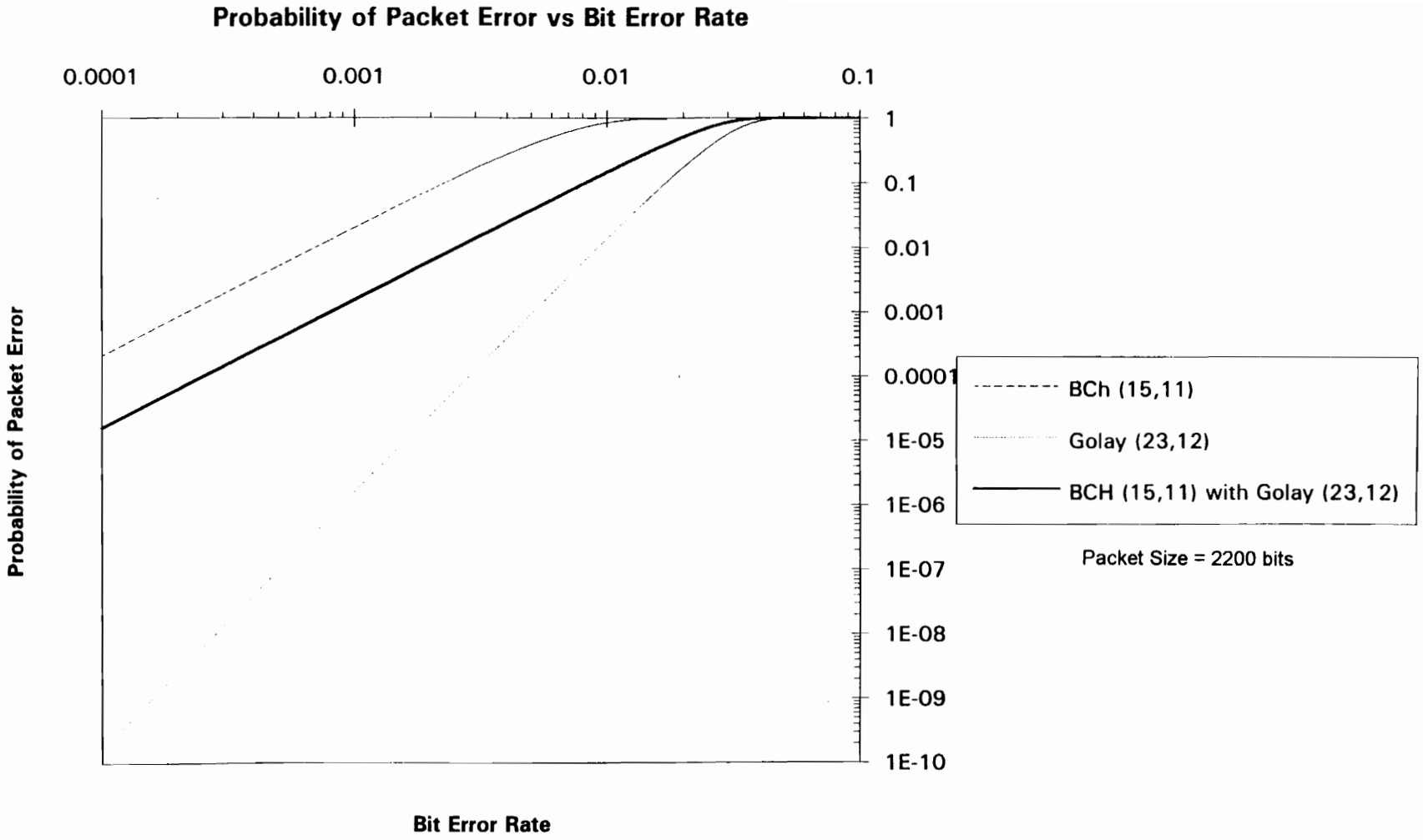

Figure 5-22 Plot of the probability of packet error vs bit error rate for a packet using both the BCH (15,11) and the Golay (23,12) codes. The Golay code is used on the information field.

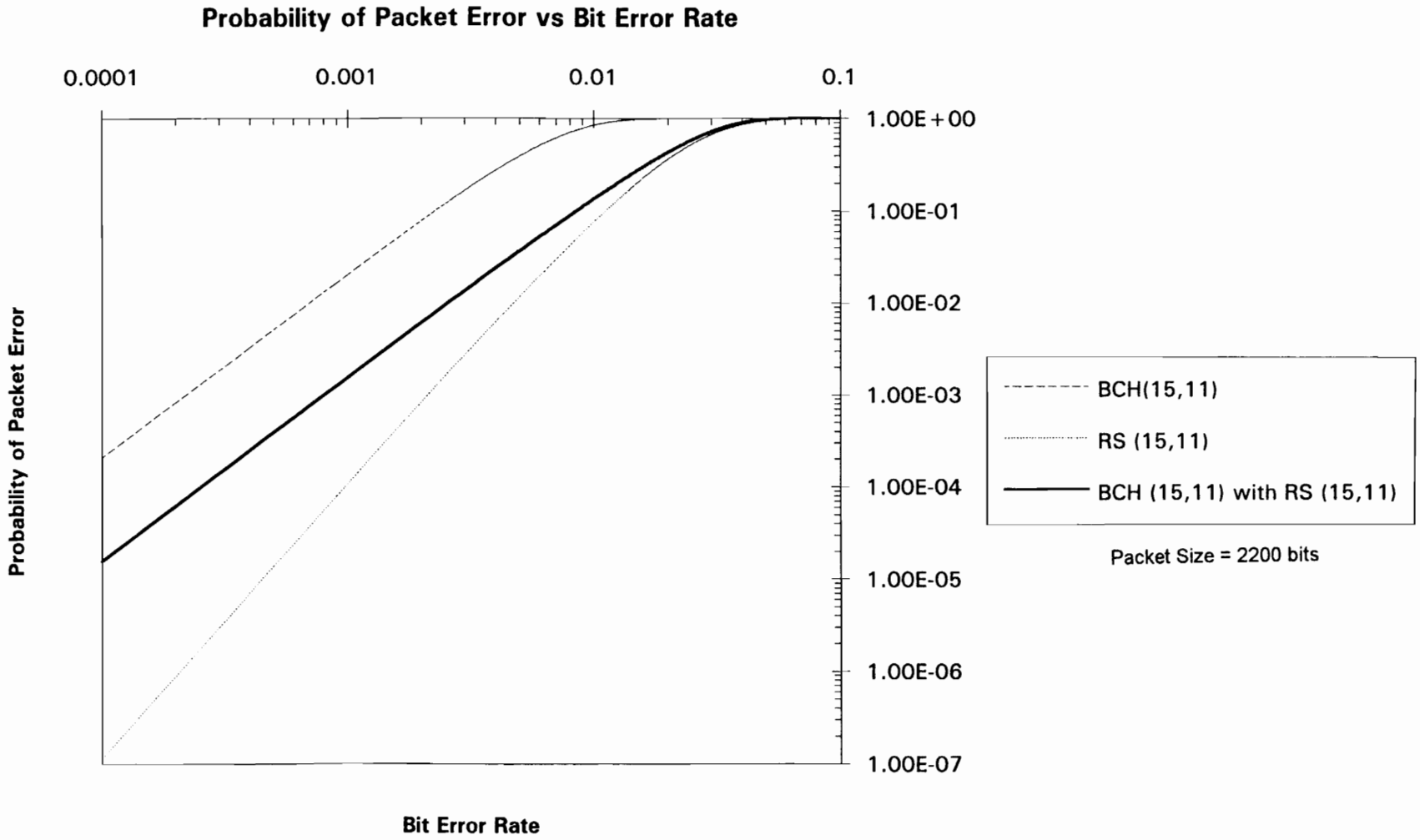

Figure 5-23 Plot of the probability of packet error vs bit error rate for a packet using both the BCH (15,11) and the RS (15,11) codes. The RS code is used on the information field.

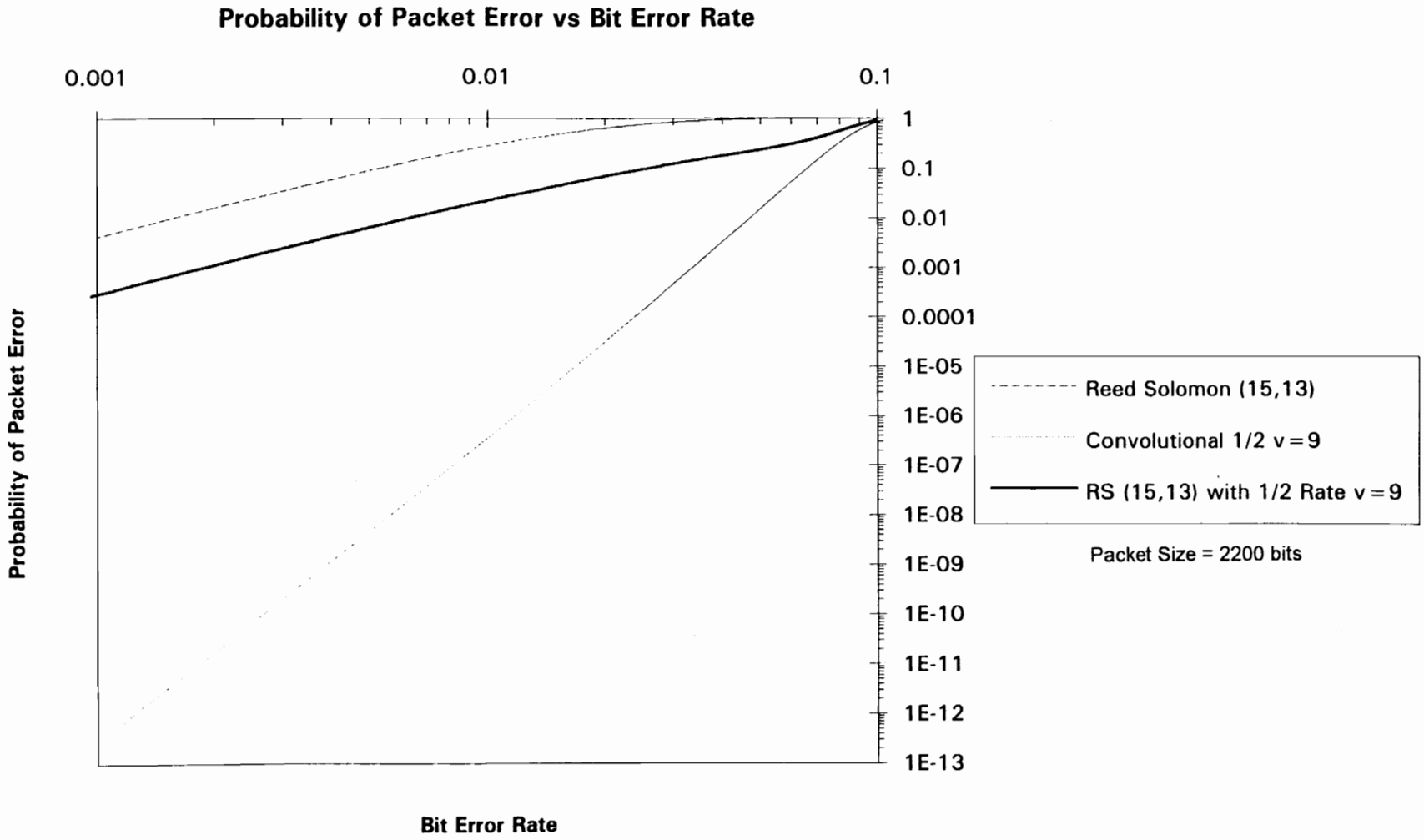

Figure 5-24 Plot of the probability of packet error vs bit error rate for a packet using both the RS (15,13) and the convolutional 1/2 rate v=9 codes. The convolutional code is used on the information field.

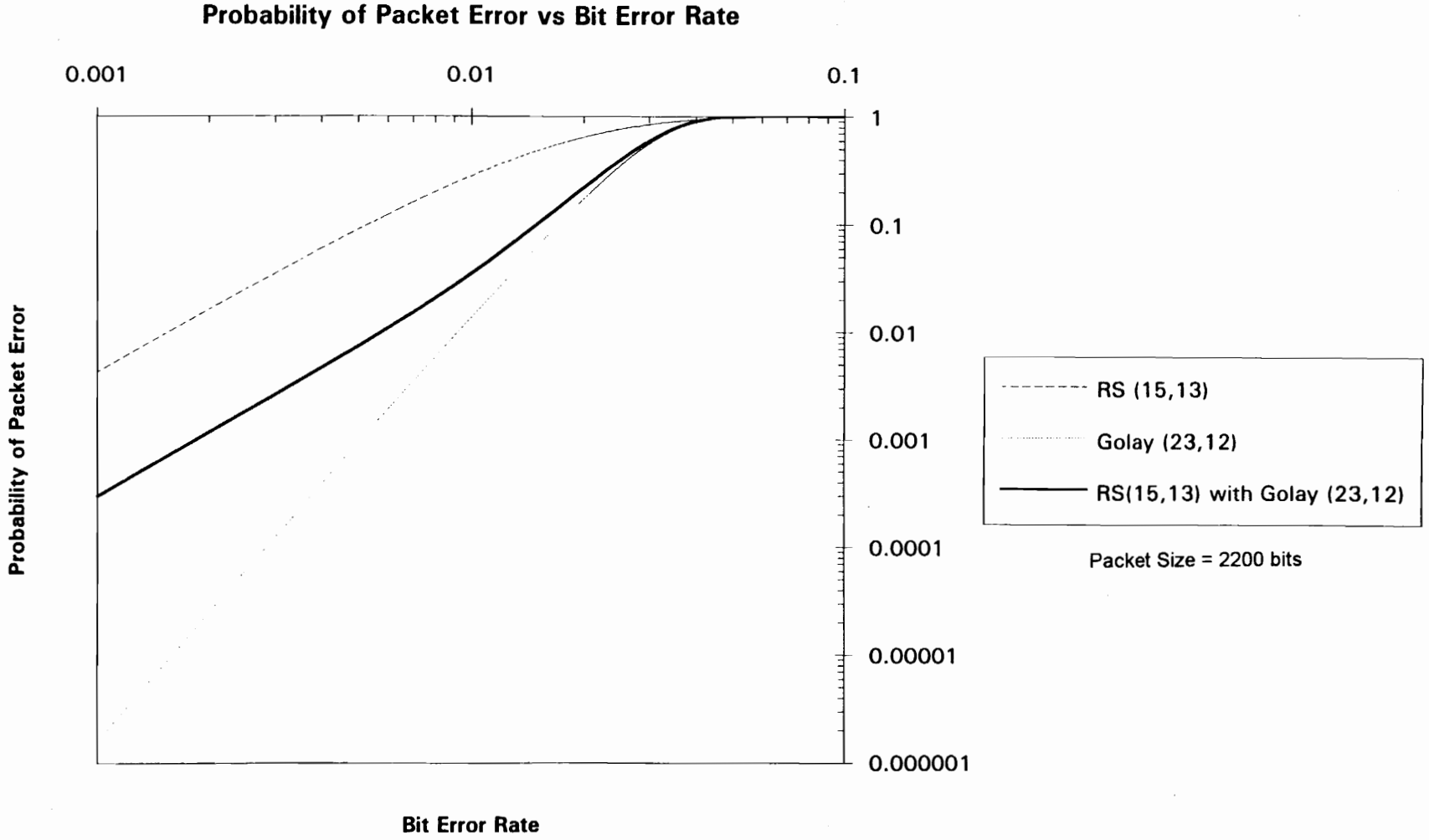

Figure 5-25 Plot of the probability of packet error vs bit error rate for a packet using both the RS (15,13) and the Golay (23,12)codes. The Golay code is used on the information field.

5.2.3 Throughput Efficiency

The throughput for the system described in section 5.2.2 will be dependent upon the type of ARQ scheme used. The equations for the throughput will be the same as those given is Chapter 3, with the probability of packet error given by equation 5-6. The following figures show the throughput for the systems of Figures 5-26 through 5-32.

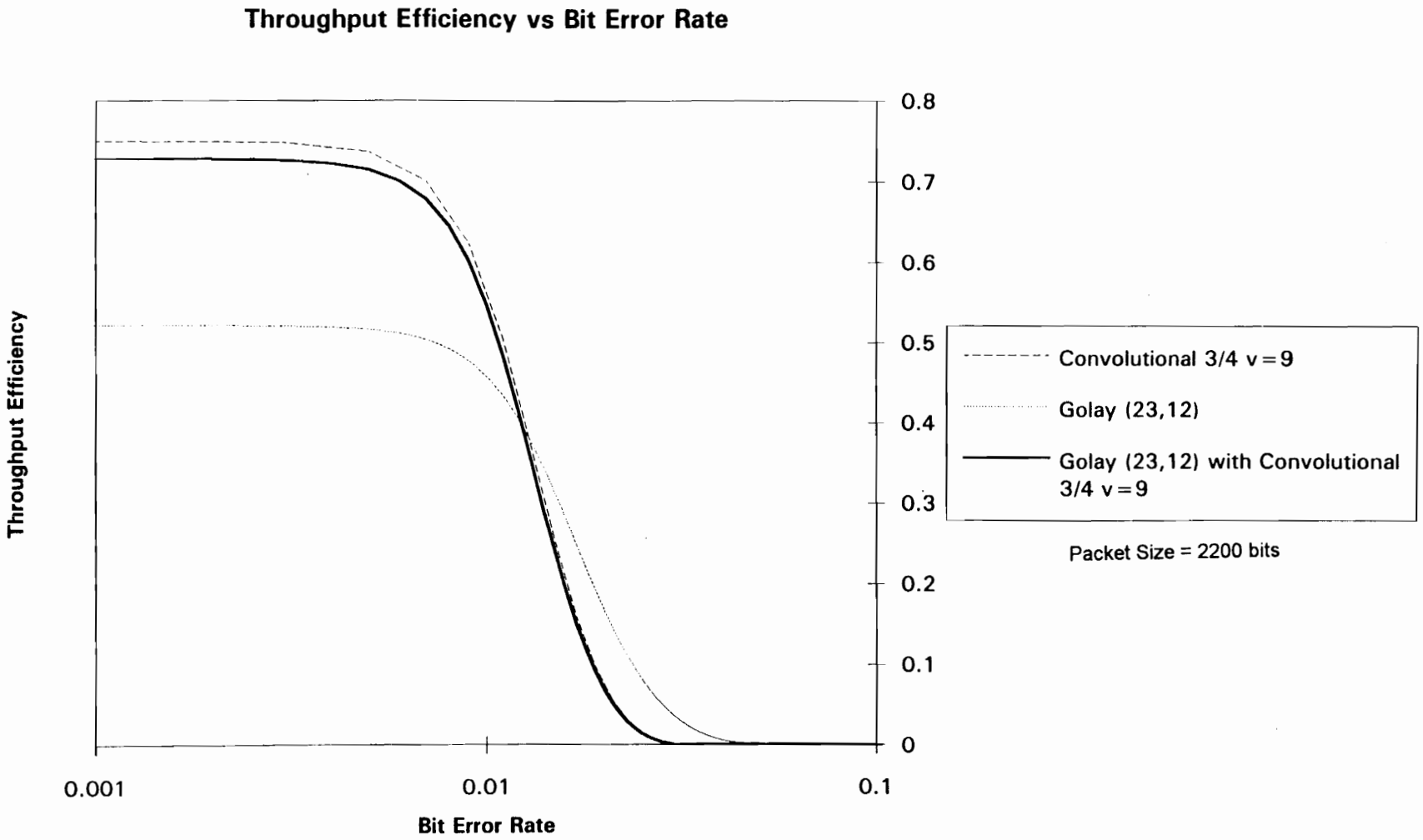

Figure 5-26 Plot of the throughput efficiency vs bit error rate for a packet which uses both the Golay (23,12) code and the Convoutional 3/4 rate v=9 codes. The system uss Go-Back-N ARQ. The convolutional code is used on the information field.

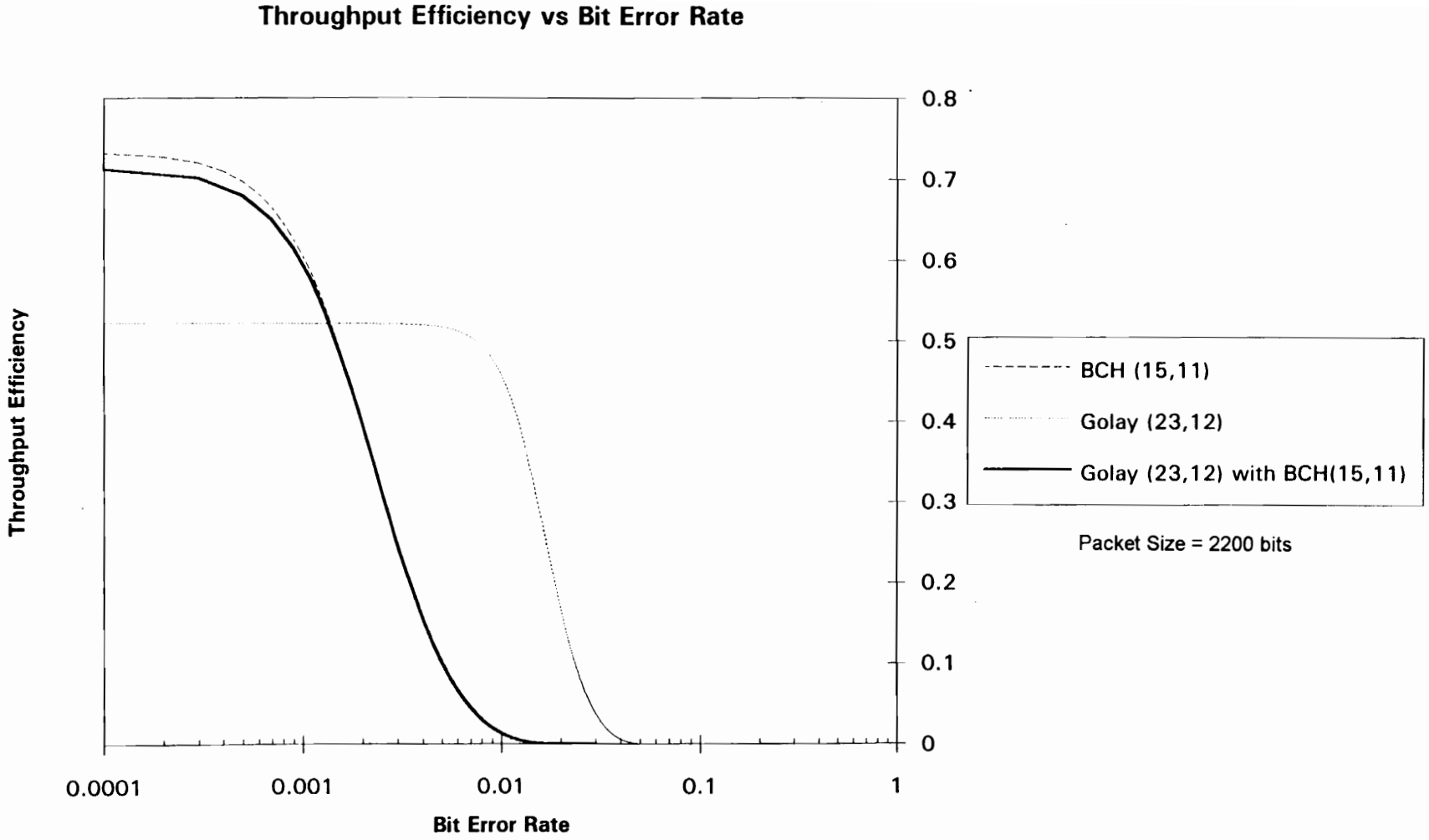

Figure 5-27 Plot of the throughput efficiency vs bit error rate for a packet which uses both the Golay (23,12) code and the BCH (15,11) codes. The system uss Go-Back-N ARQ. The BCH code is used on the information field.

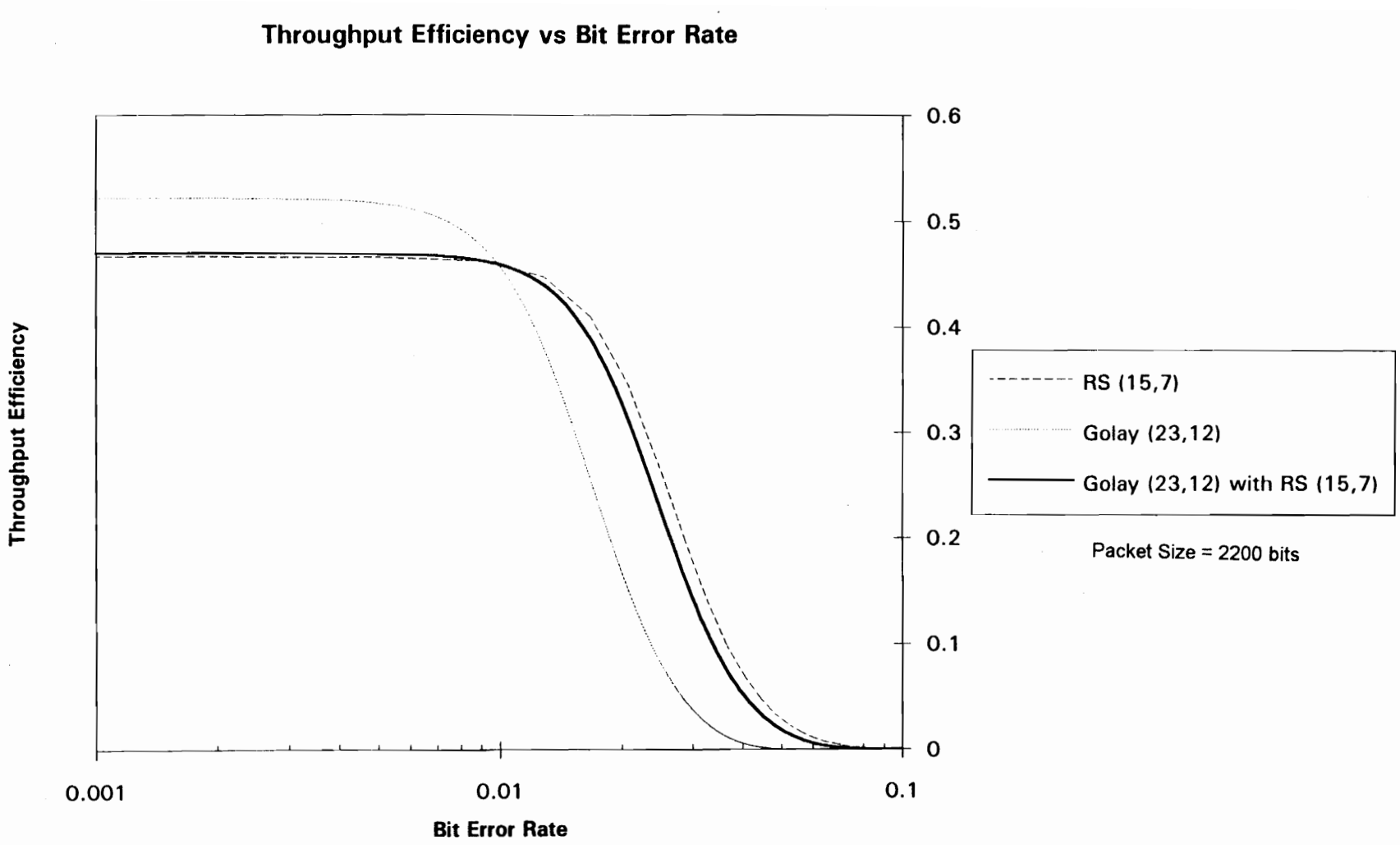

Figure 5-28 Plot of the throughput efficiency vs bit error rate for a packet which uses both the Golay (23,12) code and the RS (15,7) codes. The system uss Go-Back-N ARQ. The RS code is used on the information field.

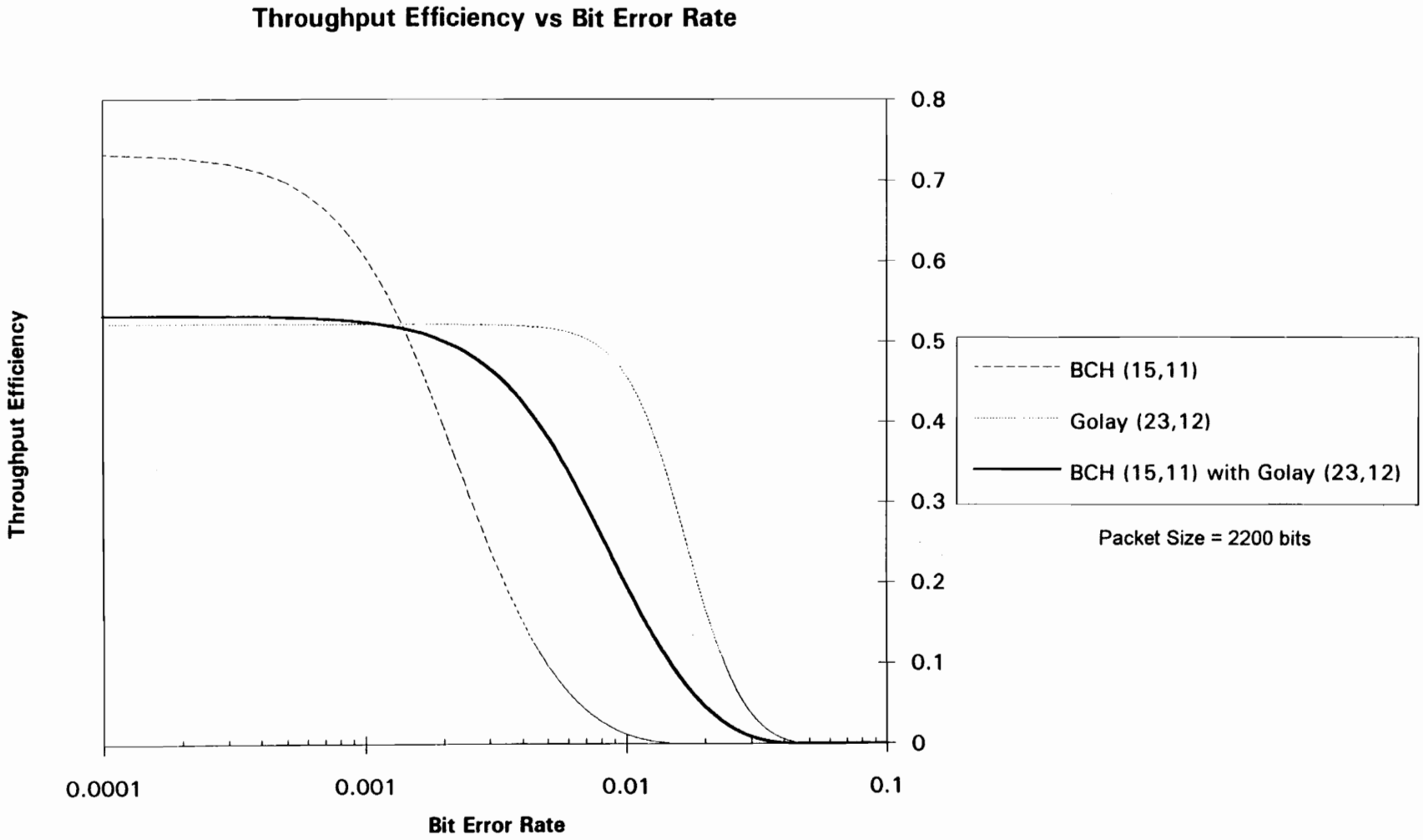

Figure 5-29 Plot of the throughput efficiency vs bit error rate for a packet which uses both the BCH (15,11) with the Golay (23,12) codes. The system uss Go-Back-N ARQ. The Golay code is used on the information field.

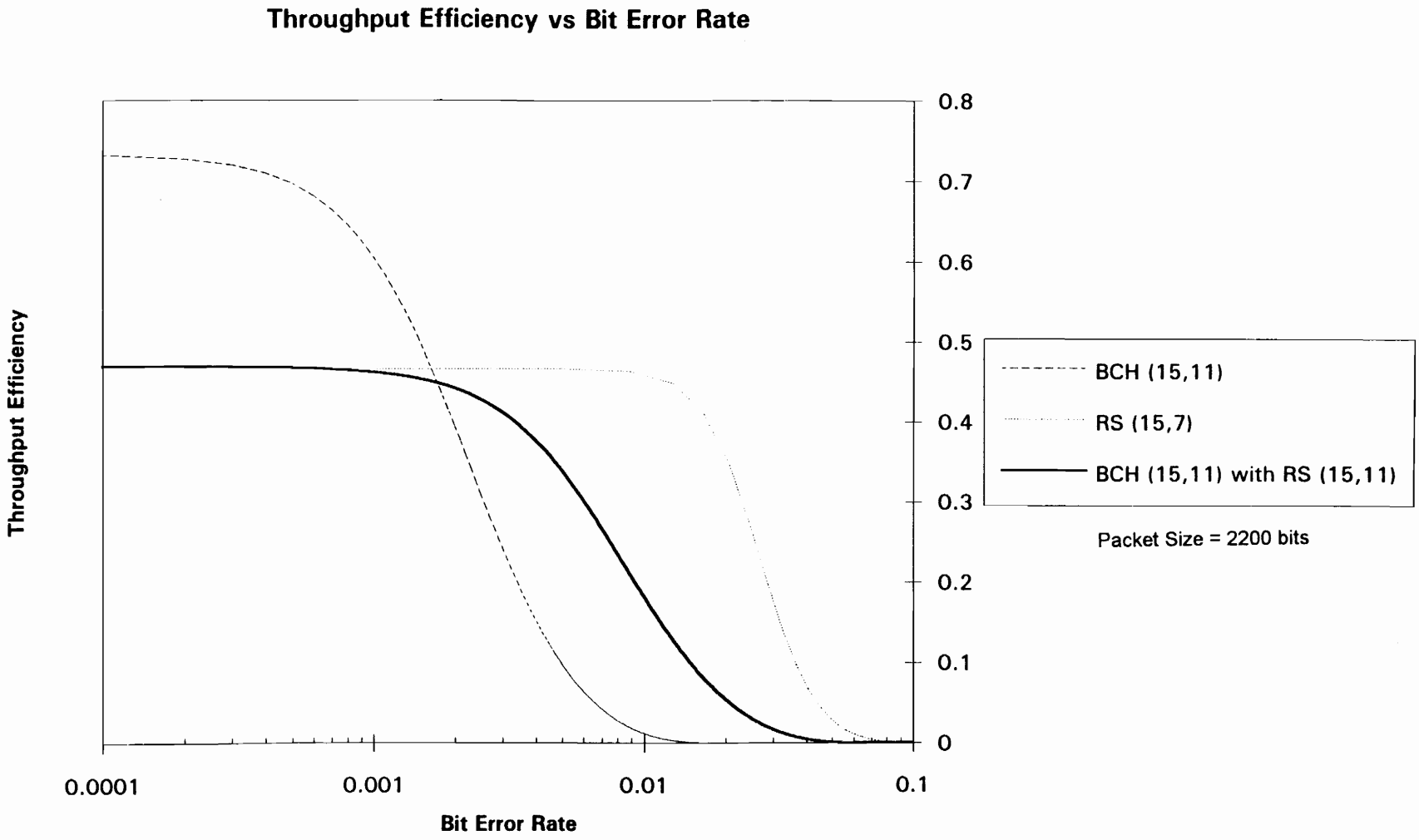

Figure 5-30 Plot of the throughput efficiency vs bit error rate for a packet which uses both the BCH (15,11) and the RS (15,11) codes. The system uss Go-Back-N ARQ. The RS code is used on the information field.

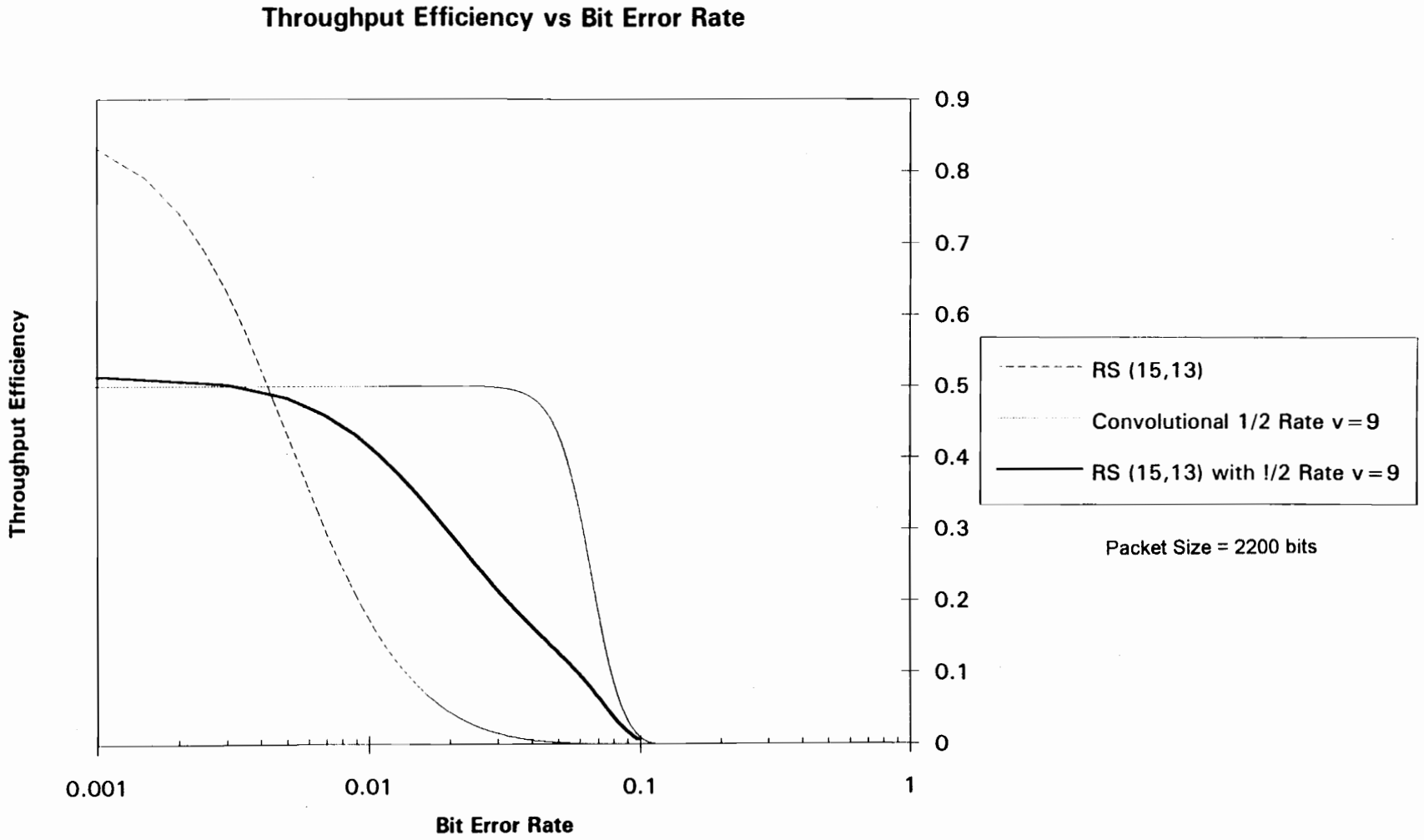

Figure 5-31 Plot of the throughput efficiency vs bit error rate for a packet which uses both the RS (15,13) code and the Convoutional 1/2 rate v=9 code. The system uses Go-Back-N ARQ. The convlutional code is used on the information field.

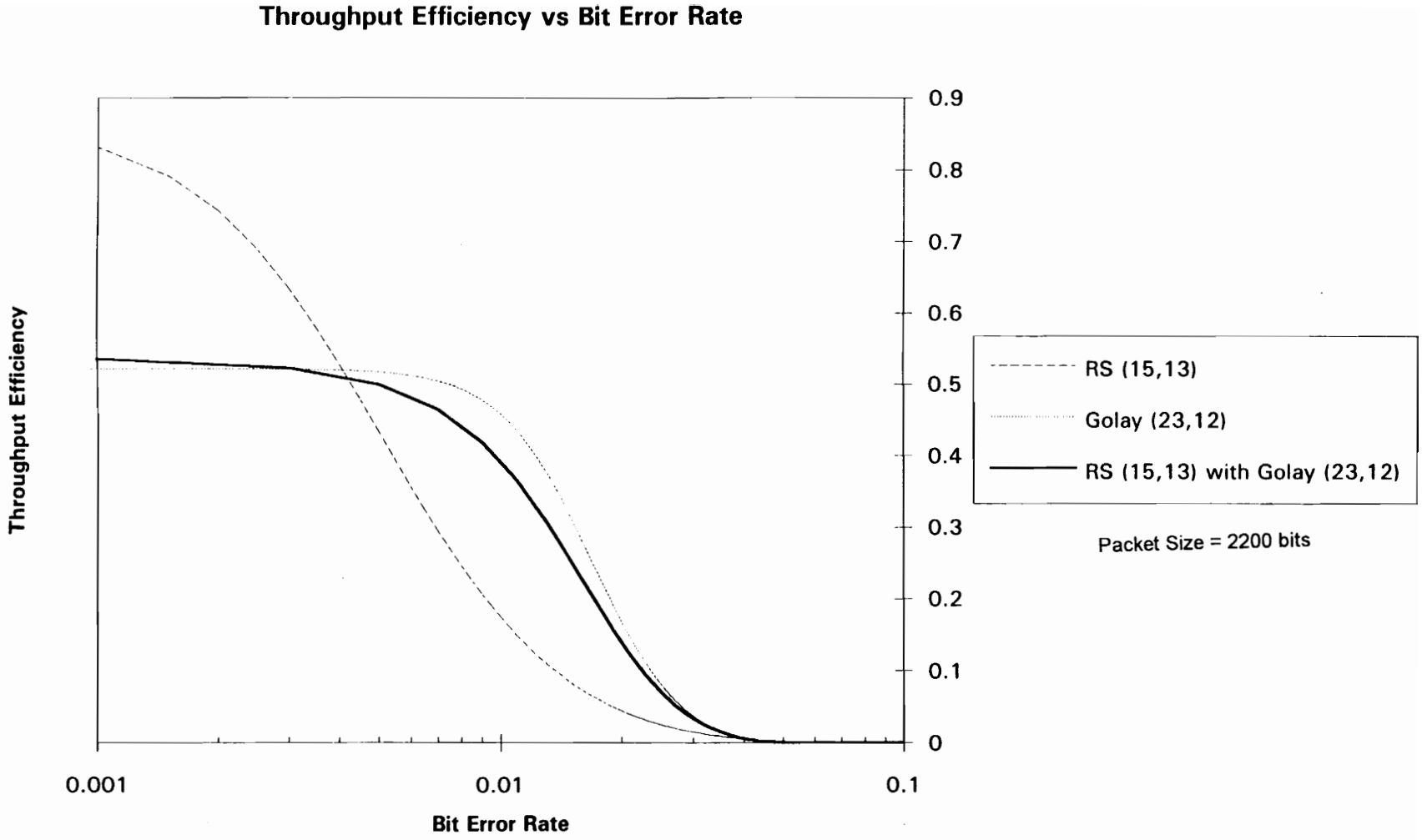

Figure 5-32 Plot of the throughput efficiency vs bit error rate for a packet which uses both the RS (15,13) code and the Golay (23,12) code. The system uses Go-Back-N ARQ. The Golay code is used on the informaition field.

5.3 Concatenated Codes

One powerful method for error control can be accomplished by concatenating two or more codes. In this process the information to be transmitted is first encoded with a code referred to as the *outer* code. This stream of data is then regarded as a new information string and fed into another encoder where the code here is referred to as the *inner* code. At the receiver, the demodulated data is first decoded with a decoder for the inner code, and these symbols are then decoded with a decoder for the outer code.

Concatenation can provide for good error correction, but one subsequent problem is that the resulting overall packet length will be considerably larger then when using only one code. For example, if a packet which is 2200 bits in length is first encoded with a half rate code then this results in a packet of 4400 bits. If the inner code is also a half rate code the total concatenated packet will contain 8800 bits which is four times the length of the original packet. This could severely decrease the throughput efficiency of the system because many more redundant bits are being transmitted than information bits.

5.3.1 Probability of Packet Error

When trying to measure the performance of the concatenated scheme the coded packet can be looked upon as a parallel system. This means that the system will fail only if all of its components fail. Thus, the packet will have an error only if both the inner and outer codes fail to correct any transmission errors. In a parallel system, the reliability is given by the following equation:

$$R_s = 1 - P(\text{all components fail})$$
$$= 1 - \prod_{i=1}^{k} [1 - R_i] \tag{5-7}$$

where k is the total number of components. Converting the terms in the above equation into probabilities of packet error, the total probability of packet error for a concatenated system becomes:

$$P_{pe} = P_{e1} \cdot P_{e2} \tag{5-8}$$

with P_{e1} equal to the probability of packet error for the outer code, and P_{e2} equal to the probability of packet error for the inner code.

Using the above equation for the packet error probability, several different plots were made comparing some of the possible combinations of inner and outer codes. These plots are presented below in Figures 5-33 through 5-38.

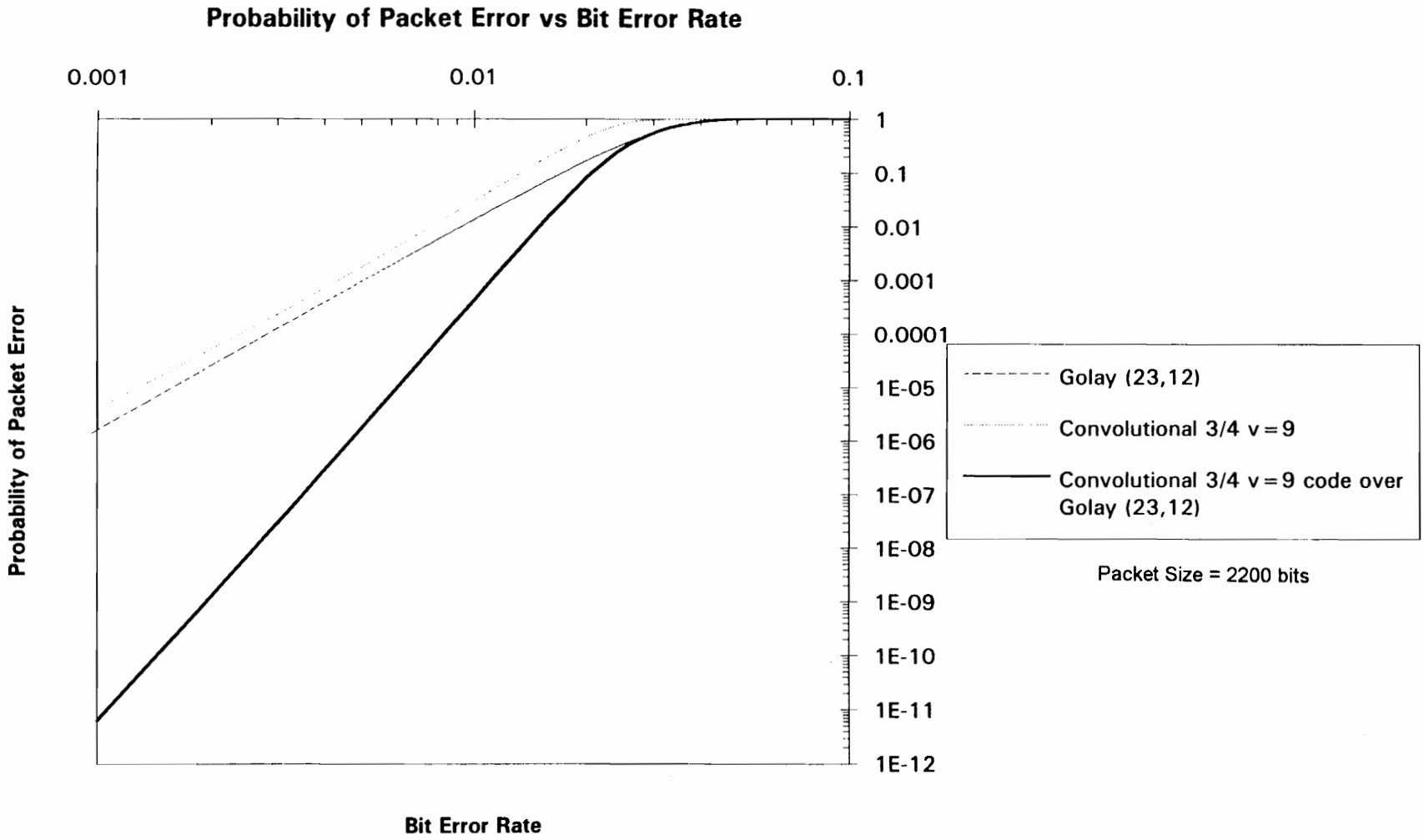

Figure 5-33 Plot of the probability of packet error vs bit error rate for a concatenated packet with 3/4 rate convolutional v=9 code over top the Golay (23,12) code.

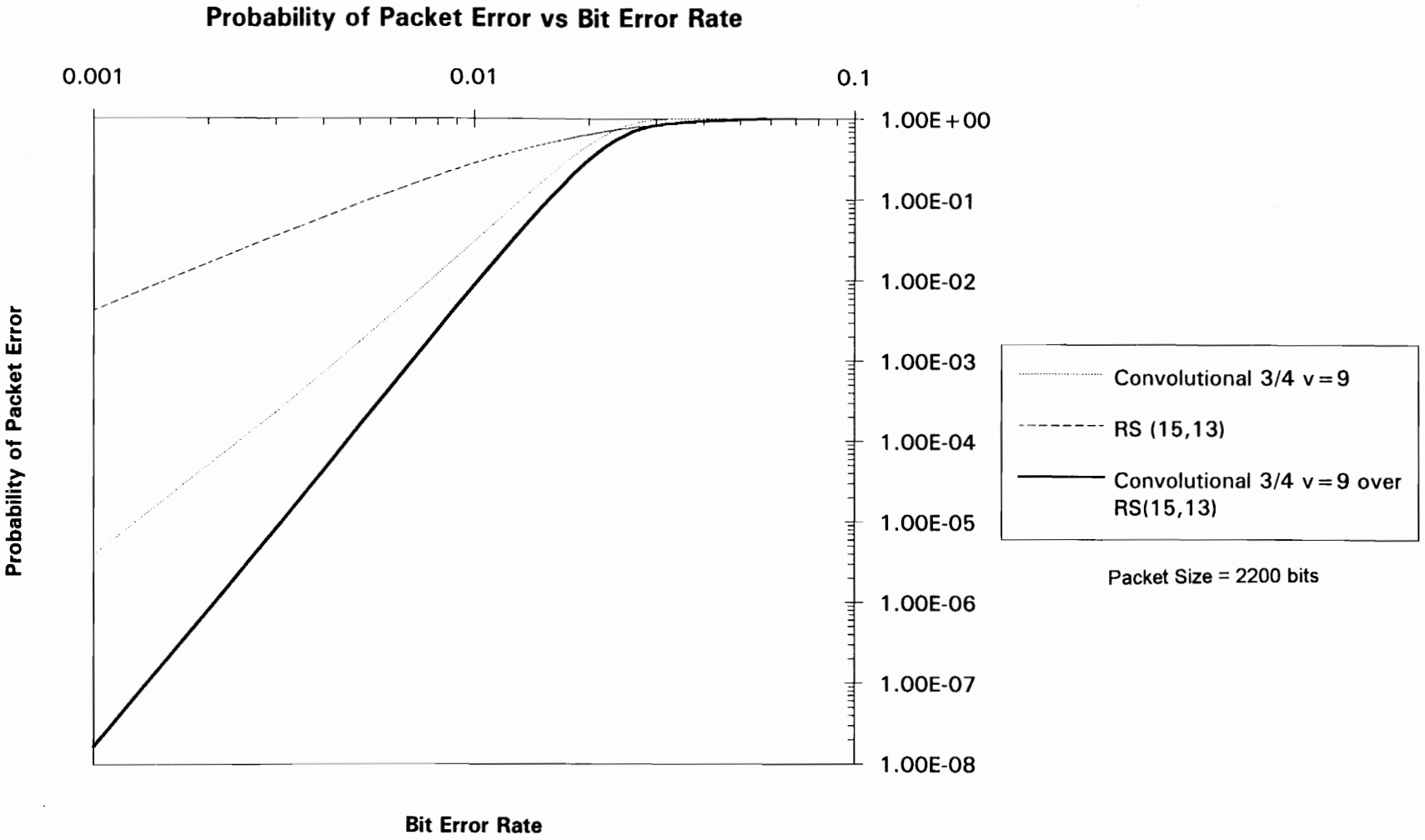

Figure 5-34 Plot of the probability of packet error vs bit error rate for a concatenated packet with 3/4 rate convolutional v=9 code over top the RS (15,13) code.

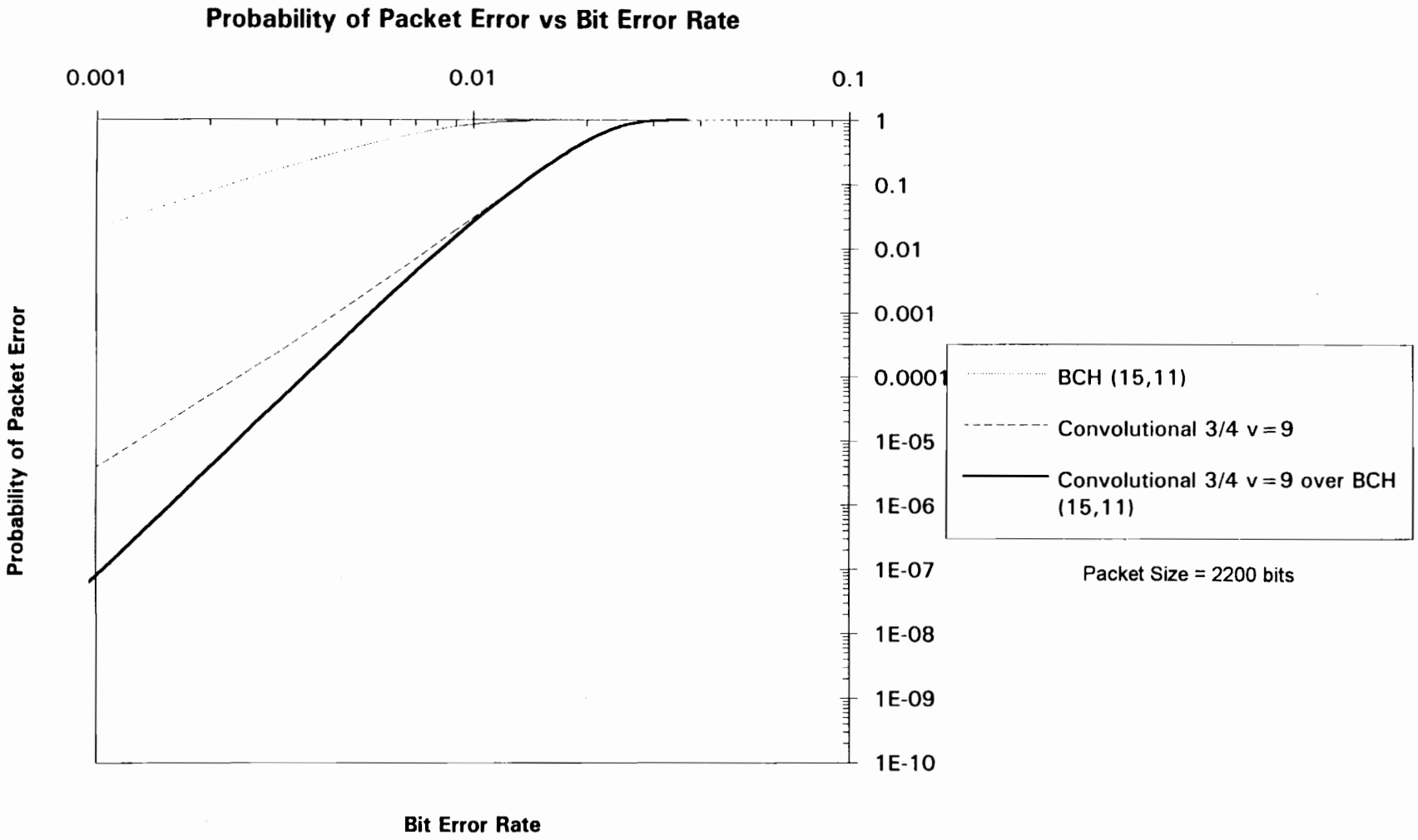

Figure 5-35 Plot of the probability of packet error vs bit error rate for a concatenated packet with 3/4 rate convolutional v=9 code over top the BCH (15,11) code.

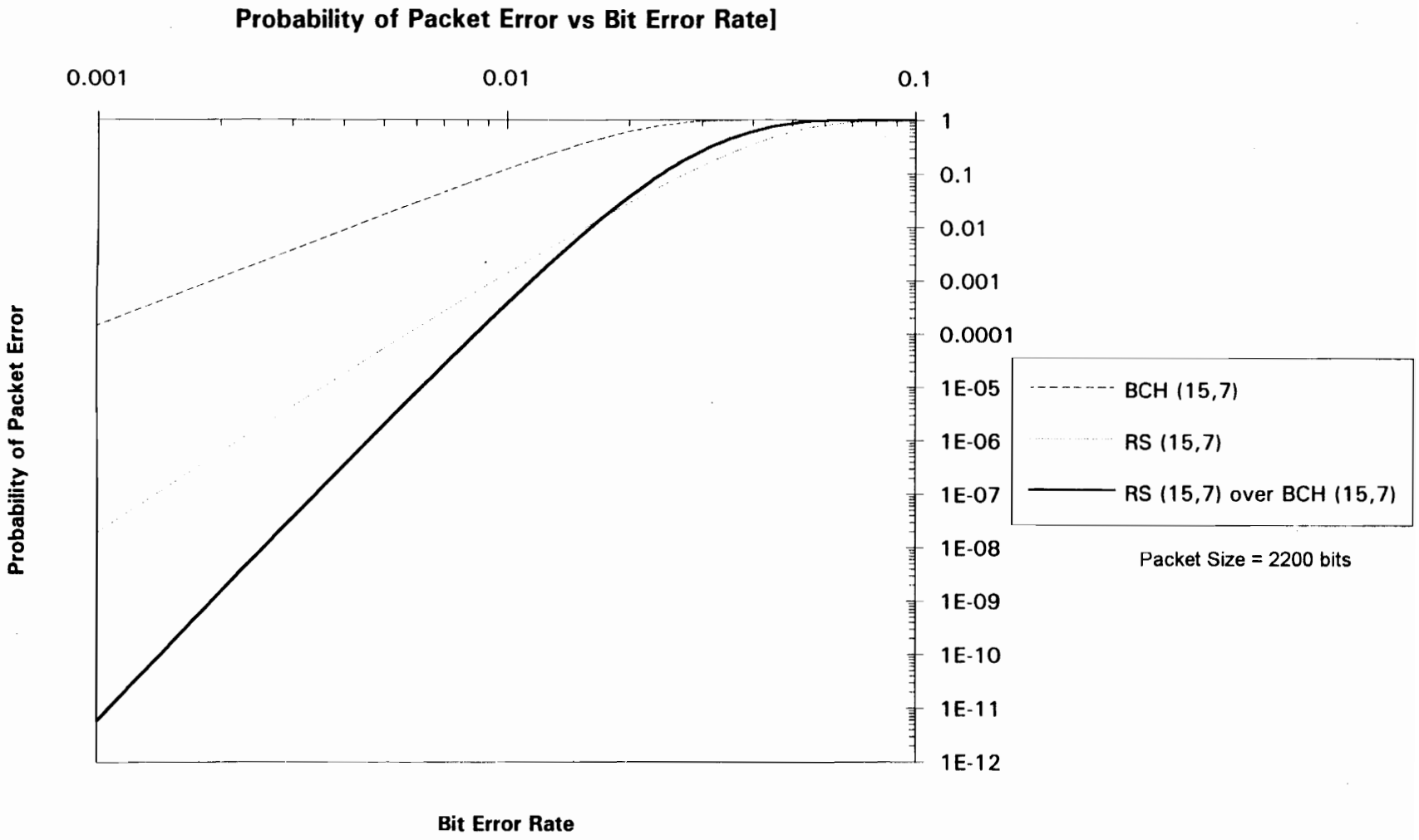

Figure 5-36 Plot of the probability of packet error vs bit error rate for a concatenated packet with RS (15,7) over the BCH (15,7) code.

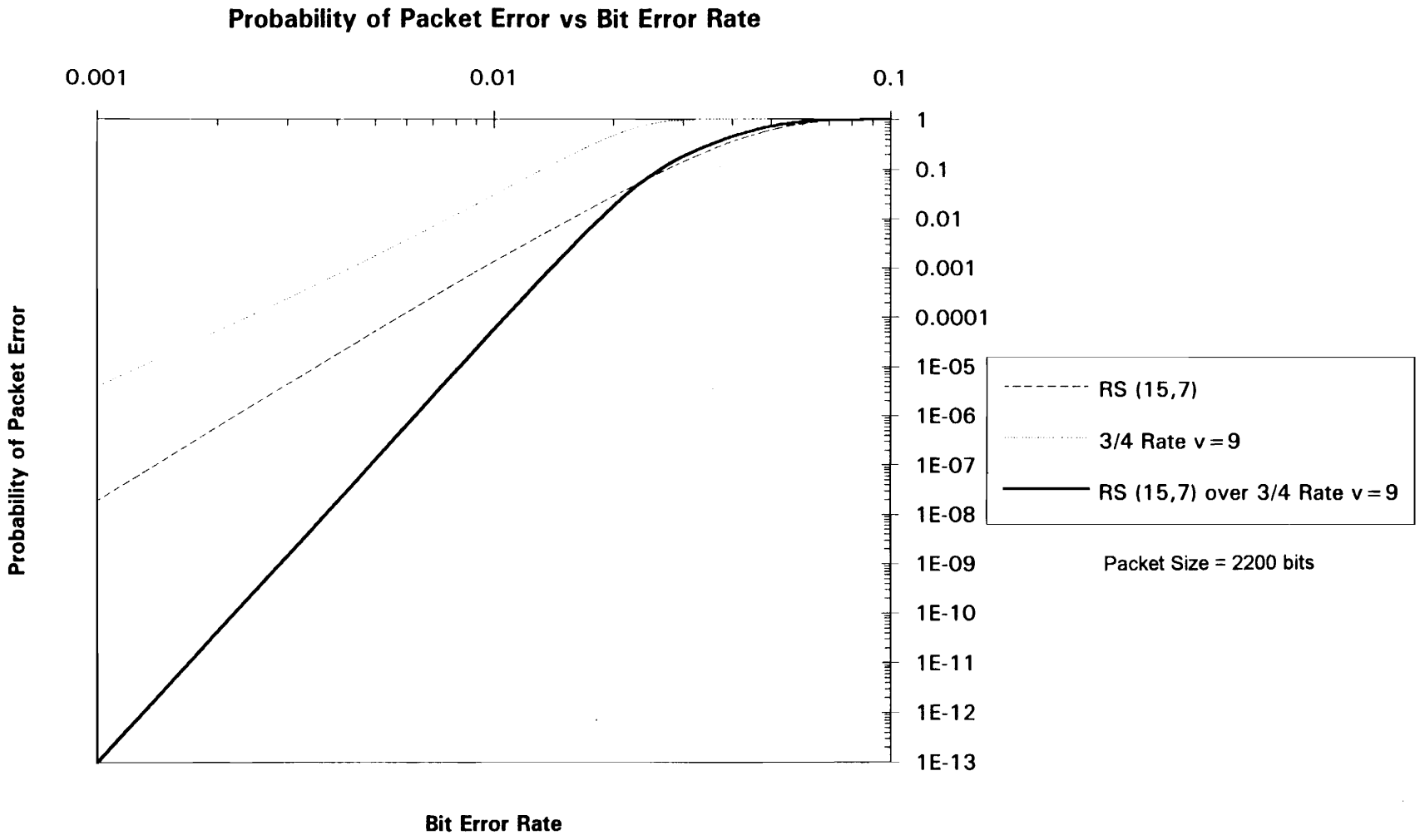

Figure 5-37 Plot of the probability of packet error vs bit error rate for a concatenated packet with RS (15,7) over 3/4 rate convolutional v=9 code .

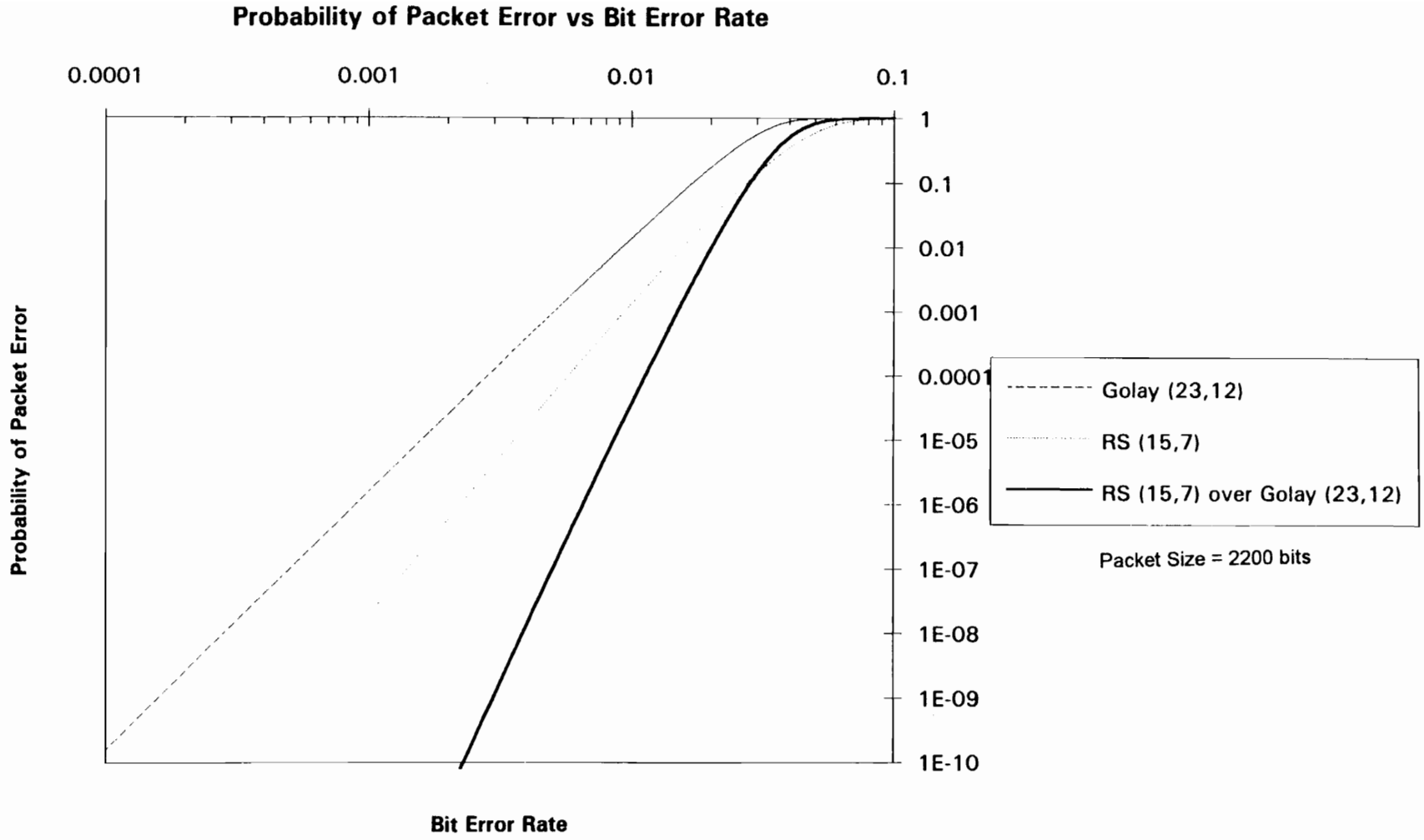

Figure 5-38 Plot of the probability of packet error vs bit error rate for a concatenated packet with RS (15,7) over the Golay (23,12) code.

5.3.2 Throughput Efficiency

The throughput of the concatenated system will be the same as the throughput equations given in Chapter 3, depending upon which ARQ scheme is being used. For the following graphs, the Go-back-N ARQ has been used and the probability of packet error is that of the graphs in section 5.3.1.

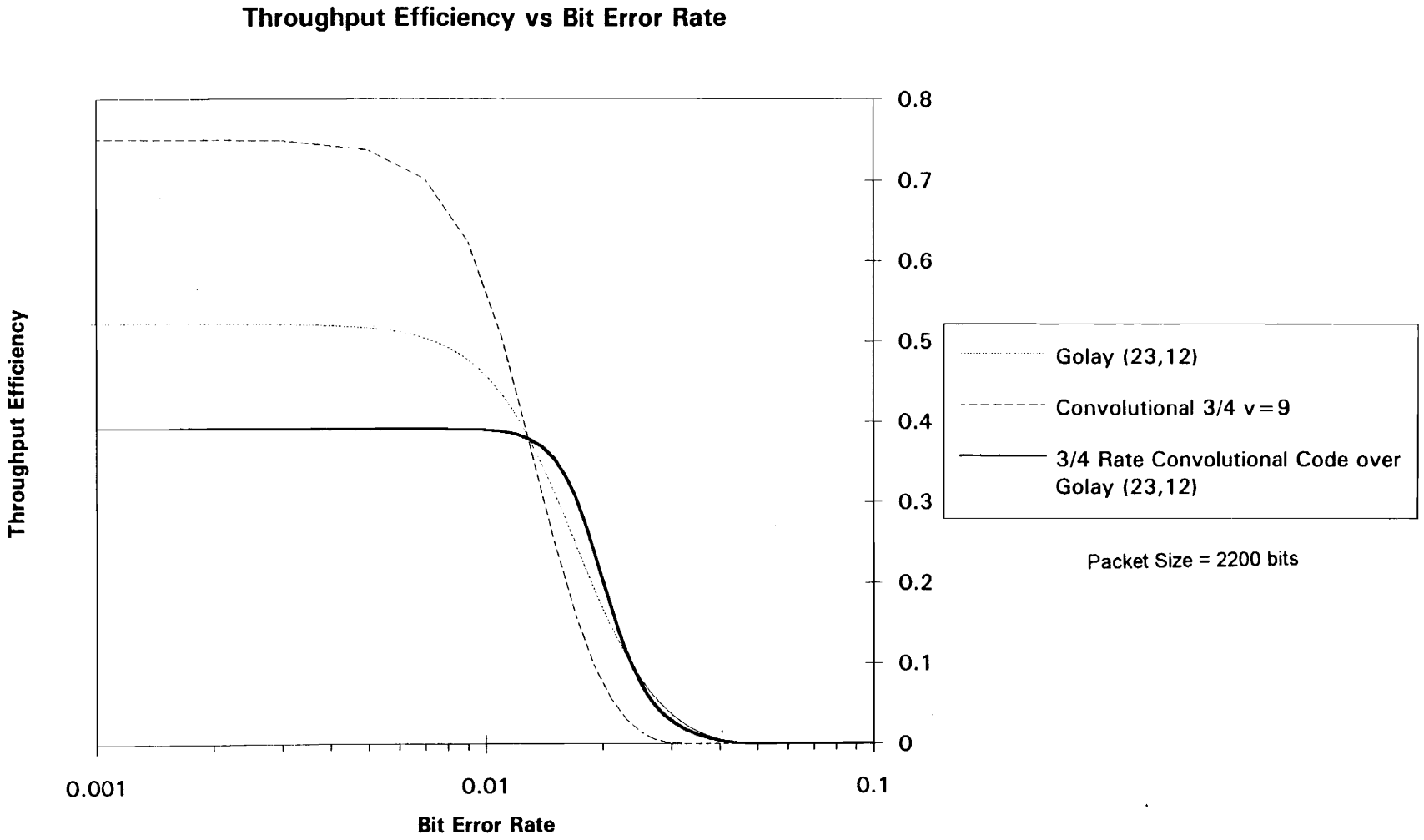

Figure 5-39 Plot of the throughput efficiency vs bit error rate for a concatenated packet with 3/4 rate v=9 convolutional code over the Golay (23,12) code.

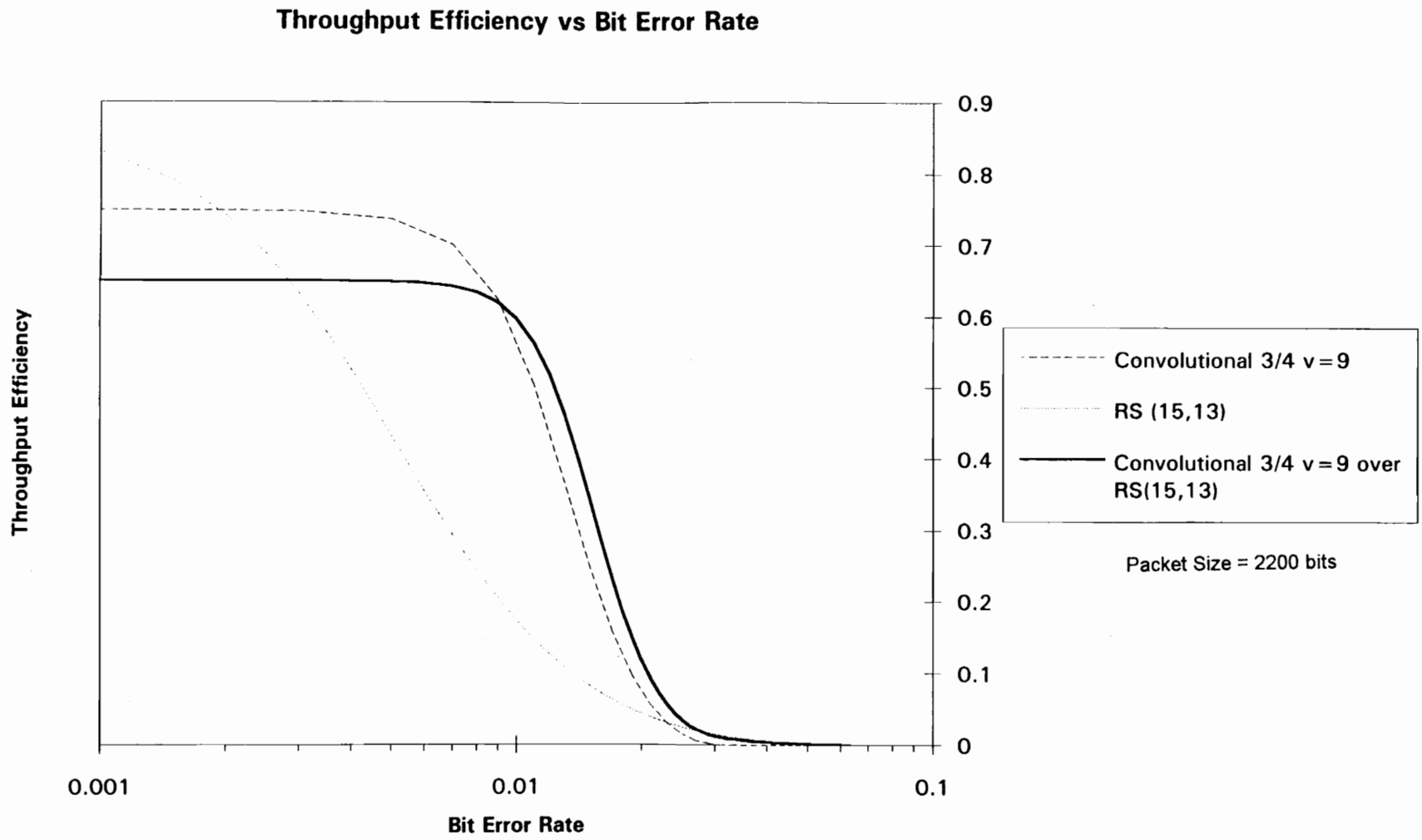

Figure 5-40 Plot of the throughput efficiency vs bit error rate for a concatenated packet with 3/4 rate v=9 convolutional code over the RS (15,13) code.

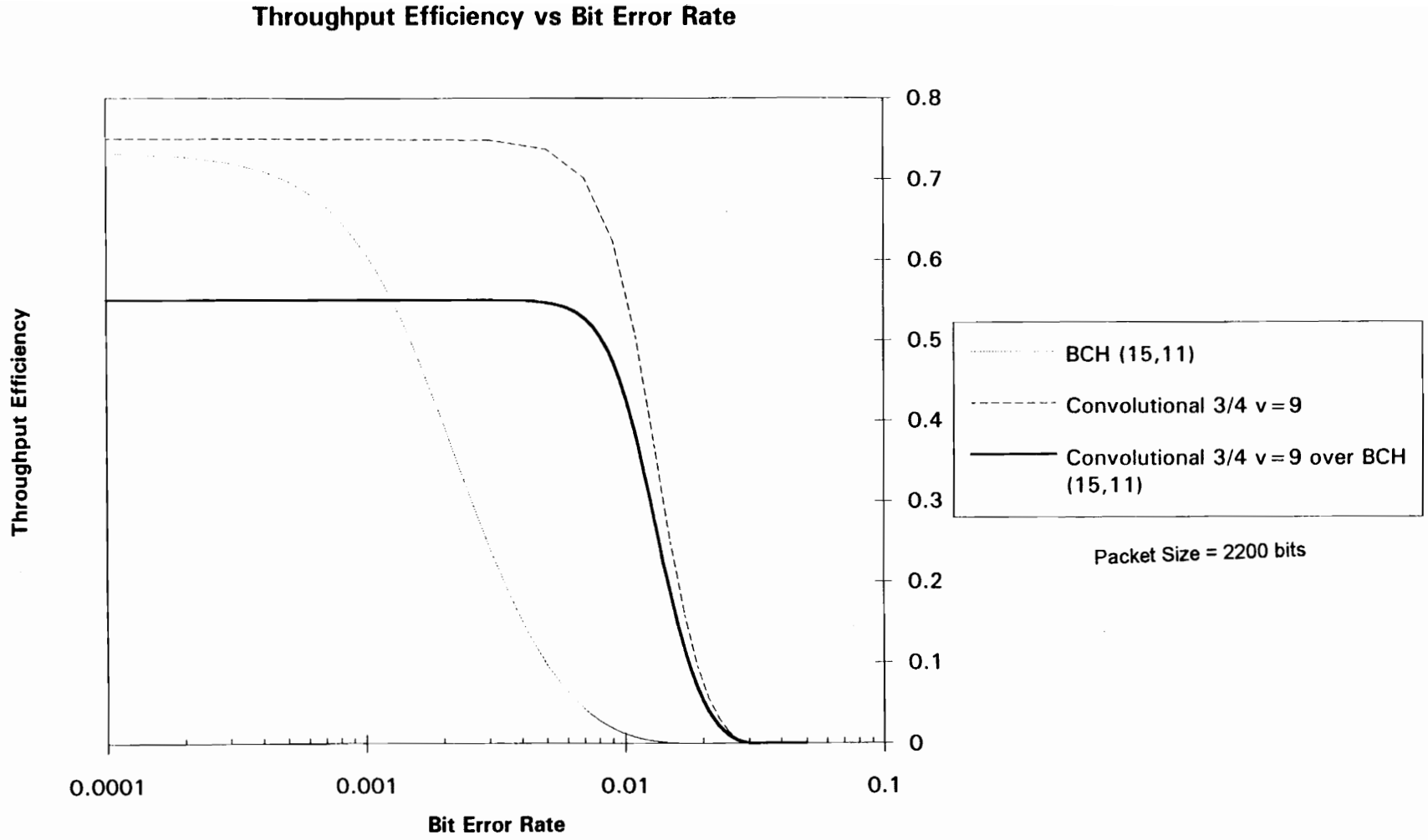

Figure 5-41 Plot of the throughput efficiency vs bit error rate for a concatenated packet with 3/4 rate v=9 convolutional code over the BCH (15,11) code.

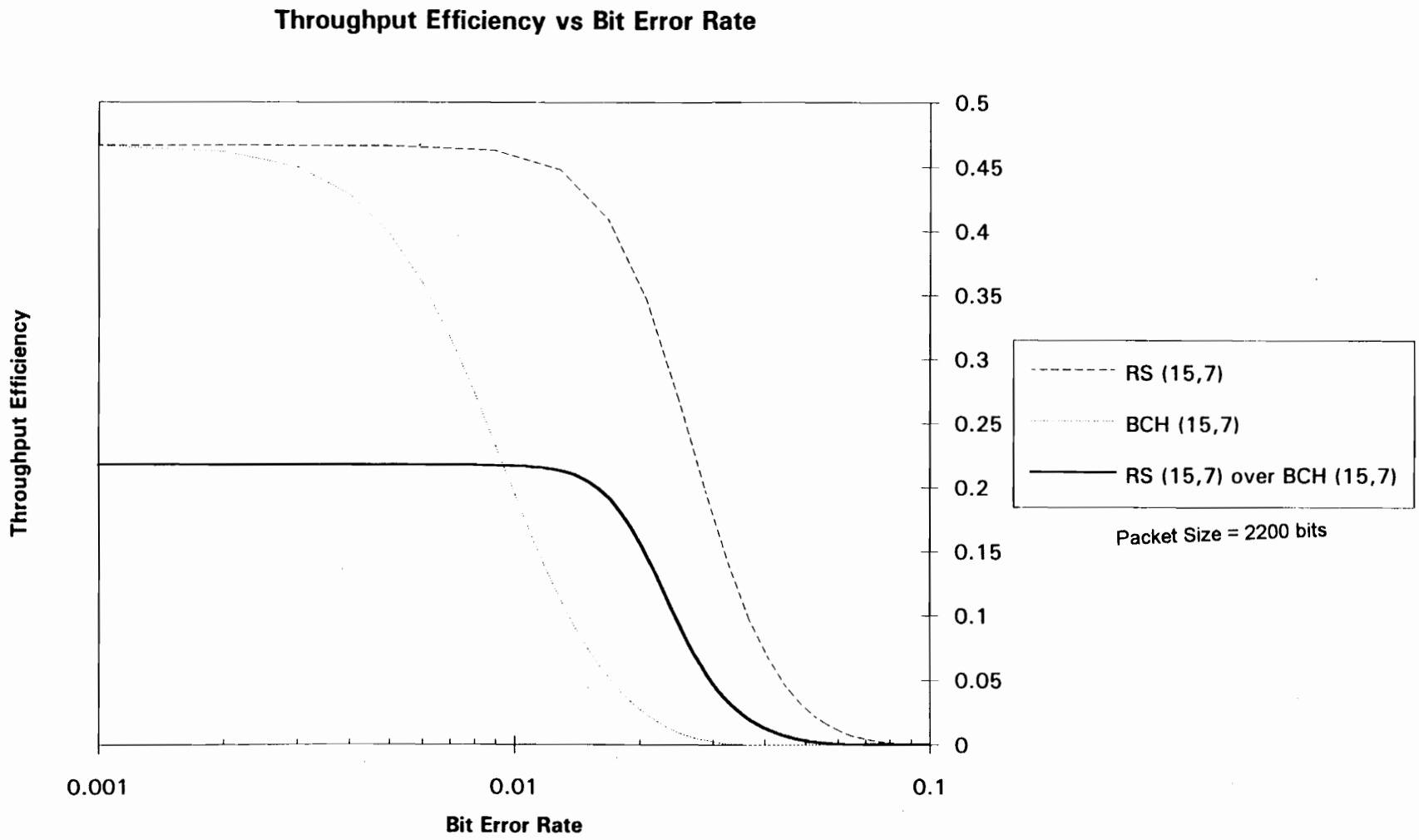

Figure 5-42 Plot of the throughput efficiency vs bit error rate for a concatenated packet with RS(15,7) over BCH (15,7) code.

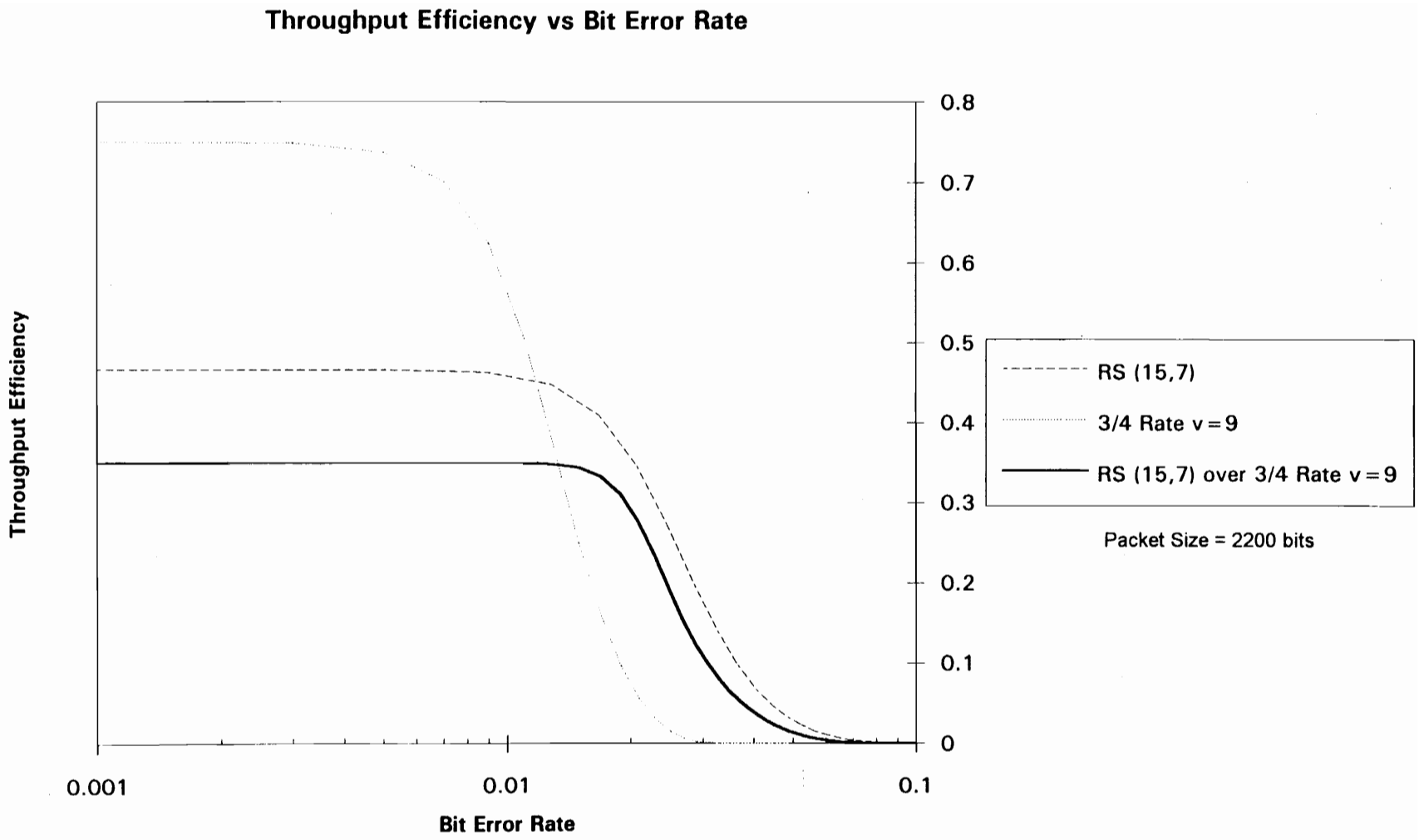

Figure 5-43 Plot of the throughput efficiency vs bit error rate for a concatenated packet with RS (15,7) over the 3/4 rate v=9 convolutional code.

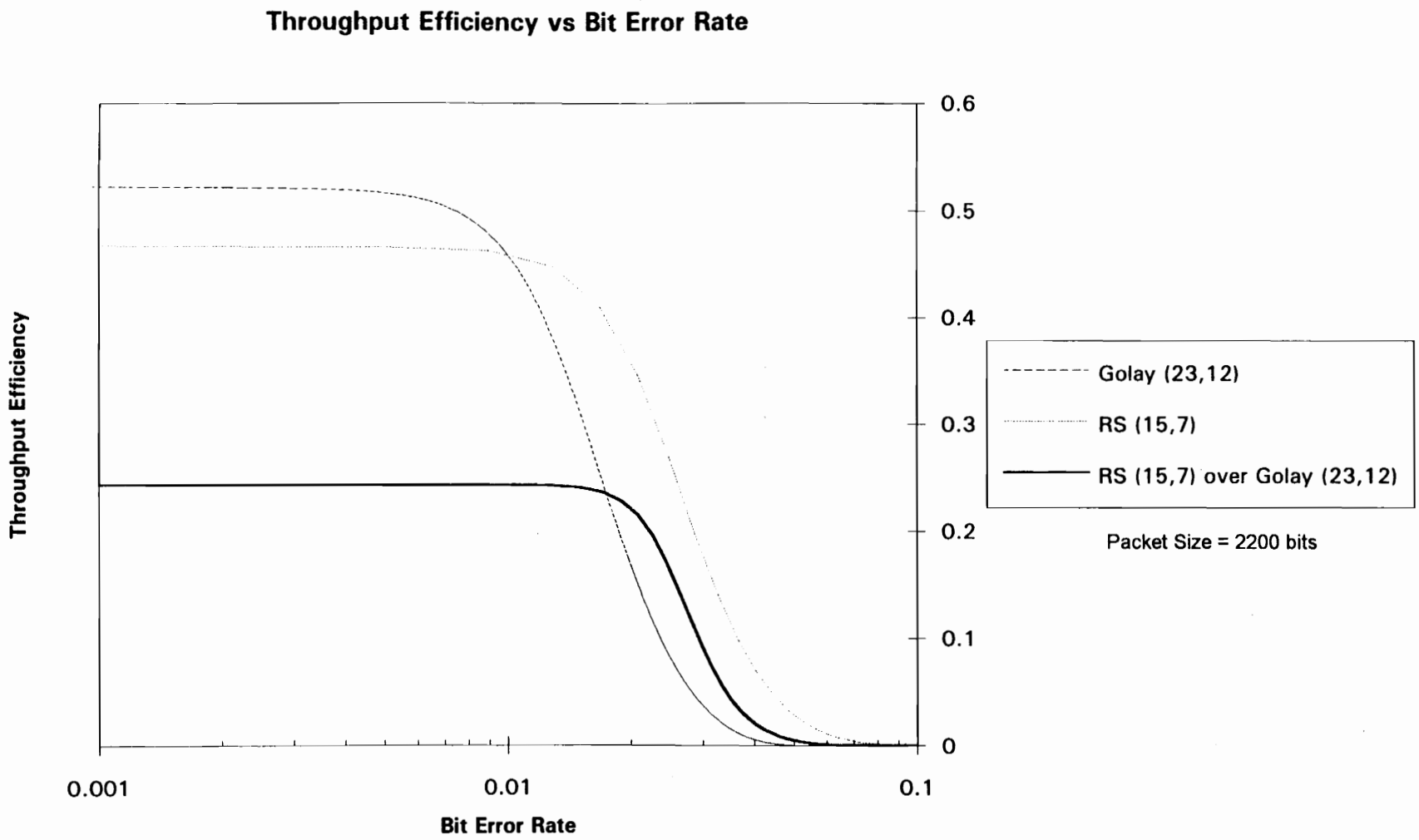

Figure 5-44 Plot of the throughput efficiency vs bit error rate for a concatenated packet with RS(15,7) over the Golay (23,12) code.

5.4 The Golay Code

The Golay code is an interesting code because it is one of the few known perfect codes (see Chapter 3). In order to measure the performance of this code, a computer simulation was used. Packets of length 2208 bits were encoded and then corrupted at the bit error rate. After this they were then sent to the decoder..

In order to encode the data, the multiplication shown by Equation 5-9 was used. In this equation g(x) is given by

$$g(x) = 1 + x^2 + x^4 + x^5 + x^6 + x^{10} + x^{11} \tag{3.3}$$

The resulting code word u(x)g(x) is 23 bits. In order to make this a half rate code, a parity bit could be added making the code word 24 bits.

$$[u_0 u_1 \ldots u_{11}] \begin{bmatrix} g(x) \\ xg(x) \\ \cdot \\ \cdot \\ \cdot \\ x^{11}g(x) \end{bmatrix} = (u_0 + u_1 x + \ldots + u_{11} x^{11}) g(x) \tag{5-9}$$

In the above equation, u_n refers to the bits of the uncoded information. To decode the received information packet, a decoding algorithm is needed. The one originally chosen for this simulation was a variation of the error-trapping decoder and can be found in Adamak [1] and Lin and Costello [2]. This method is presented as follows:

Step 1: Compute the Syndrome of the received word **r**

Step 2: Find the syndrome $\mathbf{s}^{(i)}$ of the cyclic shifts $\mathbf{r}^j$ of the received word (to the right) for l=0,1,...,22. If the Hamming weight of

some $\mathbf{s}^{(i)}$ is at most 3, the decoded word is the ith cyclic shift of $\mathbf{r}^{(i)} - \mathbf{s}^{(i)}$ to the left

Step 3: If all the syndromes in step 2 have Hamming weight larger than 3, change one bit of the received word, say, the bit r_0. Repeat step 2, searching for a syndrome of Hamming weight at most 2. If such a syndrome is found, the decoding is finished: the syndrome plus the bit r_0 form the error pattern to be corrected. If no syndrome of weight at most 2 is found, then r_0 is correct. Reset it to its original value, and repeat step 3 with other bits changed.

5.4.1 Performance

In order to measure the performance of this code, the encoded packet was first corrupted at the bit error rate. It was then passed to the decoder for decoding. If the decoded packet did not match the original packet, then an error was found to have occurred and this packet was marked. Once 10 such markings had occurred the probability of packet error could easily be obtained by dividing 10 (the number of marked packets) by the total number of packet which had been sent. A plot of the probability of packet error and the throughput for the simulated Golay code can be seen in Chapter 6.

CHAPTER 6

DATA ANALYSIS

This chapter looks at the data obtained in Chapter 5 and analyses the results to determine which, if any, error control scheme should be added to the current AX.25 packet radio system. The decision will be based on the criteria specified in Chapter 1: throughput, code rate, system complexity and ease of implementation.

A system with the largest throughput for a given bit error rate will be considered to have better performance than one with a lower throughput because more information is passed through the system in a given period of time when the throughput is larger. The maximum throughput value is equal to the code rate, so a code with a large rate is preferred in order to maximize the throughput. A system which is complex to build or not easily implemented will not be chosen.

6.1 Throughput Efficiency

The main parameter used for performance analysis will be the throughput efficiency. A system which can maintain a constant throughput with increasing bit error rate is desired because this means that as the channel error rate increases, information can still be passed through the system at a constant rate.

The first codes to be examined will be the BCH codes. Figures 6.1 and 6.2 show the throughput performance for various BCH codes. On each of these graphs, the code with the highest code rate has the largest throughput as expected, but the code with the highest rate also has the lowest performance. This means that the throughput begins to drop off at a lower bit error rate. For example, the 0.333 rate (15,5) BCH code maintains its peak throughput value until the bit error rate reaches 0.01, while the throughput of the 0.733 rate (15,11) BCH code begins to decline at a bit error rate of 0.005. The (15,5) BCH code has 10 redundant bits in a code word where the (15,11) code only has 4 bits. Since the (15,5) code adds more redundancy, it is expected that its error correction capabilities are better than the (15,11) code. This translates into increased throughput performance for the packet system because more packets can be passed to the user the first time they are received, instead of calling for packet re-transmission which results in a decline in the throughput.

One important note is that the lager the value of n in the (n,k) BCH codes, the better the performance. One reason for this is that when a large block length is used, it is possible to make a code with a large minimum distance which increases the error correction capability of the code.

The next codes to be looked at will be the convolutional codes. Figures 6.3 through 6.6 show the throughput performance for various rate convolutional codes with different constraint lengths. On these plots, it is seen that for each of the different code rates, the code with the largest constraint length has the better performance. Comparing the different rate codes, it is observed that the codes

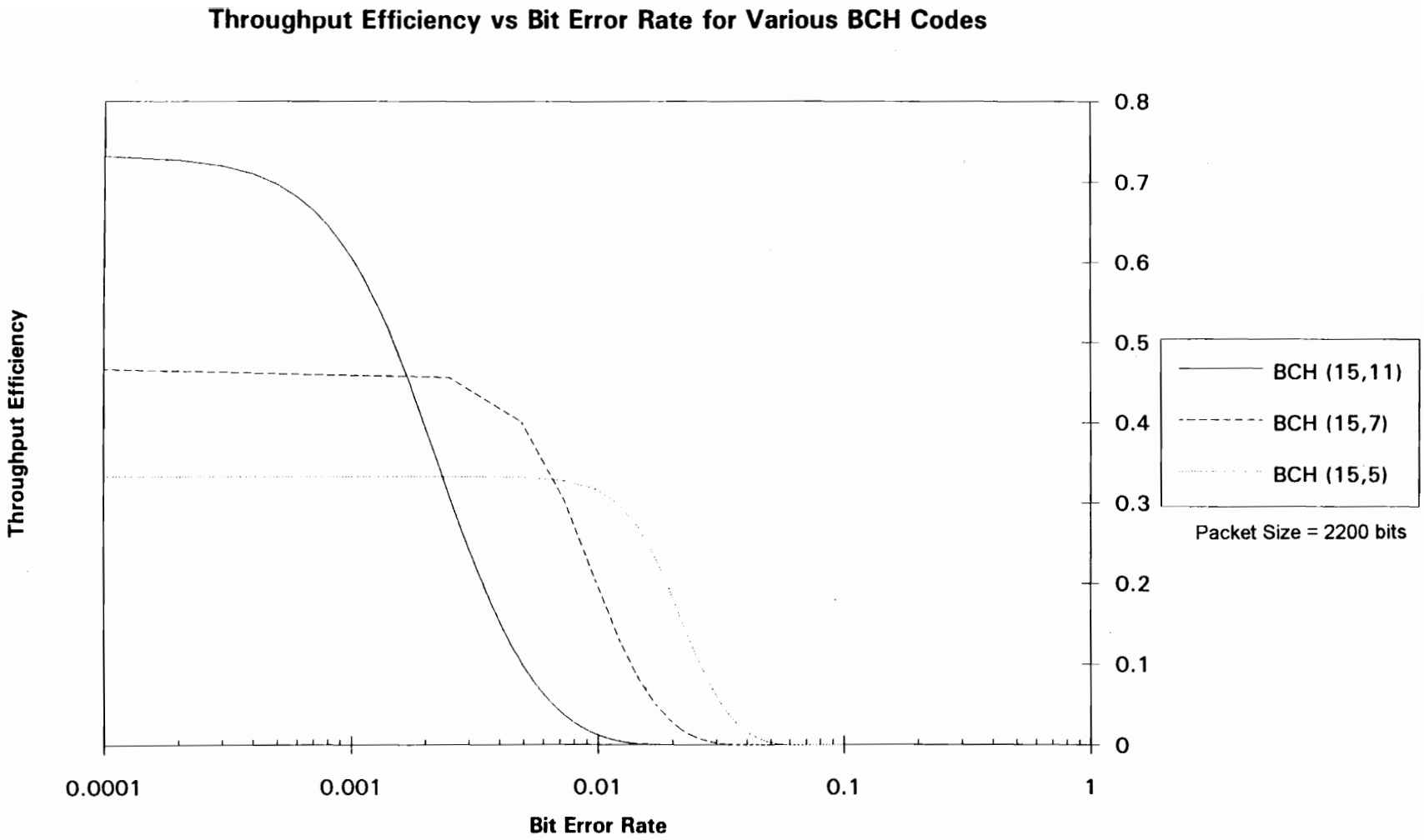

Figure 6-1 Plot of the throughput efficiency vs bit error rate for various length BCH codes. The system uses Go-Back-N ARQ.

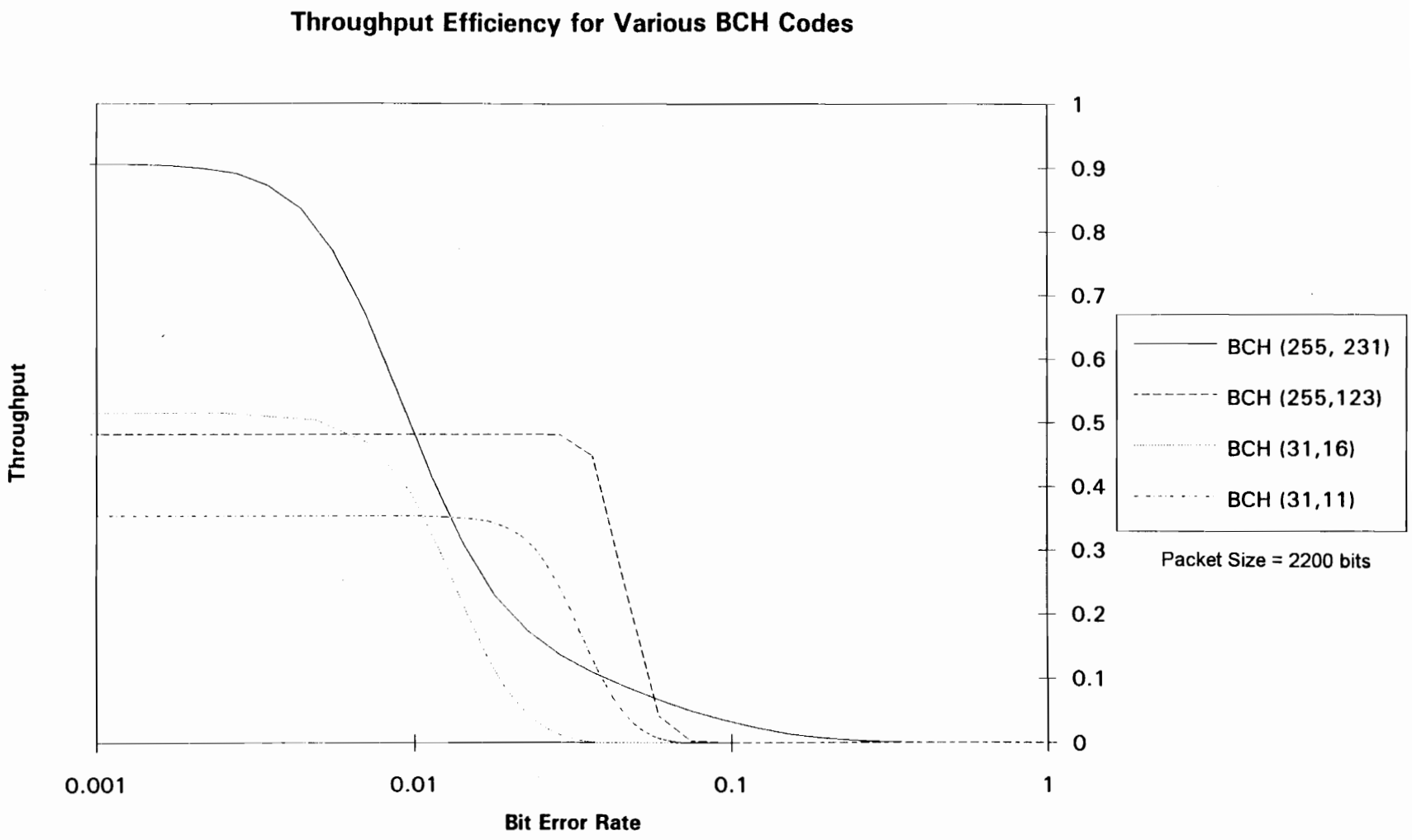

Figure 6-2 Plot of the throughput efficiency vs bit error rate for various length BCH codes. The system uses Go-Back-N ARQ.

with the lower rates perform better, but for reasons mentioned before, they have a lower throughput. For example, the 1/3 rate, constraint length ν=9 convolutional code has a maximum throughput of 0.333, while the 3/4 rate ν=9 convolutional code has a maximum throughput of 0.75, but the 3/4 codes throughput begins falling off before the 1/3 rate code.

Just as the ability of BCH codes to correct errors improves as the block length n increases, the performance of convolutional codes will improve as the constraint length increases. This is because the separation of code word sequences increases as the constraint length of a convolutional code increases,. Only a small fraction of the possible input sequences are used, just as with block codes which use only a small number of the possible n-vectors available as code words.

Another set of codes to be tested were some Reed-Solomon codes. Figures 6.7 and 6.8 show the throughput performance for various Reed-Solomon codes. Just like the BCH codes, the Reed-Solomon codes perform better as n increases.

When looking at the equation for the probability of packet error:

$$P_{pe} = 1 - \left(1 - P_c\right)^q \tag{6.1}$$

it can be shown that as q, the number of code words in the packet, approaches infinity, the probability of packet error approaches 1 since the probability of code word error, P_c, is less than or equal one. When q is 1, then the packet error probability is simply the probability of code word error, P_c. A code which results in fewer code words will therefore tend to have a better performance since the

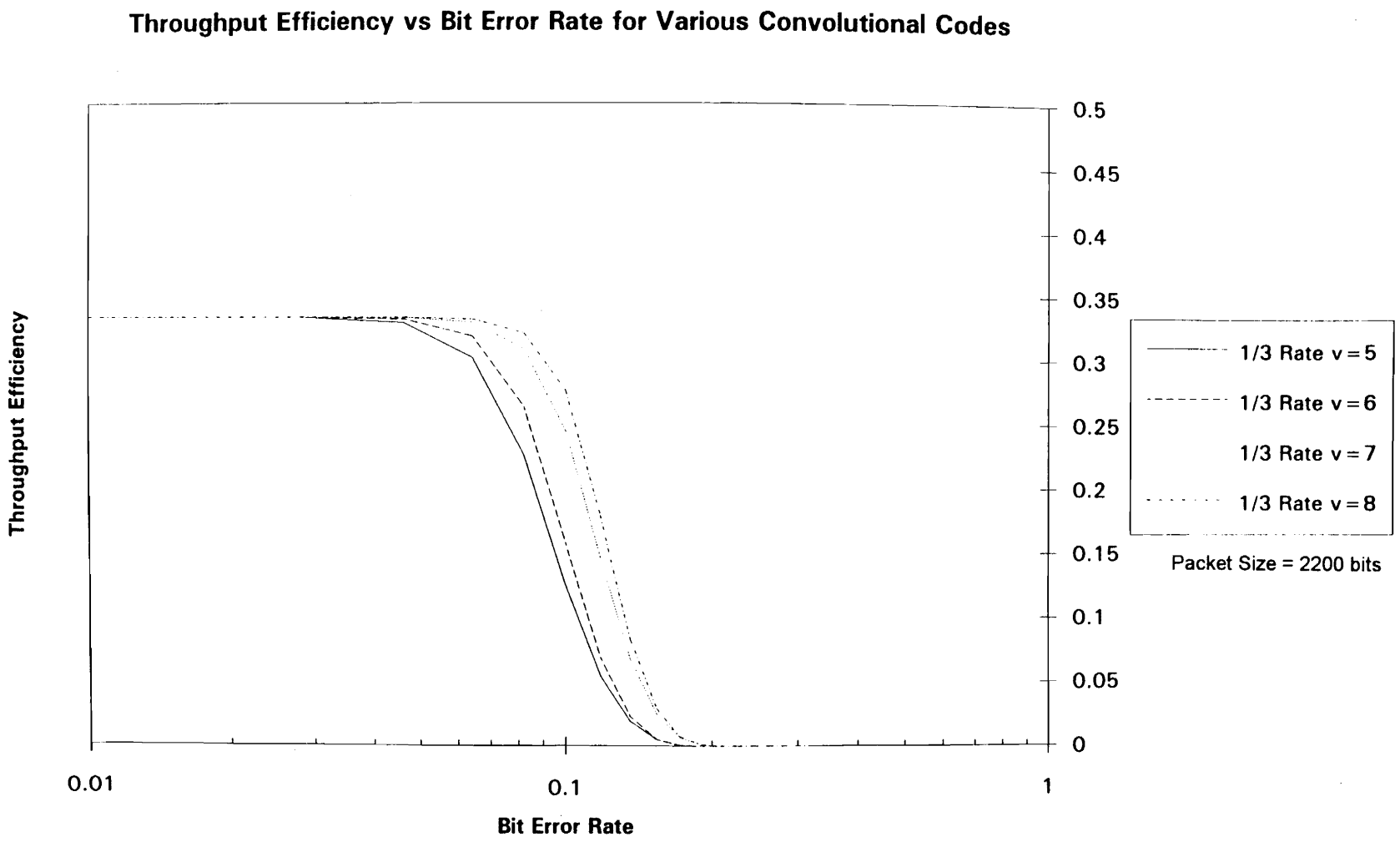

Figure 6-3 Plot of the throughput efficiency vs bit error rate for 1/3 rate convolutional codes with v=5-8. The system uses Go-Back-N ARQ.

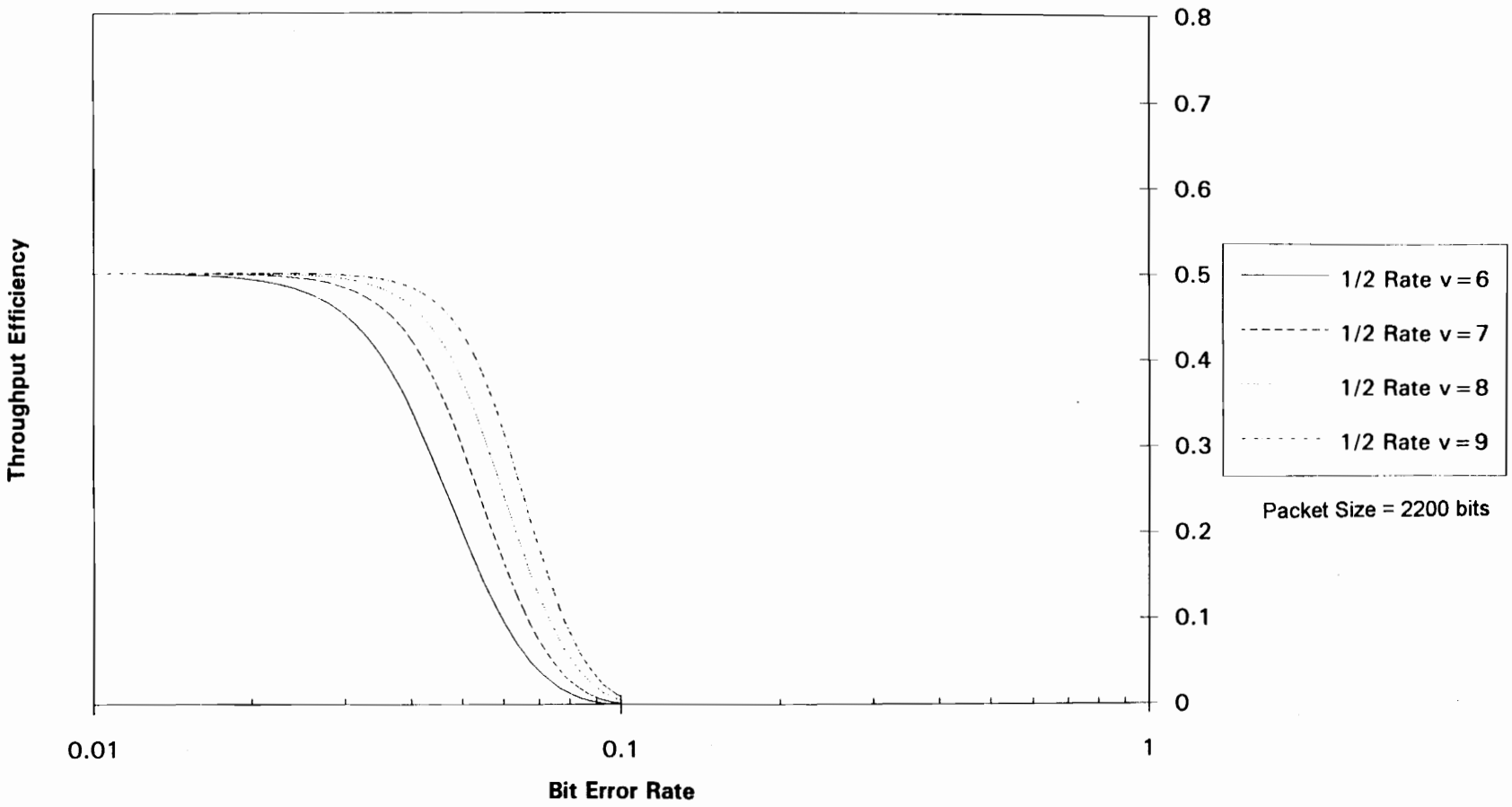

Figure 6-4 Plot of the throughput efficiency vs bit error rate for 1/2 rate convolutional codes with v=6-9. The system uses Go-Back-N ARQ.

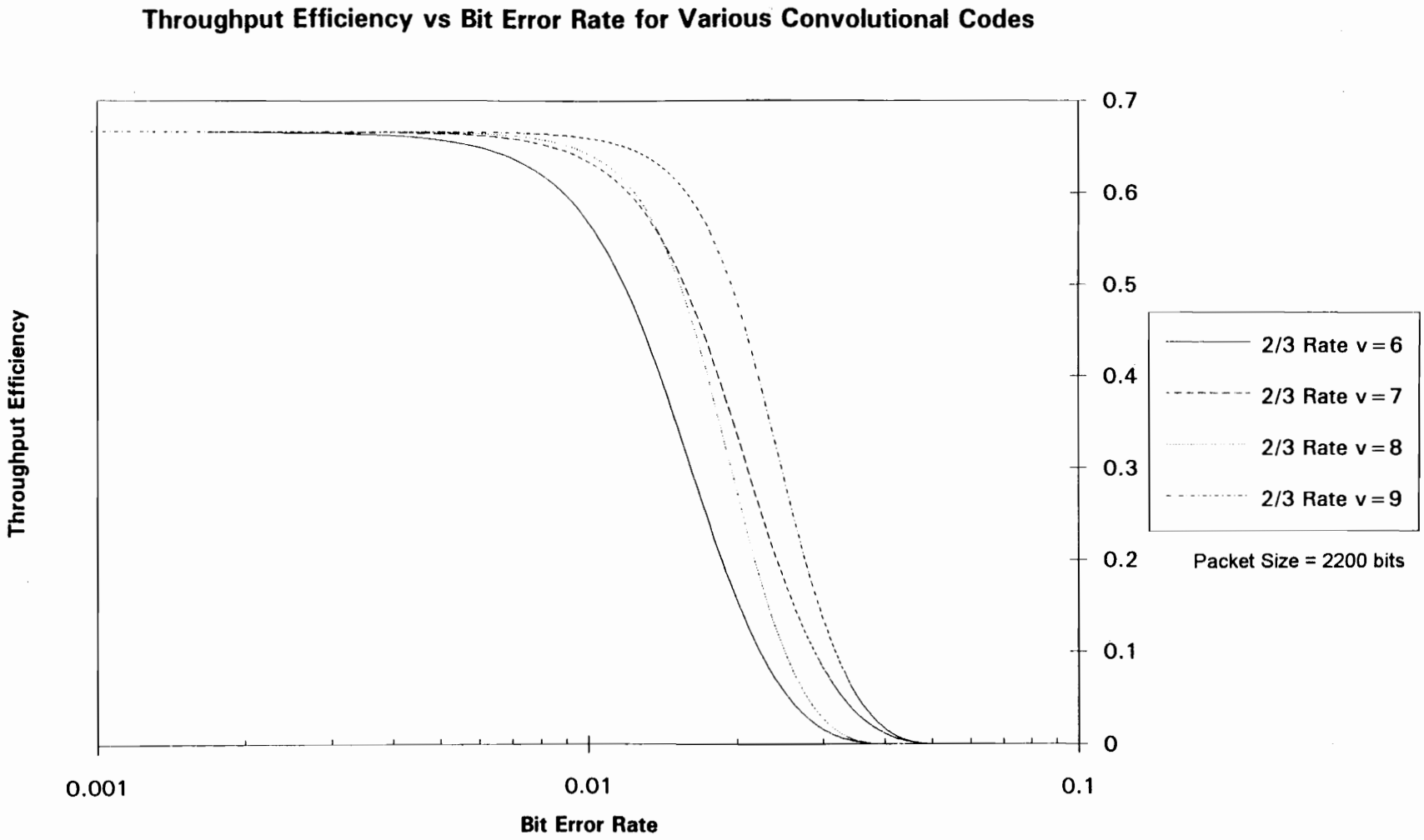

Figure 6-5 Plot of the throughput efficiency vs bit error rate for 2/3 rate convolutional codes with v=6-9. The system uses Go-Back-N ARQ.

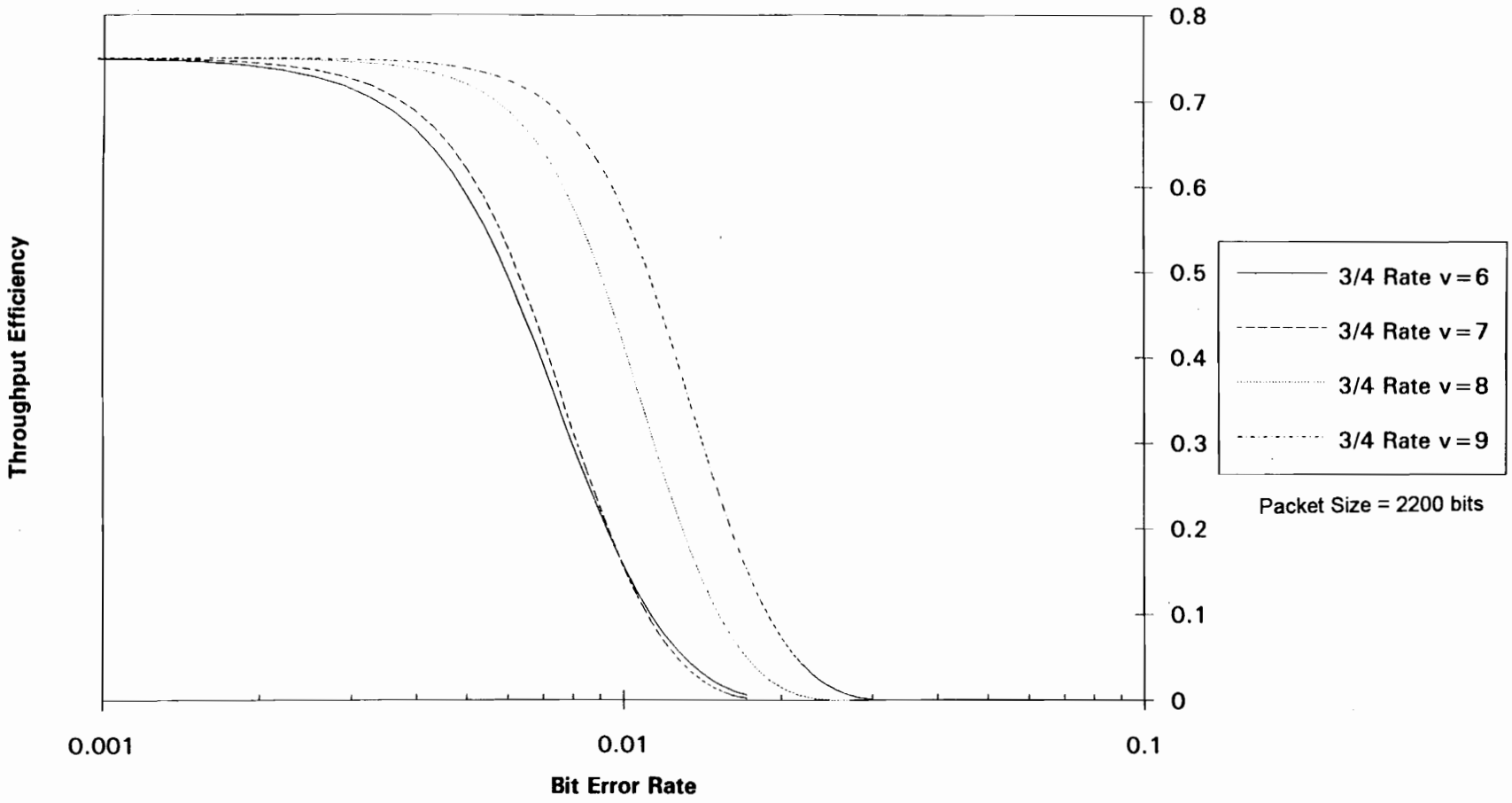

Figure 6-6 Plot of the throughput efficiency vs bit error rate for 3/4 rate convolutional codes with v=6-9. The system uses Go-Back-N ARQ.

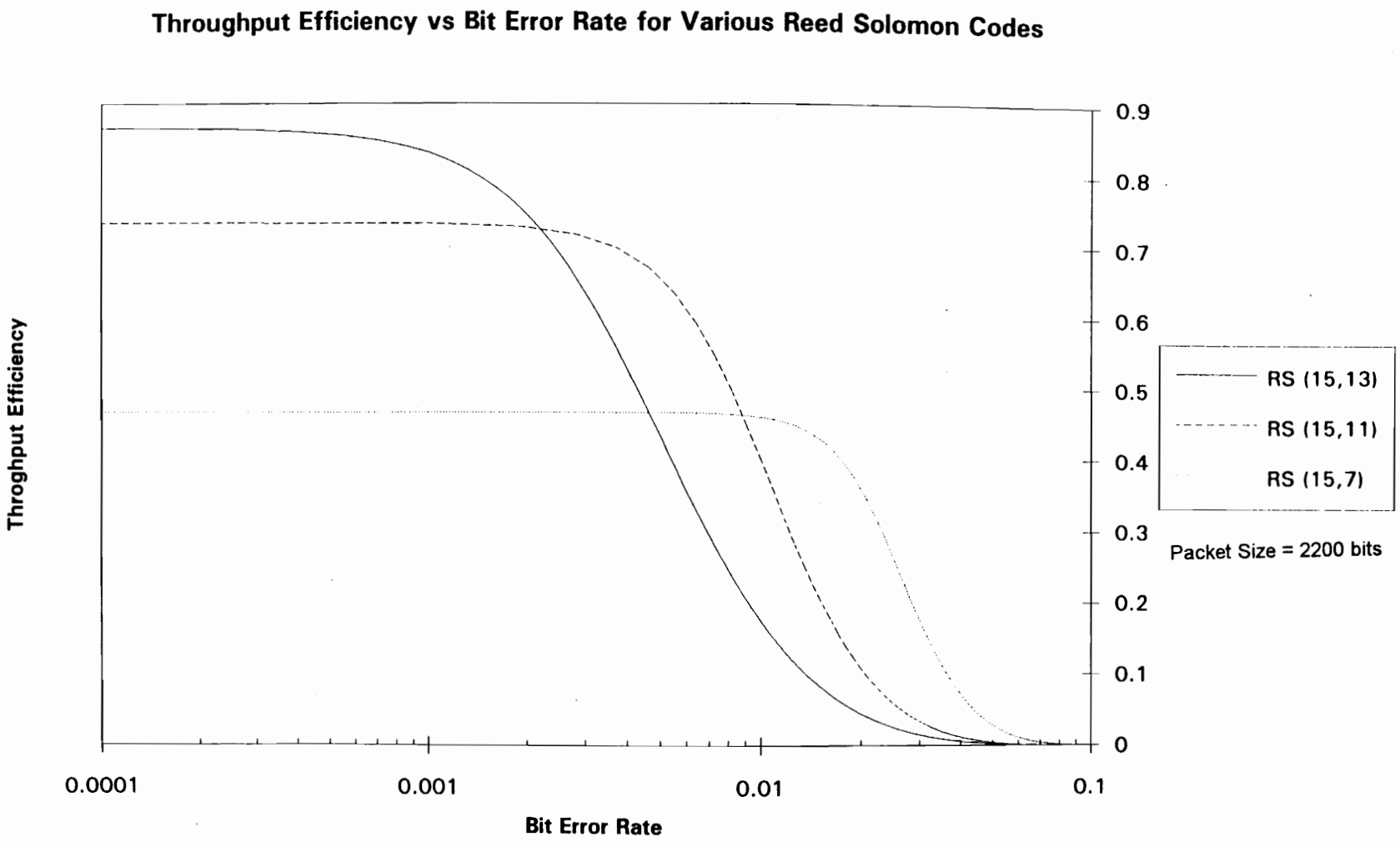

Figure 6-7 Plot of the throughput efficiency vs bit error rate for various length Reed Solomon codes. The system uses Go-Back-N ARQ.

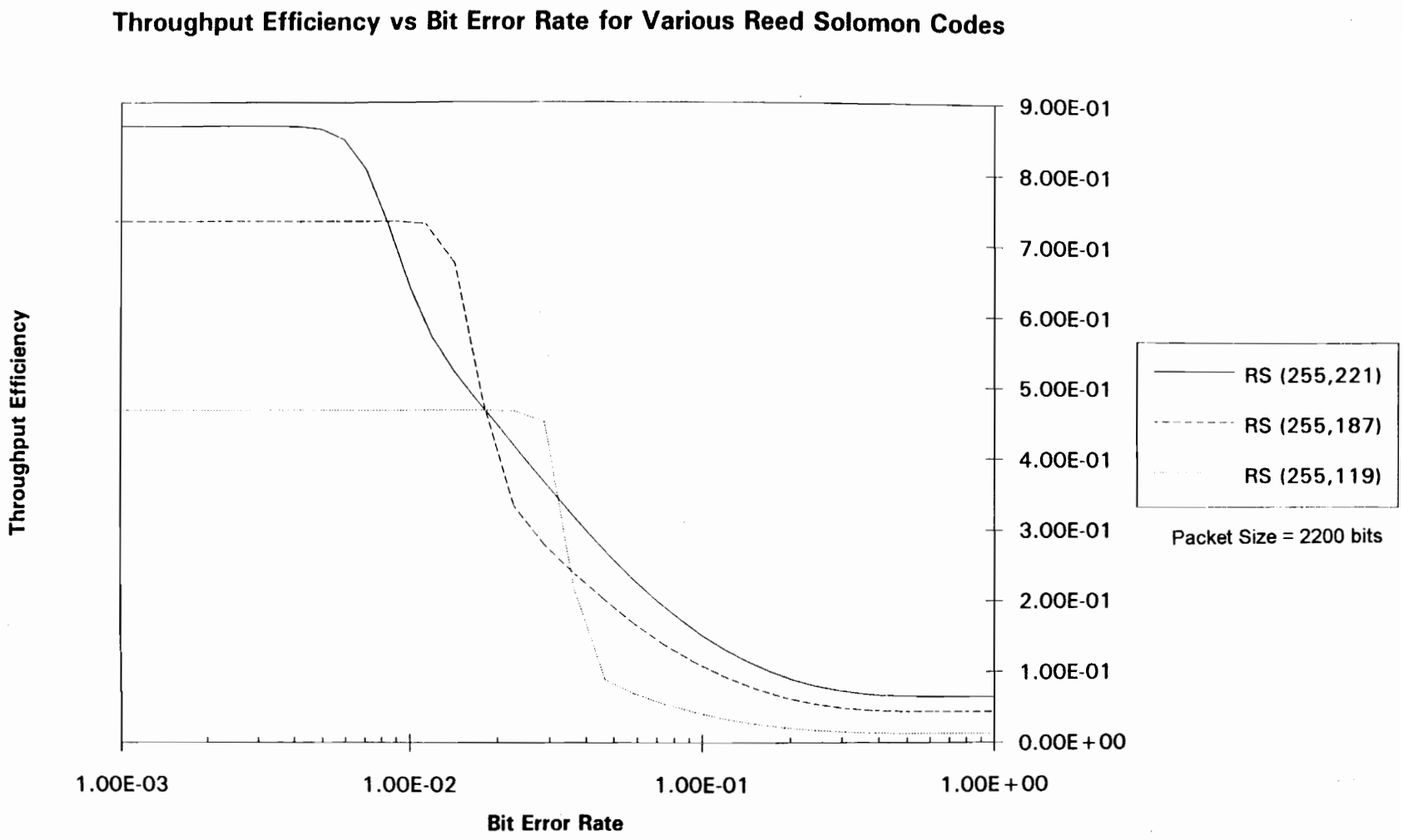

Figure 6-8 Plot of the throughput efficiency vs bit error rate for various length Reed Solomon codes. The system uses Go-Back-N ARQ.

lower throughput. For example, the 1/3 rate, constraint length ν=9 convolutional code has a maximum throughput of 0.333, while the 3/4 rate ν=9 convolutional code has a maximum throughput of 0.75, but the 3/4 codes throughput begins falling off before the 1/3 rate code.

Just as the ability of BCH codes to correct errors improves as the block length n increases, the performance of convolutional codes will improve as the constraint length increases. This is because the separation of code word sequences increases as the constraint length of a convolutional code increases,. Only a small fraction of the possible input sequences are used, just as with block codes which use only a small number of the possible n-vectors available as code words.

Another set of codes to be tested were some Reed-Solomon codes. Figures 6.7 and 6.8 show the throughput performance for various Reed-Solomon codes. Just like the BCH codes, the Reed-Solomon codes perform better as n increases.

When looking at the equation for the probability of packet error:

$$P_{pe} = 1-\left(1-P_c\right)^q \tag{6.1}$$

it can be shown that as q, the number of code words in the packet, approaches infinity, the probability of packet error approaches 1 since the probability of code word error, P_c, is less than or equal one. When q is 1, then the packet error probability is simply the probability of code word error, P_c. A code which results in fewer code words will therefore tend to have a better performance since the

probability of packet error will be lower at a given bit error rate. The throughput is dependent upon the probability of packet error so it benefits from a system which uses fewer code words. In the equation for the throughput of a go-back-N ARQ system

$$\eta_{GBN} = \frac{\left(1-P_{pe}\right)}{\left(1-P_{pe}\right)+P_{pe}N}\left(\frac{k}{n}\right) \tag{6.2}$$

it can be seen that as the probability of packet error, P_{pe}, approaches 0, the throughput approaches its maximum, k/n. If the packet error probability is equal to 1, then the resulting throughput is 0 which means that no new information is being fed through the system. Reed-Solomon (RS) codes can be thought of as accepting $k' = k \cdot m$ information bits and mapping these into code word blocks having length $n' = n \cdot m$ binary channel symbols. Thus, for a packet of 2200 bits, there would be 2200/(k′) code words of length n′. For example, if the (255,119) RS code is used there would be $2200/(11 \cdot 4) = 2.3$ or 3 code words. This gives the value of q = 50 in Equation 6.1. It has been shown that the smaller the value of q in Equation 6.1, the lower the probability of packet error, so it should be evident that Reed-Solomon codes will have good results for the probability of packet error which translates into good system throughput performance.

The next code studied was the Golay Code. Figure 6.9 shows a plot of the probability of packet error for the theoretical Golay code and for the code when it was simulated on a computer. This graph shows that the simulated code performed as

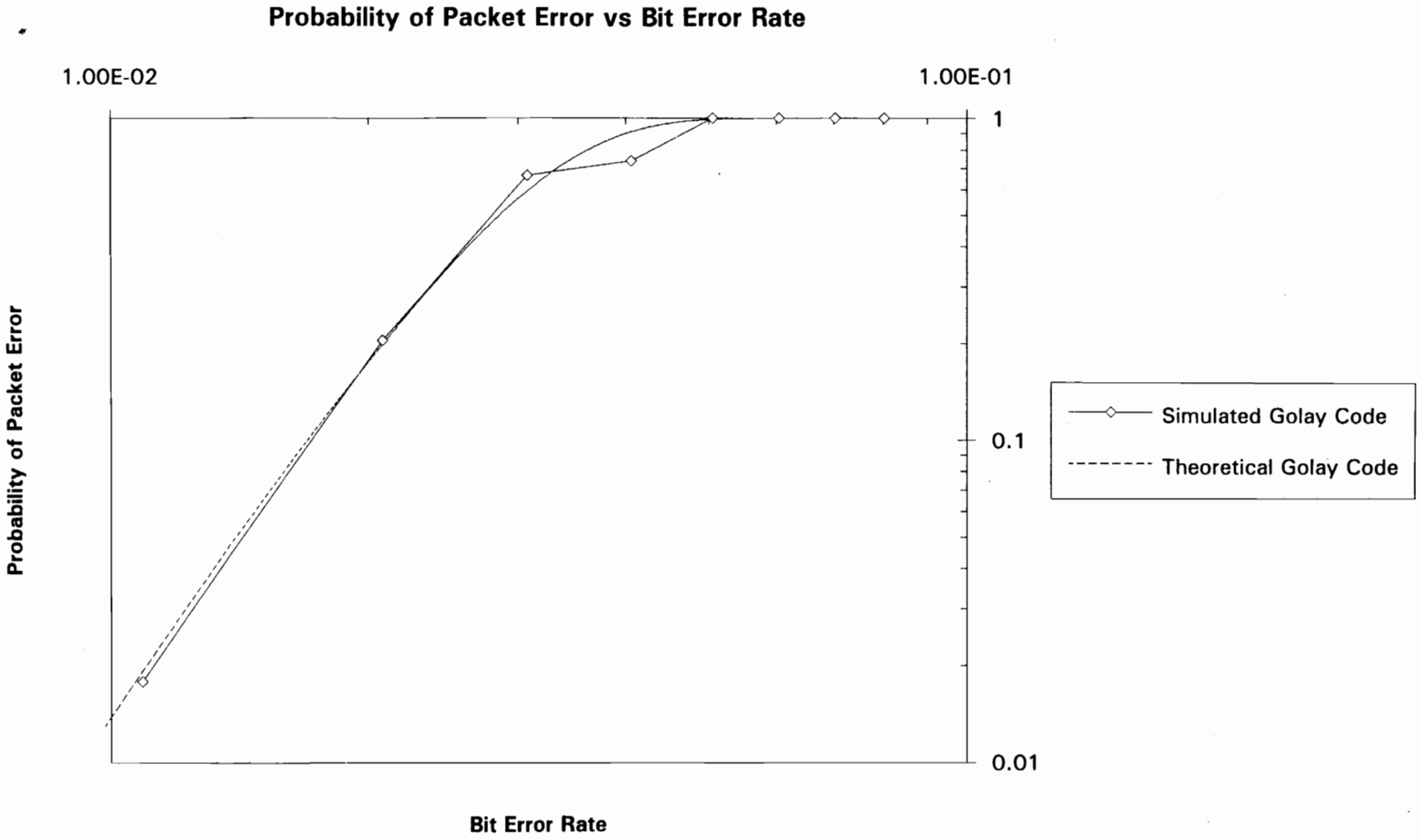

Figure 6-9 Plot of the probability of packet error vs bit error rate for the theoretical Golay code and the simulated Golay code

the theoretical code. This means that the algorithm used to simulate the Golay code worked and was able to correct all error patterns of up to 3 bits.

The packet length used for the encoding process was 2200 bits. This value includes the 2048 bits of the information field and 152 bits made up of the flag, address, control, PID, and FCS fields. The 152 bits which make up the preamble were deemed more important than the 2048 bits of the information field in Chapter 5. Therefore, some different codes were mixed, using various combinations for encoding the 152 bits and the 2048 bits. Figures 6.10 through 6.16 show the throughput results for the different systems tested.

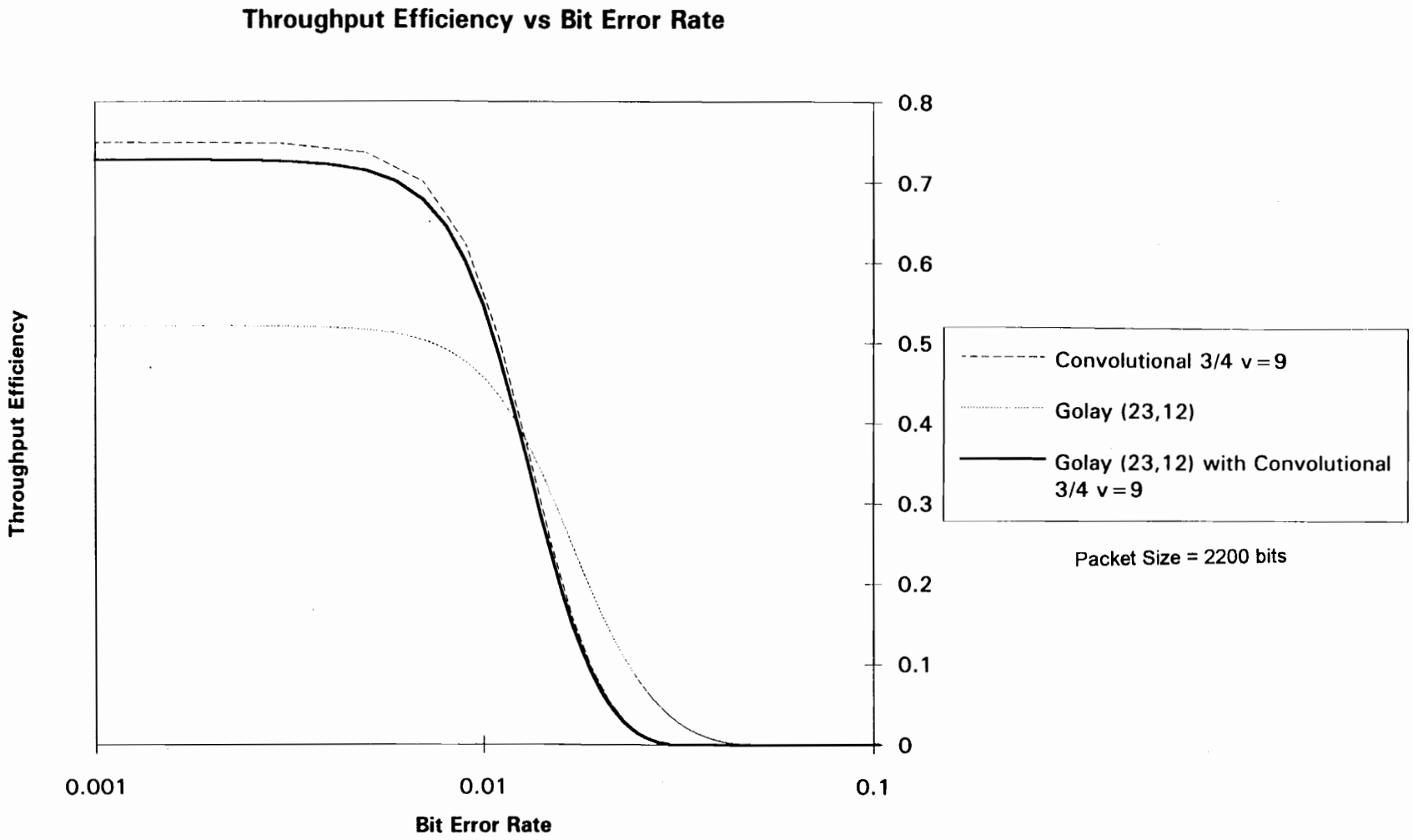

Figure 6-10 Plot of the throughput efficiency vs bit error rate for a packet which uses both the Golay (23,12) code and the Convoutional 3/4 rate v=9 codes. The system uss Go-Back-N ARQ. The convolutional code is used on the information field.

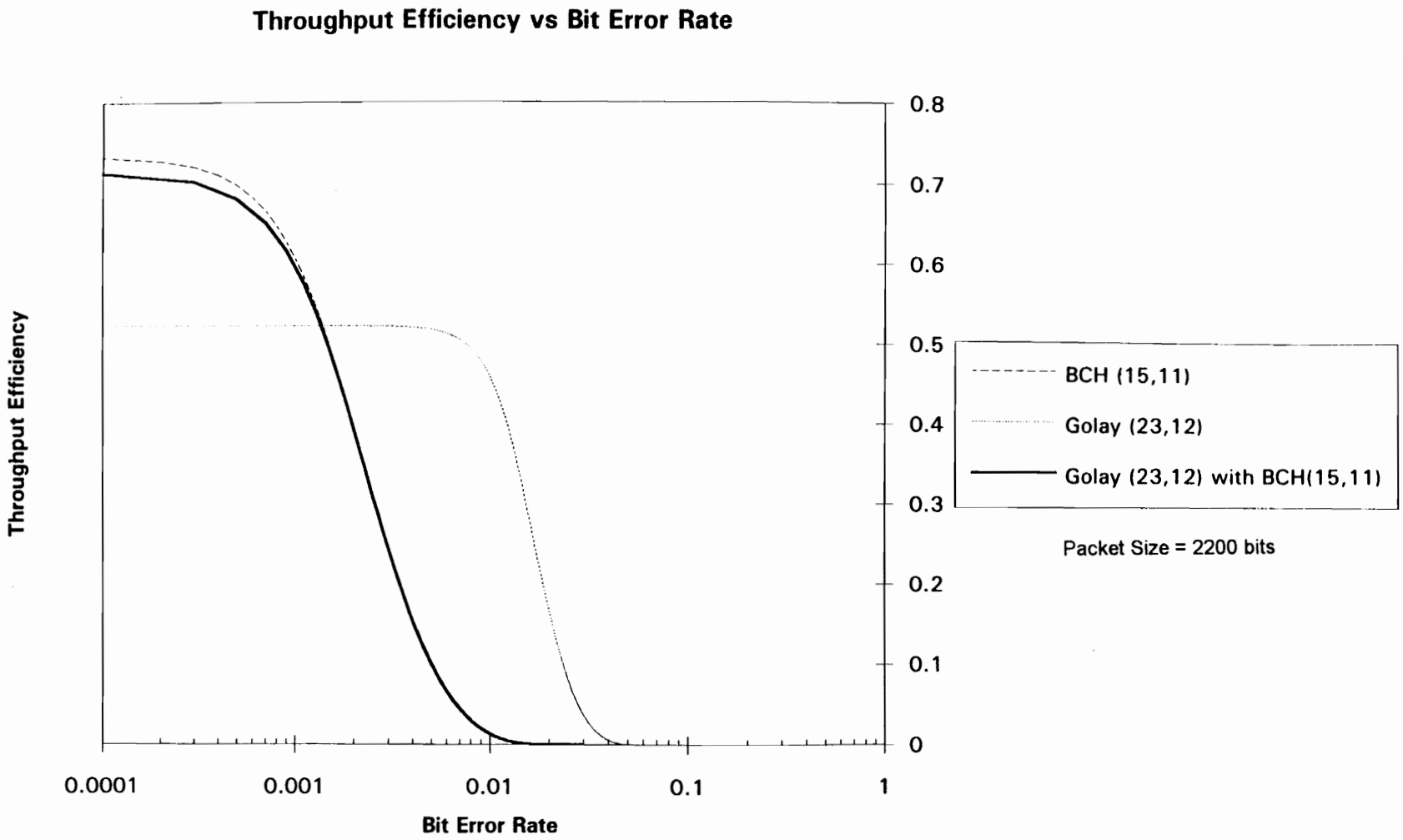

Figure 6-11 Plot of the throughput efficiency vs bit error rate for a packet which uses both the Golay (23,12) code and the BCH (15,11) codes. The system uss Go-Back-N ARQ. The BCH code is used on the information field.

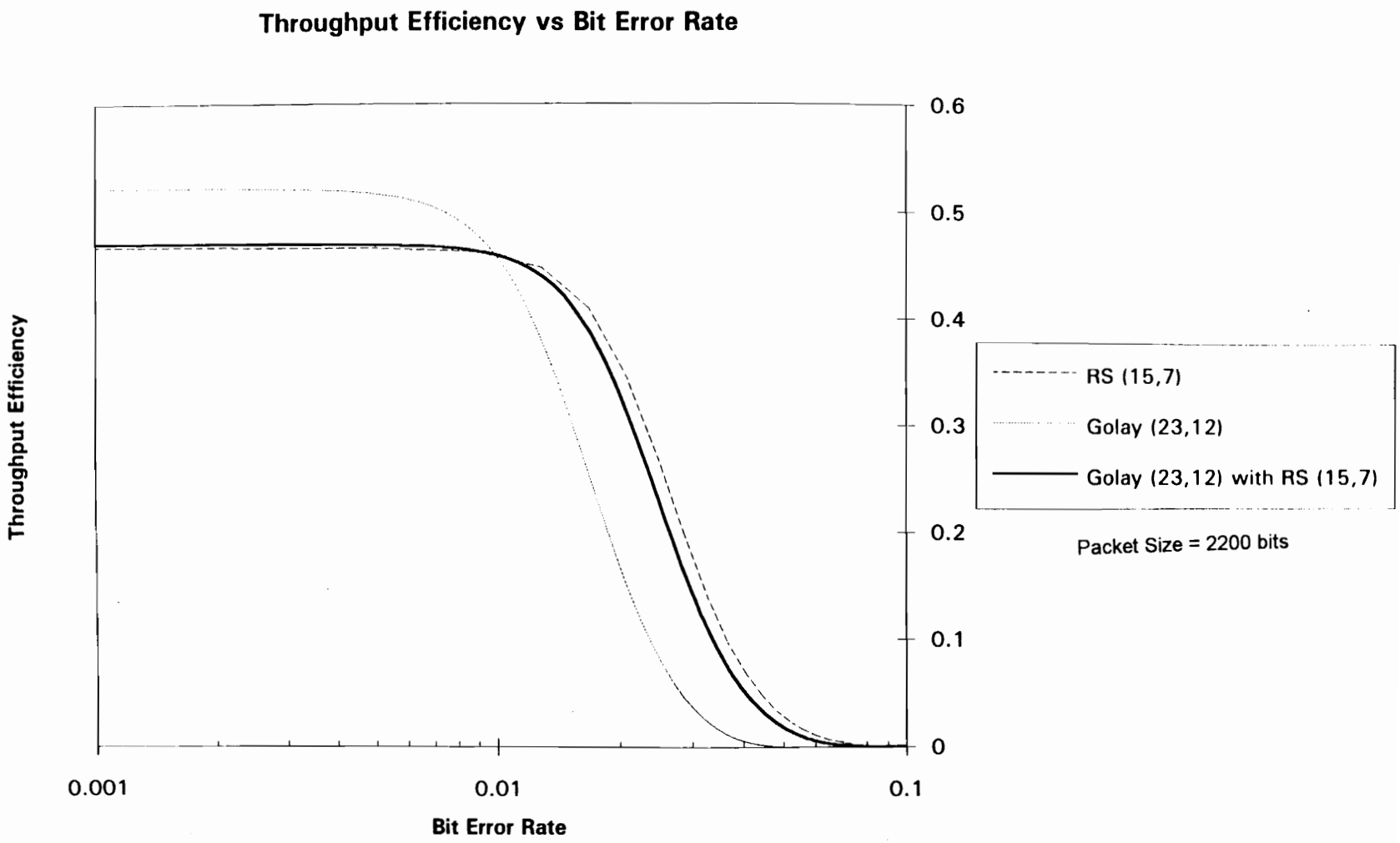

Figure 6-12 Plot of the throughput efficiency vs bit error rate for a packet which uses both the Golay (23,12) code and the RS (15,7) codes. The system uss Go-Back-N ARQ. The RS code is used on the information field.

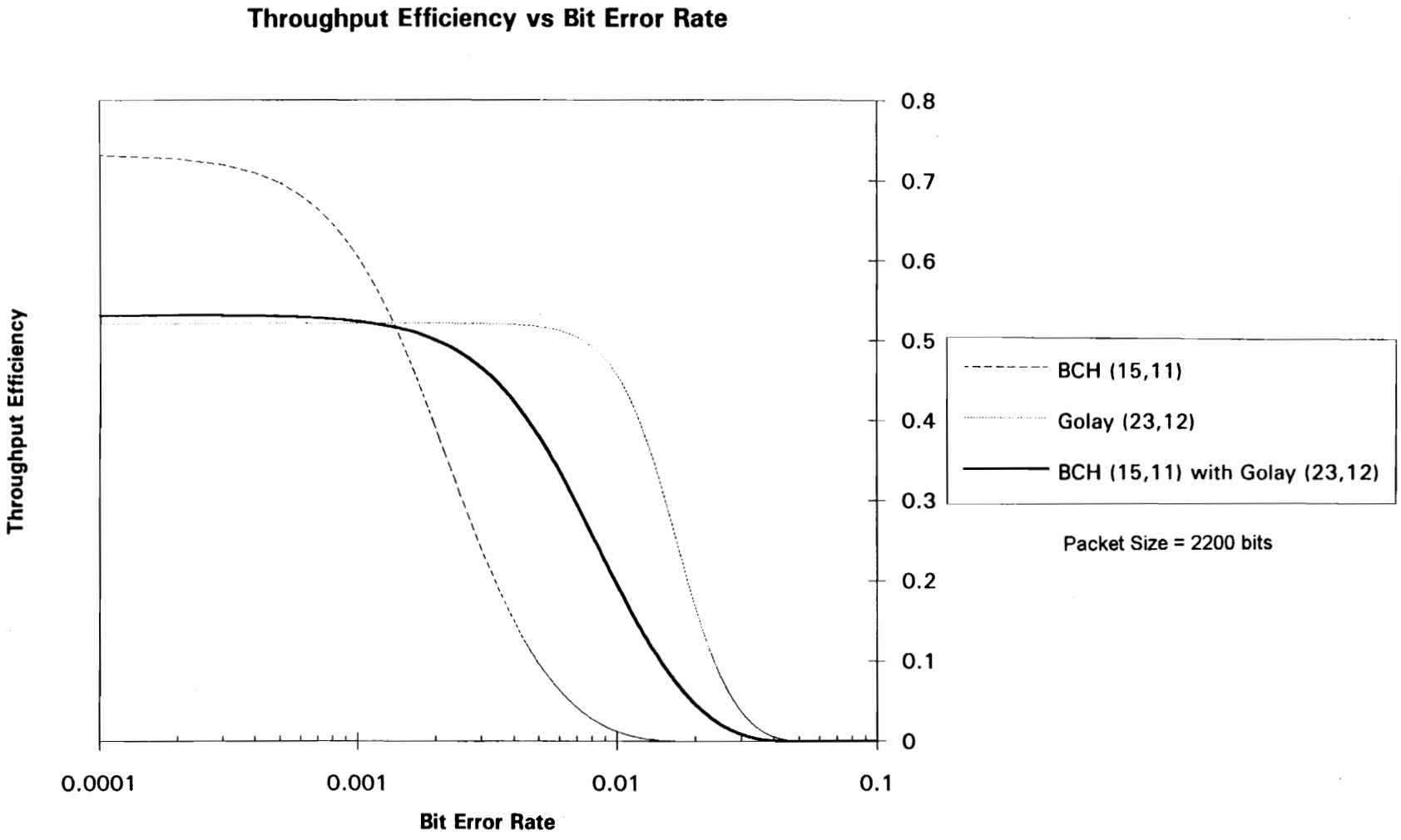

Figure 6-13 Plot of the throughput efficiency vs bit error rate for a packet which uses both the BCH (15,11) with the Golay (23,12) codes. The system uss Go-Back-N ARQ. The Golay code is used on the information field.

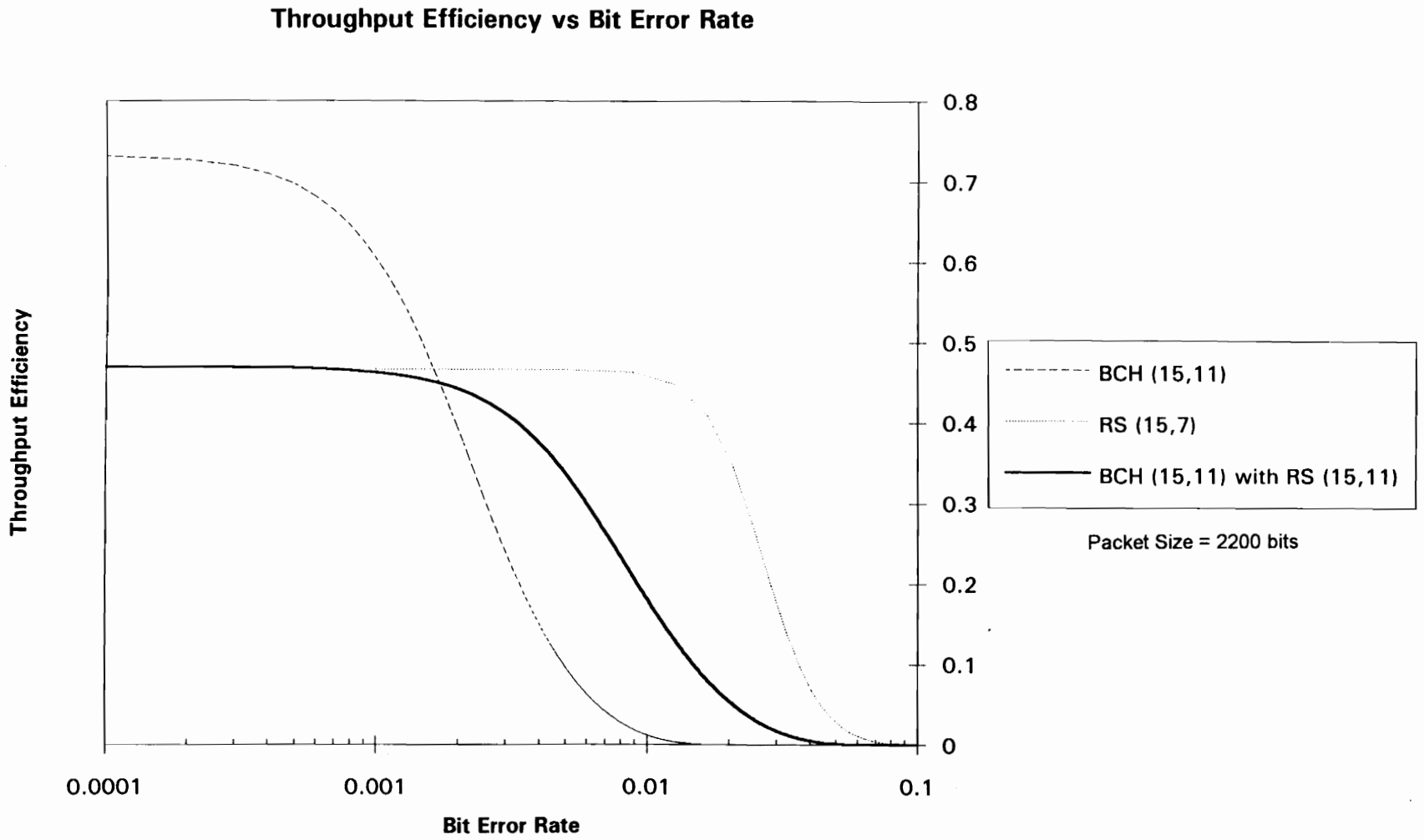

Figure 6-14 Plot of the throughput efficiency vs bit error rate for a packet which uses both the BCH (15,11) and the RS (15,11) codes. The system uss Go-Back-N ARQ. The RS code is used on the information field.

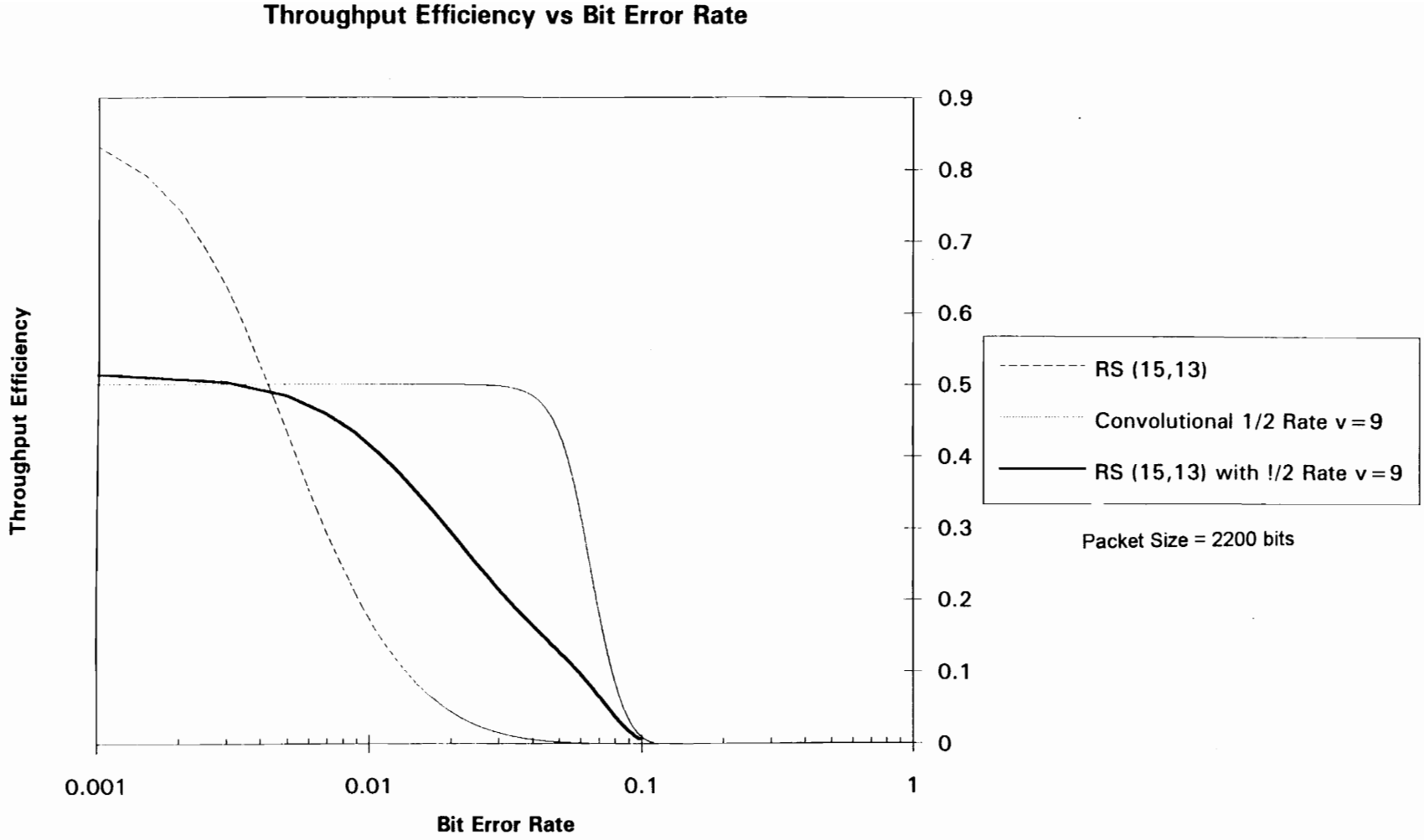

Figure 6-15 Plot of the throughput efficiency vs bit error rate for a packet which uses both the RS (15,13) code and the Convoutional 1/2 rate v=9 code. The system uses Go-Back-N ARQ. The convlutional code is used on the information field.

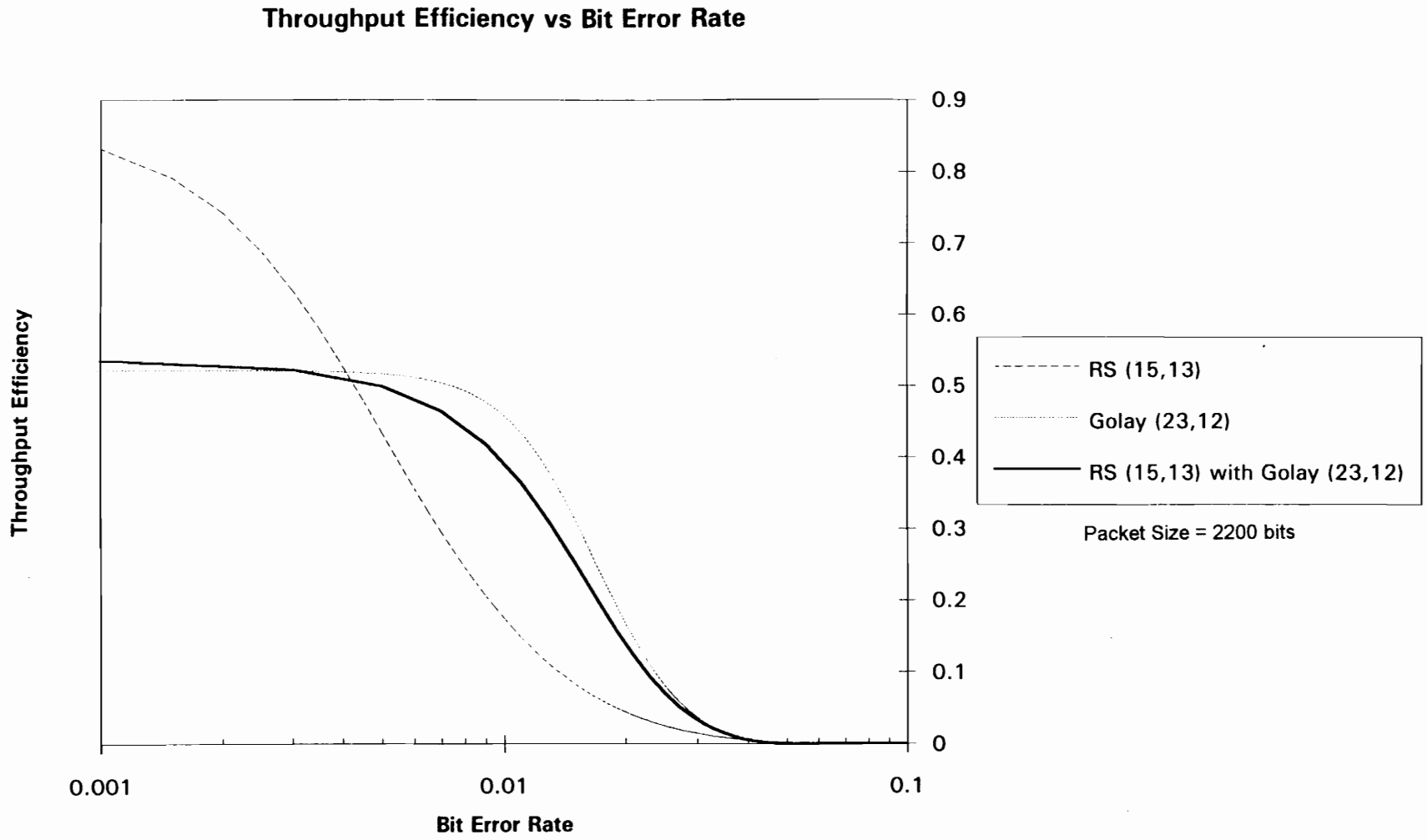

Figure 6-16 Plot of the throughput efficiency vs bit error rate for a packet which uses both the RS (15,13) code and the Golay (23,12) code. The system uss Go-Back-N ARQ. The Golay code is used on the informaition field.

When observing the performance of these systems it is seen that the plots of the probability of packet error all have something in common (Figures 5.19-5.25). The overall system performance is a combination of the two codes used to encode the packet. Looking at Figure 6.17 for example, we see that the probability of packet error for the code combination lies in between the probability of packet error of the BCH and Golay codes. In the linear region of the plot the slope of the probability of packet error equals the slope of the BCH code used to encode the preamble data(flag, address, control, PID, and FCS fields). In the non-linear region of the plot, the curve begins where the Golay code begins; the Golay code has the better performance of the two codes.

Figures 6.10 through 6.16 show the throughput results when code combining was used on the packet. As mentioned, the maximum value of throughput is given by the code rate of the code used to encode the packet. The code rate is defined as the ratio of the number of information bits to redundant bits, so when using a combination of codes to encode a packet, the code rate will be a combination of the combined codes. The maximum throughput value when a combination of two codes is used is given by

$$\eta_{\max} = \frac{k_{preamble} R_{preamble} + k_{info} R_{info}}{k_{tot}} \tag{6.3}$$

$k_{preamble}$	number of bits in the preamble
k_{info}	number of bits in the information field
k_{tot}	number of bits in the packet
$R_{preamble}$	code rate of the code used to encode the preamble
R_{info}	code rate of the code used to encode the information field

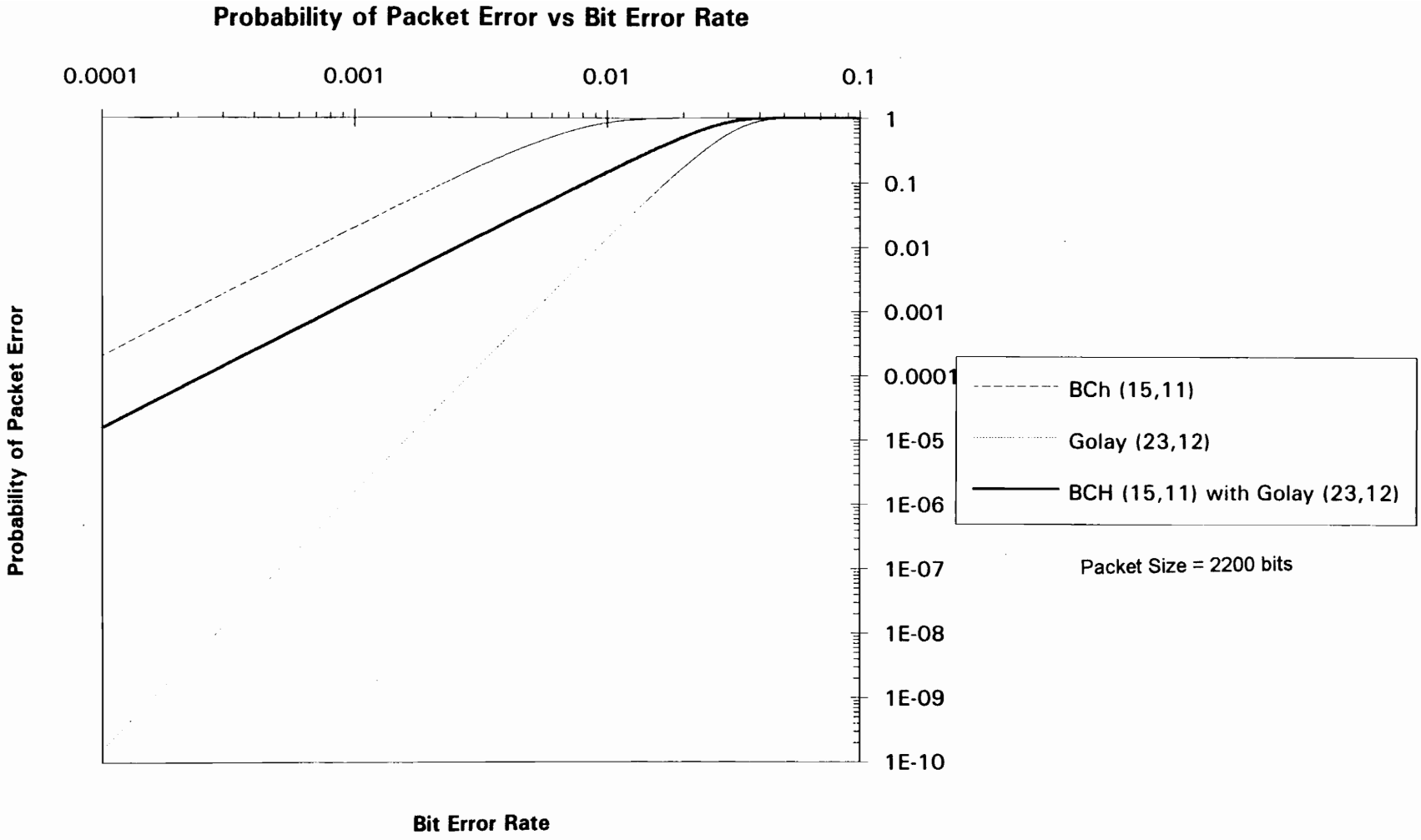

Figure 6-17 Plot of the probability of packet error vs bit error rate for a packet using both the BCH (15,11) and the Golay (23,12) codes. The Golay code is used on the information field.

To illustrate Equation 6.3, when the Golay code is used to encode the preamble and the 3/4 rate convolutional code is used on the information field, the maximum throughput is given by

$$\eta_{max} = \frac{152 \cdot (12/23) + 2048 \cdot (3/4)}{2200} = 0.734 \qquad (6.4)$$

When code combining is used, a common pattern emerges in the system throughput. At low bit error rates, when the throughput is constant for each code used in the combination, the overall throughput is constant. As the bit error rate increases and the throughput begins to fall off for the individual codes, the overall throughput begins to decline until it reaches zero. For each of the combinations used, the overall throughput reaches zero at or just below the bit rate when the throughput of the code used to encode the information field equals zero.

After mixing of codes was attempted, concatenation was used to see what, if any, effect this had on the system performance. Figure 6.18 shows the throughput for the concatenated code which uses 3/4 rate v=9 convolutional code over the (23,12) Golay code. The probability of packet error vs. bit error rate for this concatenated code is shown in Figure 6.19. It is clear from Figure 6.19 that by using concatenation, there is a significant increase in performance. However, this performance increase is greater as the bit error rate decreases. As the bit error rate increases, the performance increase is less and less.

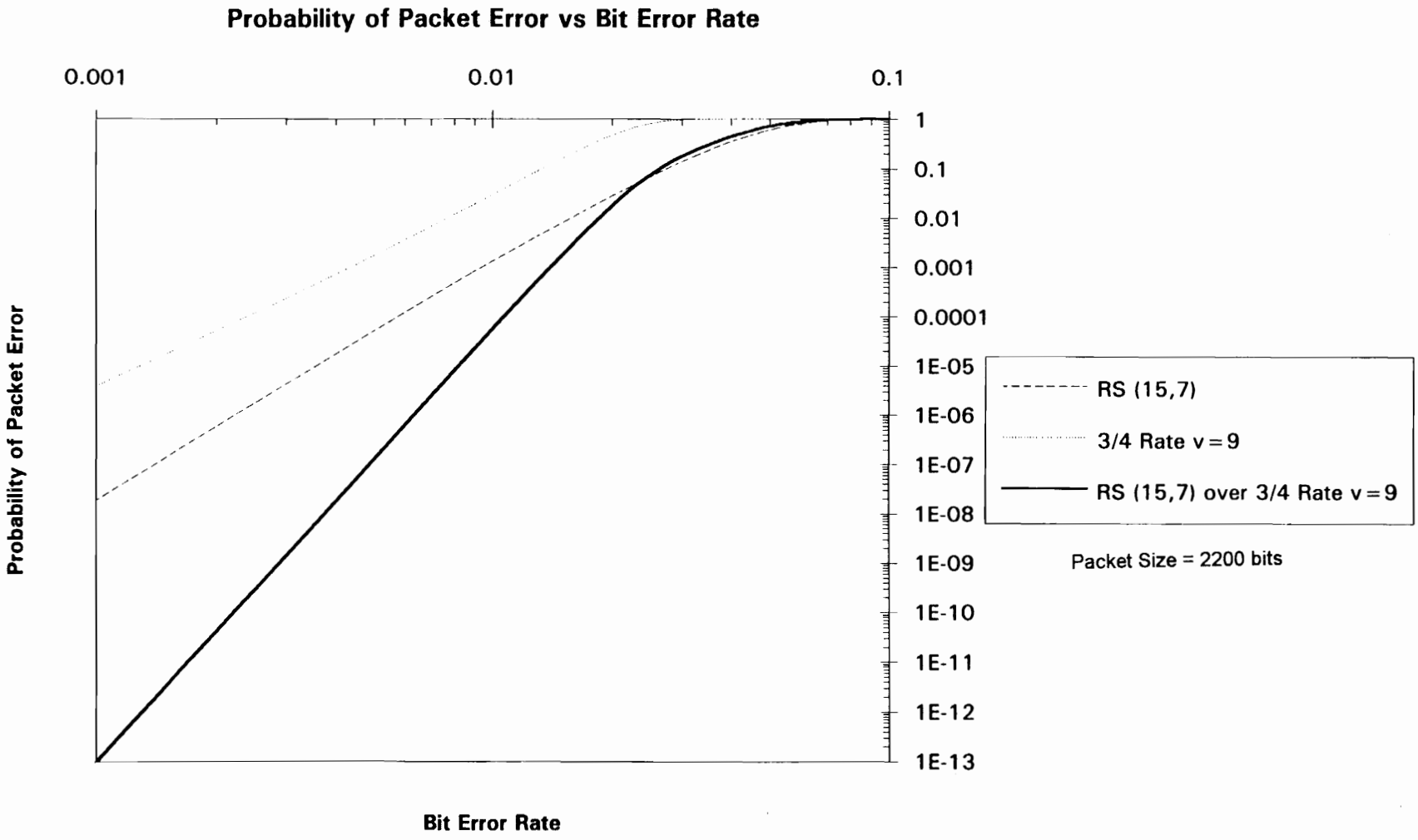

Figure 6-18 Plot of the probability of packet error vs bit error rate for a concatenated packet with RS (15,7) over 3/4 rate convolutional v=9 code .

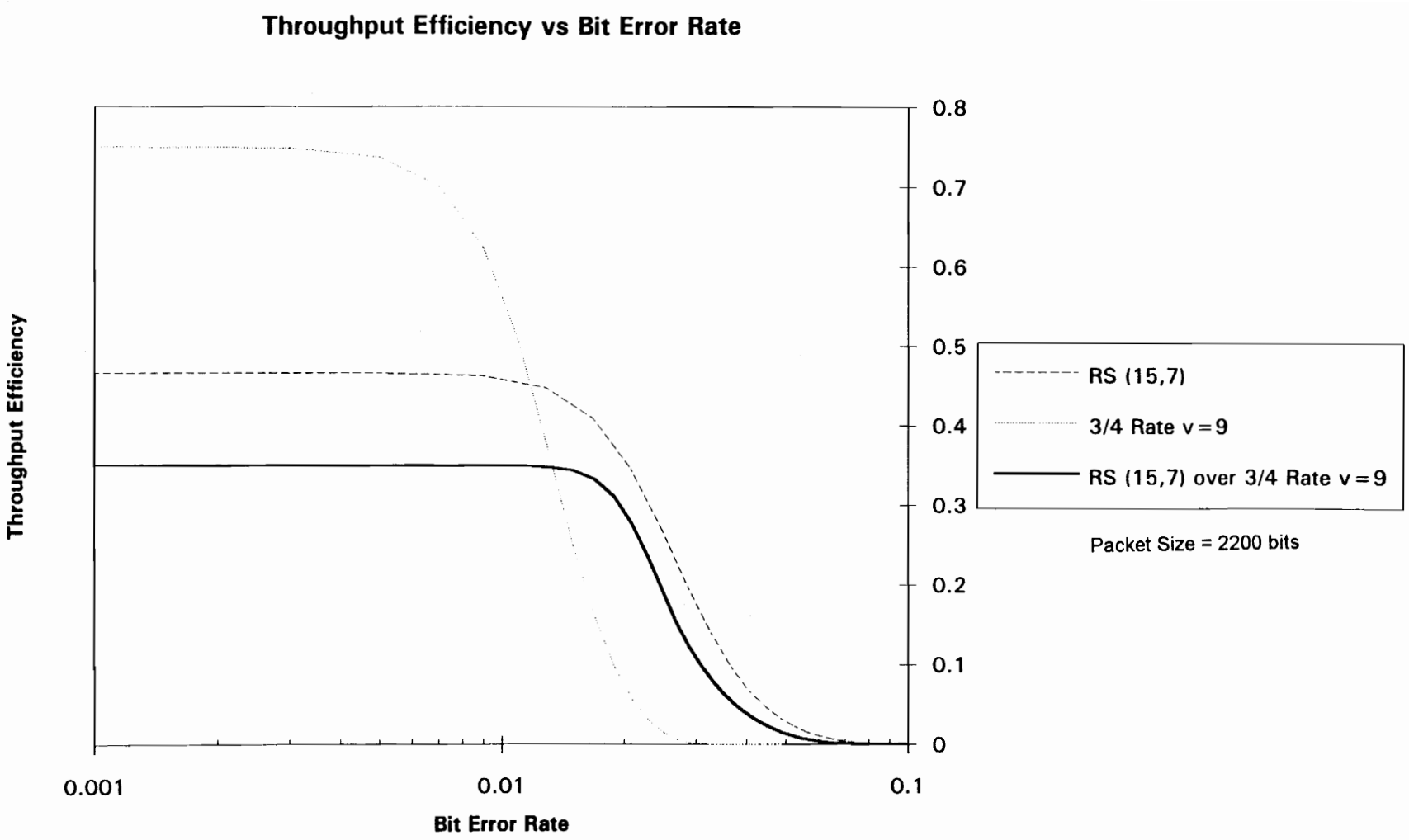

Figure 6-19 Plot of the throughput efficiency vs bit error rate for a concatenated packet with RS (15,7) over the 3/4 rate v=9 convolutional code.

Observing the effect upon the system throughput shows that concatenation severely decreases the throughput. When concatenation is used, the maximum throughput efficiency is given by

$$\eta_{max} = R_{inner} \cdot R_{outer} \tag{6.5}$$

where R_{inner} is the code rate of the inner code and R_{outer} is the code rate of the outer code used. If, for example, two 1/2 rate codes are used, the maximum throughput of the concatenated code is only 1/4. This means that there is a 1 to 4 ratio of information bits being passed through the system to the number of redundant bits.

In the cases tested in Chapter 5, it is seen that the use of concatenation in most cases does not improve throughput performance. The reason for this is related to the performance of the probability of packet error. The improvement in the packet error probability, which affects the throughput, occurs at bit error rates where the throughput is still at its peak value. Any further decrease in the packet error probability will not have any effect on the throughput. At lower bit error rates where the throughput begins falling off, concatenation does not improve the probability of packet error sufficiently to alter the throughput.

6.1.1 Golay Code

In order to study the Golay code an encoder and decoder were computer simulated. The algorithm first used for encoding and decoding is discussed in Chapter 5. However, when this algorithm was implemented, it was found not to work as presented. The decoding algorithm presented in [9] (pp. 184-185)

states in step 2 that if the weight of the syndrome ever falls to 3 or less, then this syndrome matches the error pattern and correction can be made. In order to test the algorithm, a sample data stream was encoded and then 1 bit was corrupted. This code word was passed through the decoder, checking all the syndromes to see if their weight ever fell to 3 or less. As it turned out, there were several syndromes which had a Hamming weight of 3 or less. The first one which occurred did not match the correct error pattern so all subsequent syndromes were checked. It was found that the last syndrome was the correct error pattern from which correction could be made.

In order to observe what happened when multiple errors occurred, the same code word used for the first simulation was used, but this time two bits were corrupted. When this was done the same results were obtained as those for the case when only one bit was corrupted. There were several syndromes with Hamming weight less that or equal to 3. It was therefore obvious, that in the form the algorithm was presented some manipulation had to be performed before it could be implemented.

It was observed that whenever any random single bit was changed the decoder always produced a syndrome which matched the correct error pattern. Therefore, the decoder I implemented used a variation of the one stated in Chapter 5 found in [9] and [15]. The algorithm proposed is as follows:

Step 1: Compute the Syndrome of the received word **r**

Step 2: Find the syndrome $\mathbf{s}^{(i)}$ of the cyclic shifts $\mathbf{r}^{i}$ of the received word (to the right) for I=0,1,...,22. Look for any syndromes with Hamming weight equal to 1. If any occur, try all of

these syndromes as error patterns and decode the received word as the ith cyclic shift of $\mathbf{r}^{(i)}$ - $\mathbf{s}^{(i)}$ to the left for each syndrome. Then recompute the syndrome looking for one which is 0. If a syndrome is found which has Hamming weight 0, then the received word has been decoded. If no syndrome of weight 1 is found, then the received word contains multiple errors.

Step 3: If all the syndromes in step 2 have Hamming weight larger than 2, change one bit of the received word, the bit r_0. Keep this bit changed and begin systematically changing one other bit at a time starting with r_1. After each change compute the syndrome and see if it is 0. If it is, then this is the correct code word. If the syndrome is not 0, change r_1 back to its original value and change r_2 repeating the syndrome calculation. Repeat this process changing all bits up to r_{23} looking for a syndrome of weight 0. If none is found, r_0 is correct. Reset it to its original value and start step 3 over first changing r_1 and r_2. Keep r_1 changed while altering the next 21 bits, one at a time looking for a syndrome of weight 0. Repeat this process until all combinations of altering 2 bits have been attempted.

Step 4: If all the syndromes in step 3 have Hamming weight larger than 0, then there must be more than two errors in the received word. Change r_0, r_1, and r_2 and compute the syndrome checking to see if it is 0. If it is then this is the correct code word. If not, then change r_2 back to its original value and begin changing one bit at a time until all bits up to and including r_{23} have been changed and the syndrome checked. Repeat the process as in step 3, checking the syndrome for all possible combinations of 3 altered bits looking for a syndrome which is 0. When one is found, the decoding is finished.

The algorithm used above is presented as computer code in Appendix A. This algorithm is guaranteed to correct any combination of three errors. Note that this systematic checking can be used to correct up to d_{min} -1 errors in the received code word, but this would make for a slow decoding process. In order to correct a single error, on average, the algorithm presented must compute 12

syndromes until one is found which is 0. The number of computations dramatically increases when multiple errors occur. If there are two errors in a received code word, an average of 116 syndromes must be calculated and this increases to an average of 886 syndromes if there are three errors in the received code word. Thus it can be seen that although this algorithm is guaranteed to correct all combinations of up to three errors, as the bit error rate increases the decoding time can increase significantly. Figure 6.9 shows the performance of this decoder compared to the theoretical Golay code results.

6.1.2 Throughput Performance Selection

When reviewing the throughput performance of the various codes and systems used, it is seen that there is a wide range of results. In order to choose a system based upon the throughput, one code from each group will be chosen, and then all of these compared.

For the BCH codes tested, the (255,123) has the best performance. Although this code does not have the largest throughput, it does maintain its peak value longer than any of the other BCH codes shown.

When deciding upon a convolutional code, the 1/2 rate v=9 convolutional code performs the best. The 1/3 rate convolutional codes maintain their maximum throughput value only slightly longer than the 1/2 rate convolutional codes, but the 1/2 rate codes pass more information through the system in a given period of time.

Between the different Reed-Solomon codes used, the (255,187) code is the best to incorporate with the ARQ. This code has a maximum throughput of 0.733 and it maintains this value over a large range of bit error rates.

When combinations of codes were used, the system which showed good performance was the one which used the Golay code to encode the preamble and the (15,7) Reed-Solomon code to encode the information field. This system has a maximum throughput near 0.5 and maintains its peak throughput up to a bit error rate of 0.01 while the next closest system throughput begins dropping off at a bit error rate of 0.06.

The concatenated system which shows the best overall performance is the one which uses the 3/4 rate v=9 convolutional code for the inner code and RS (15,13) for the outer code. This system has a good throughput (0.65). Some of the other systems maintain their maximum throughput longer, but their maximum throughput value is too small(0.4 or less).

Figure 6.20 shows a plot comparing the throughput vs. bit error rate for the codes which were chosen above. From this graph, based on the throughput, the 1/2 rate v=9 convolutional code and the (255,187) Reed-Solomon code would be the best to encode the AX.25 radio packet. Both of these codes have good maximum throughput values and they each maintain their peak value over a large range of bit error rates.

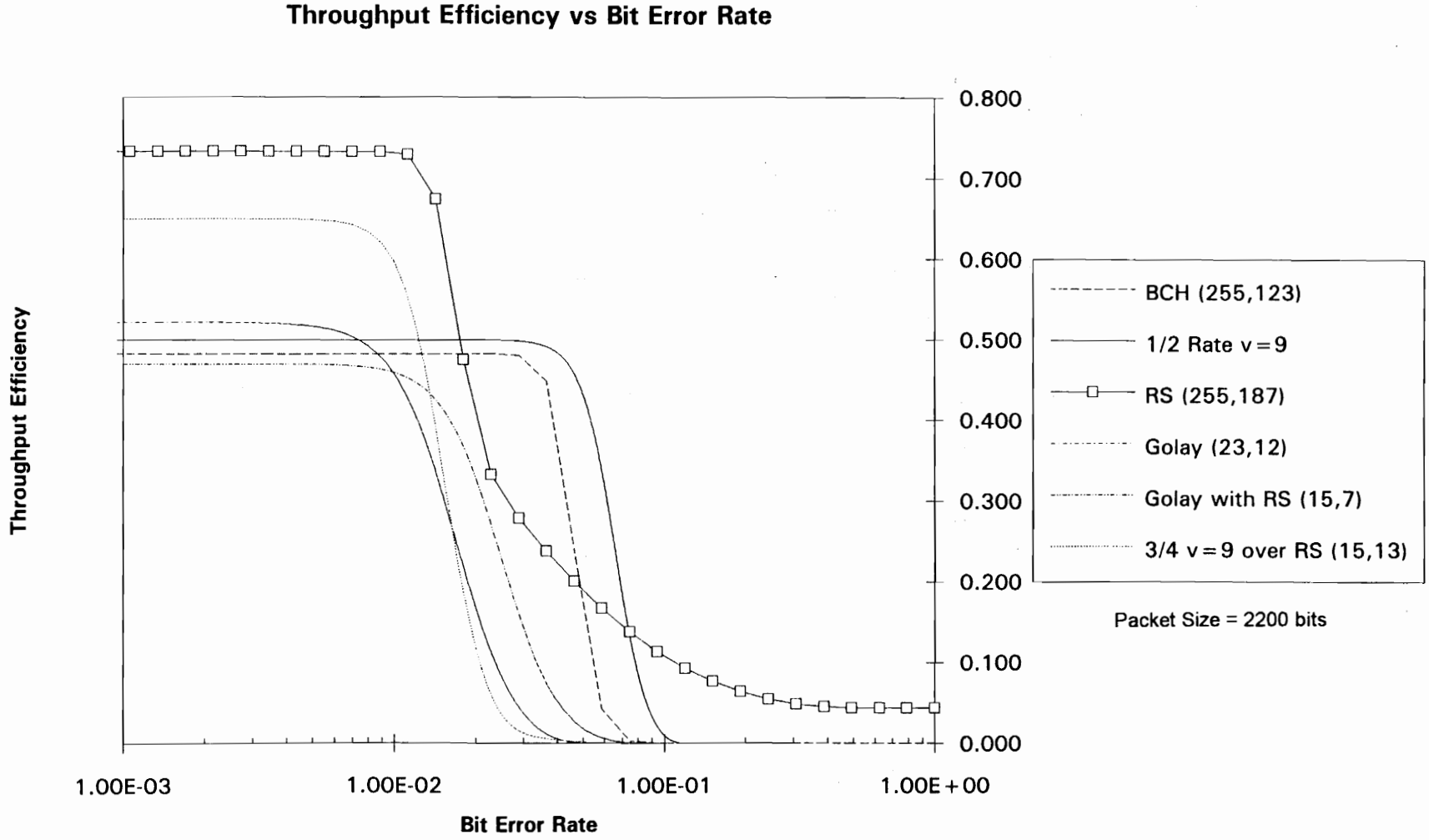

Figure 6-20 Plot of the throughput efficiency vs bit error rate for various codes

6.2 System Complexity

In order to choose a possible code to use with the AX.25 packet radio, the system complexity is an important consideration. If the code, or coding scheme used to encode the packet becomes too complex, the resulting system increases in price and decoding time can become too large. When coding is used, the main complexity of the system is in the decoding process. It is fairly easy to encode data, but in many cases it can be quite difficult to find a good decoder. Therefore, the complexity of the decoder will be a main consideration when deciding upon the overall system complexity.

Implementation of the BCH codes can be achieved either by digital hardware or by software. In the decoding process for BCH codes there are three main steps. The first step in decoding a BCH code which is capable of correcting t-errors in a code word is to compute 2t syndrome components. For software, the 2t components can be computed with (n - 1)t additions and nt multiplications. If hardware implementation is used, feedback shift registers can be used to compute the syndrome components. This process involves at the most t feedback shift registers, each consisting of at most m stages. After the syndrome has been calculated, the next step is to find the error location polynomial. With software implementation, this requires $2t^2$ additions and $2t^2$ multiplications. The same total of additions and multiplications are needed for hardware implementation. The third and final step involves computation of error

location numbers and error correction. For the worst case, nt additions and nt multiplications are required for this stage.[1]

The Golay code encoding can be accomplished by an 11-stage shift register with feedback connections. This involves shifting an 11 bit data stream into the encoder and receiving a 23 bit coded word. For decoding the Golay code, the Kasami decoder or a systematic search decoder are commonly employed. Both of these techniques are capable of decoding up to t=3 errors in the received code word. The Kasami decoder requires 46 cyclic shifts to decode the received word with 50 logic operations that correspond to each of the cyclic shifts. The systematic search decoder which was presented earlier requires an average of 886 syndrome calculations if three errors occur in the received word.

Encoders and decoders for the Reed-Solomon codes are similar to the ones used for the BCH codes except that all arithmetic is performed modulo-2^m. Like the previous codes, encoding can be accomplished with shift registers, but it is more complicated due to the fact that the system is non-binary. The decoding process involves the same three steps used for decoding the BCH codes with one additional step. In summary, the first step involves calculating 2t syndrome components. The second step is to calculate the error locator polynomial which requires $2t^2$ additions and $2t^2$ multiplications. For the third step, nt additions and nt multiplications are needed to solve for the roots of the error locator polynomial. The fourth and final step involves calculation of the t

[1] Lin and Costello, p 167-176.

error values. Calculating these error values requires $3t^2/2$ multiplications and $t^2 + t^3/2$ additions.[2]

A convolutional code is encoded using linear feedforward shift registers and modulo-2 adders. The number of modulo-2 adders required is equal to the number of outputs. The constraint length of a convolutional code is a measure of the memory within the encoder so the larger the constraint length, the more memory an encoder has. As the constraint length of the encoder increases, the system complexity increases because as the constraint length increases, more memory is required along with more connections to the modulo-2 adders.

The decoding process for convolutional codes can be quite complex. The function of the decoder is to estimate the encoder input information sequence using a rule or method which results in the minimum possible number of errors being delivered to the user. One of the more popular methods for decoding a convolutional code is the Viterbi method. The complexity of Viterbi decoding is directly proportional to the number of states in the trellis.[3] The number of states grows exponentially with constraint length and decoders become impractical for large constraint lengths. The complexity is proportional to 2^K, the number of encoder states. The number of encoder states is an important number because it represents several different factors. First, the decoder memory is proportional to 2^K. Also, 2^K comparisons must be performed per time unit which means that decoding time is also proportional to 2^K.

---

[2] Lin and Costello, p167-176.
[3] Ziemer and Peterson, p. 450.

Some of the coding schemes employed used a combination of two codes to encode the packet. In the first process, one code was used to encode the preamble which consisted of the flag, address, control, PID, and FCS fields while a second code was used to encode the information field. The complexity of this process is governed by the complexity of the different codes used in the packet. Before the packet is assembled, the appropriate fields can be fed into different encoders. The receiver must have knowledge of the frame code structure so that it can properly disassemble the packet and send the information field and the preamble to the proper decoders. This type of system does not add any complexity to the overall decoding process except that the receiver must know which type of code has been used to encode the different fields and be able to properly disassemble the packet.

The second code combination which was used was concatenation. In this process, the packet is first fed into one encoder. At the output, the encoded packet is fed directly into another encoder, thus producing a packet which has been encoded twice. As in the previous case, the system complexity depends upon the different codes used. It will take longer to decode the packet with this type of encoding because the packet has to be decoded twice, but this only adds to the system delay time, not the complexity.

Comparing the complexity of the codes tested, it was found that the BCH codes were the simplest, followed by the Golay, and then the Reed-Solomon and Convolutional codes. When code combining or concatenation was used,

the complexity was determined by the codes used within the system. Employing concatenation will cause a delay in the decoding process compared to using only a single code, but the decoding complexity should not be affected.

6.3 Code Selection

In summary, when deciding upon a code to use with the AX.25 packet radio system, several different factors were considered such as throughput performance and system complexity. After comparing some different codes and coding schemes, it was determined that the best throughput results occurred with convolutional codes and Reed-Solomon codes. The decoding process for these codes can be quite complex depending upon the constraint length or block length of the code, but algorithms and hardware are readily accessible for decoding either of these. The convolutional code which should be used is the 1/2 rate v=9 code. This code has a maximum throughput of 0.5 and maintains its throughput over a large range of bit error rates. The Reed-Solomon code which best met the criteria was the (255,187) code. It has a maximum throughput of 0.733 and also maintains its throughput over a large range of bit error rates.

Convolutional coding and Viterbi decoding can be be accomplished using a single integrated circuit, so the addition of convolutional coding to the packet radio system can be easily accomplished. The same is true for Reed-Solomon

coding and decoding. Because only one IC chip is needed at a single station, the resulting price should be affordable for the average user.

Implementation of packet coding is relatively simple. After the packet has been assembled by a transmitting station, it should be fed directly into the encoder. The resulting output data stream is the encoded packet which can be sent to a modulator and transmitted. Upon reception, after the packet has been demodulated, it can be fed into a decoder. The output of the decoder is the decoded packet which is ready to be disassembled.

If concatenation of the packet is to be used, a process similar to the one presented above can be used. The difference is that two IC chips will be needed at each station. Once the packet has been coded with an outer code, it can be input into a second encoder. The output data stream is the concatenated packet which can be transmitted. When the packet is received, it will first enter the decoder for the inner code. The output of this decoder will enter the decoder for the outer code. The output of the second decoder is the packet which can be disassembled. Concatenation will take longer than the use of a single code because the data stream is being encoded and decoded twice.

The encoding/decoding is slightly more complex when combinations of codes are used. In the encoding process, the preamble must be sent to one encoder, while the information field is sent to another encoder. The output of the two encoders can then be put together and transmitted. The receiver must know how the two coded streams are put together so it can separate the received data

stream and send the information field and the preamble to the correct decoders. The data leaving the two decoders can be rejoined and sent to the packet disassembler. For this process, two different encoding and decoding chips are needed because two different codes are being used.

When comparing the cost of the different schemes, it is seen that using a single code will be the cheapest method. Concatenation and code combining both require two encoders and decoders and the transmitting and receiving stations, while the use of a single code requires only one. The price of the encoding and decoding IC chips will depend upon the code they are implementing. If the code is a convolutional code, the larger the constraint length, the larger the cost because more gates and memory are required as the constraint length increases.

If Reed-Solomon codes are used, the number of gates in the encoder depend upon the error correcting capability of the code. A total of 2t memory devices, 2t adders, and 2t multipliers are needed for a t error correcting Reed-Solomon encoder. In the decoder, the syndrome computation circuit requires m adders, m storage devices, and m multipliers if the code is over Galois Field(2^m). So the price of a Reed-Solomon codec is dependent upon its error correction ability and the Galois Field over which it is operated.

CHAPTER 7

IMPROVING SYSTEM PERFORMANCE

Chapter 5 presented a comprehensive review of coding techniques which dramatically improve the performance of AX.25 packet radio. All of the systems in Chapter 5 used the go-back-N ARQ system which is discussed in detail in Chapter 3. There are some systems, namely Selective Repeat ARQ and a combination of Selective Repeat (SR) and Go-Back-N (GBN), which have better throughput performance than the Go-Back-N ARQ. This chapter shows the improvement upon the performance of some of the systems presented in Chapter 5 when different ARQ methods are used.

7.1 Selective Repeat ARQ

7.1.1 Throughput Efficiency

The throughput efficiency for the SR ARQ is given by

$$\eta_{SR} = \left(1 - P_{pe}\right)\left(\frac{k}{n}\right) \tag{7.1}$$

where k/n is the rate of the (n,k) code used in the system and P_{pe} is the probability of packet error for the system. Figures 7.1 through 7.4 show the improvement in the throughput for some of the different systems which were

presented in Chapter 5 when Selective Repeat ARQ is used instead of the Go-Back-N method.

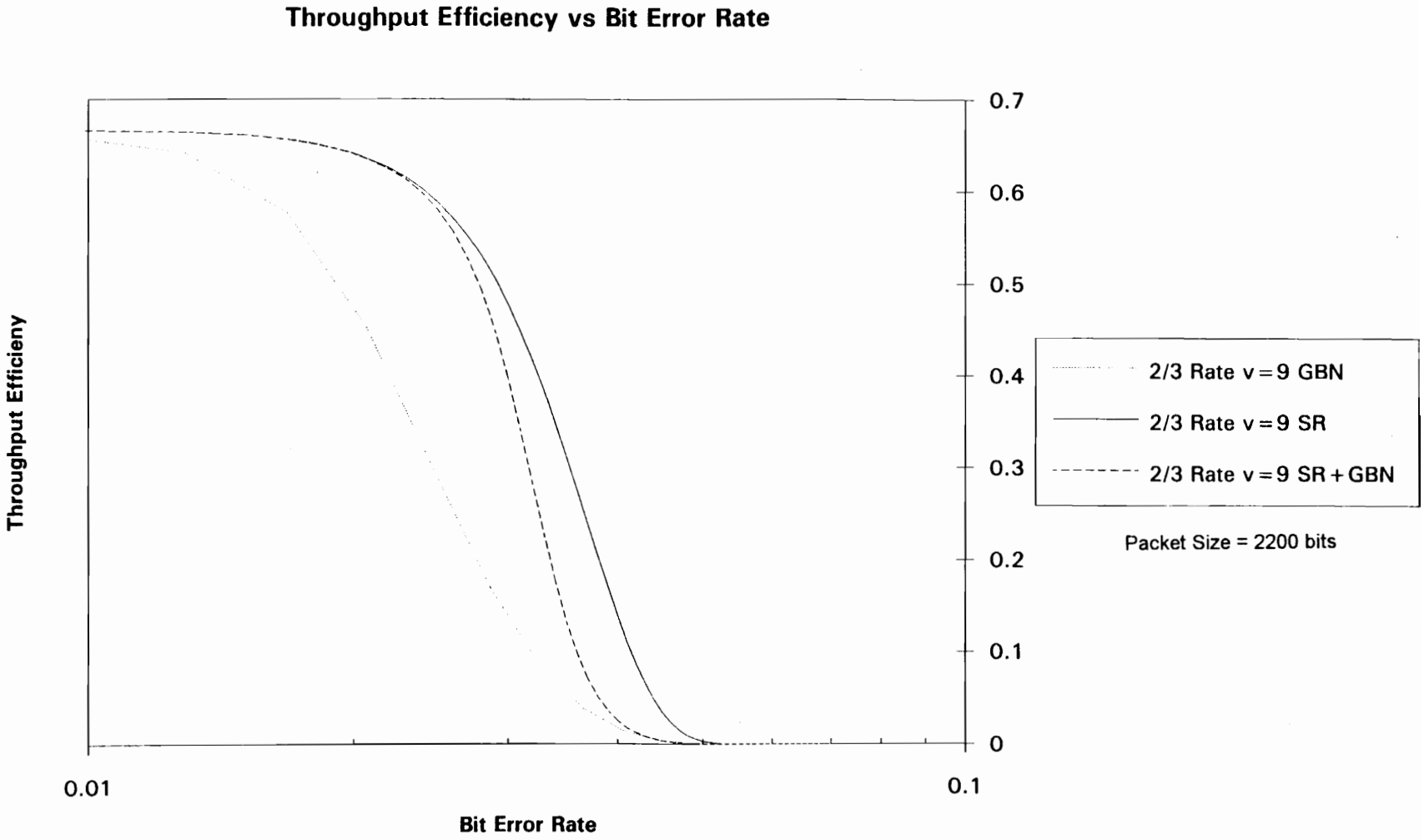

Figure 7-1 Plot of the throughput efficiency vs bit error rate for various ARQ schemes

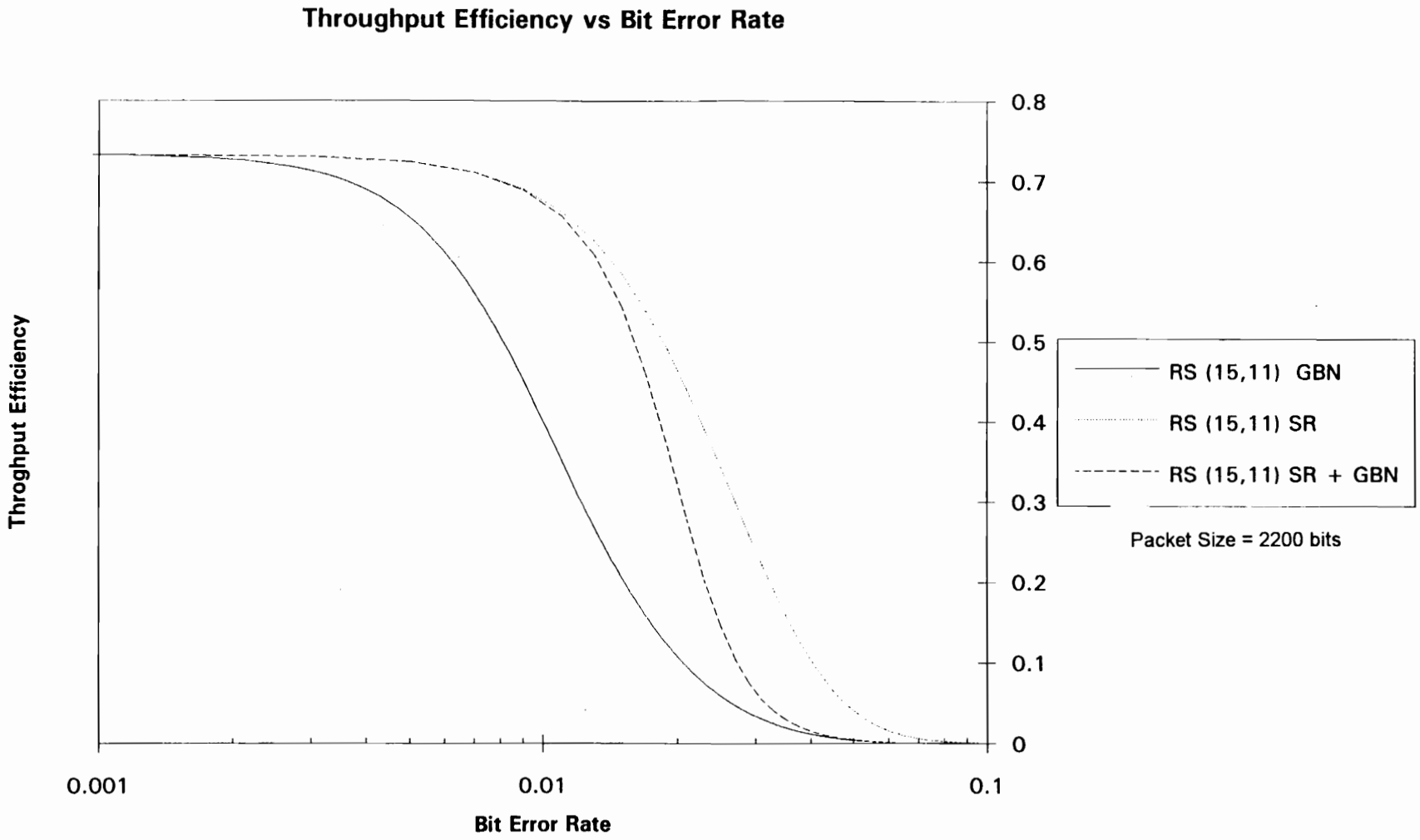

Figure 7-2 Plot of the throughput efficiency vs bit error rate for various ARQ schemes

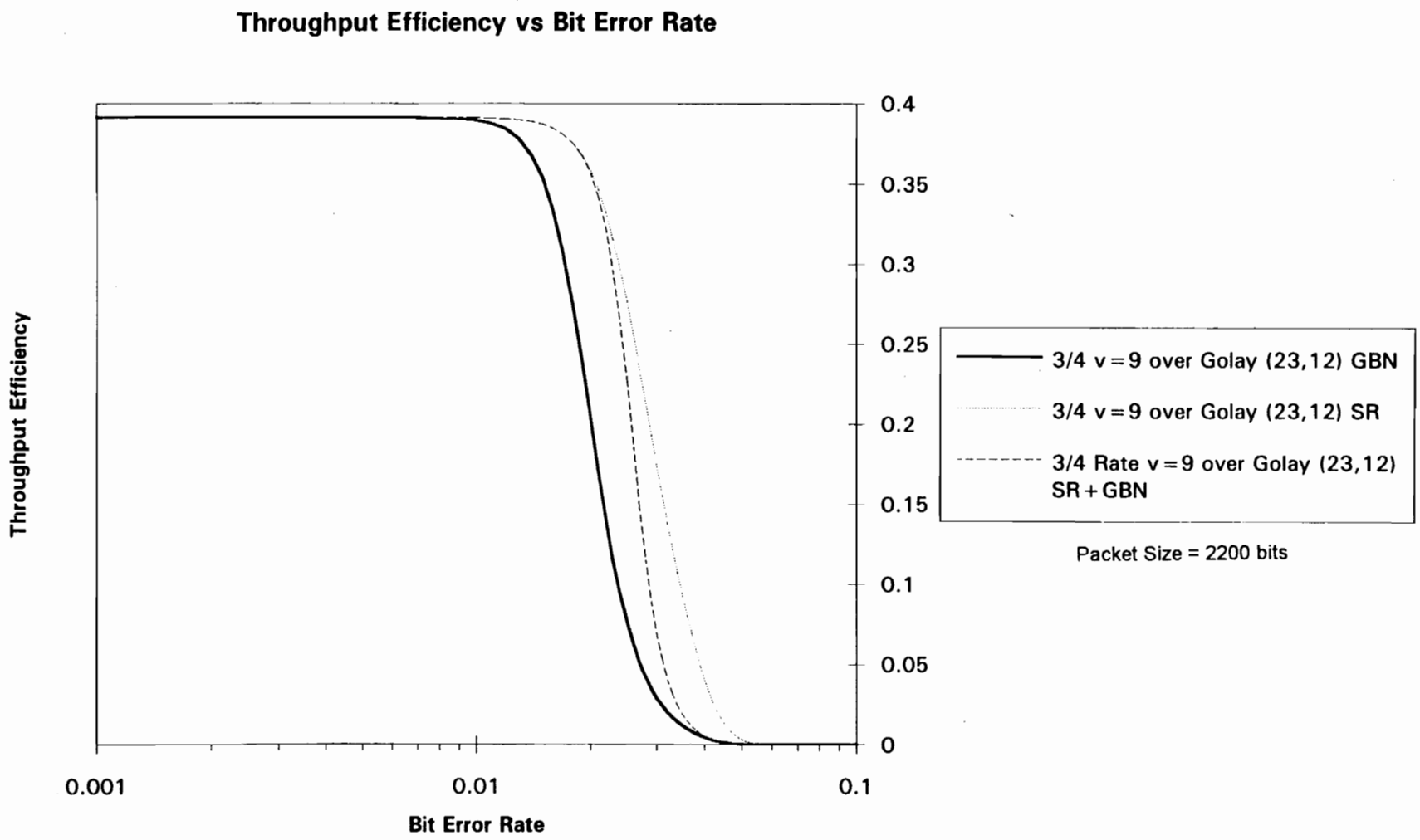

Figure 7-3 Plot of the throughput efficiency vs bit error rate for various ARQ schemes

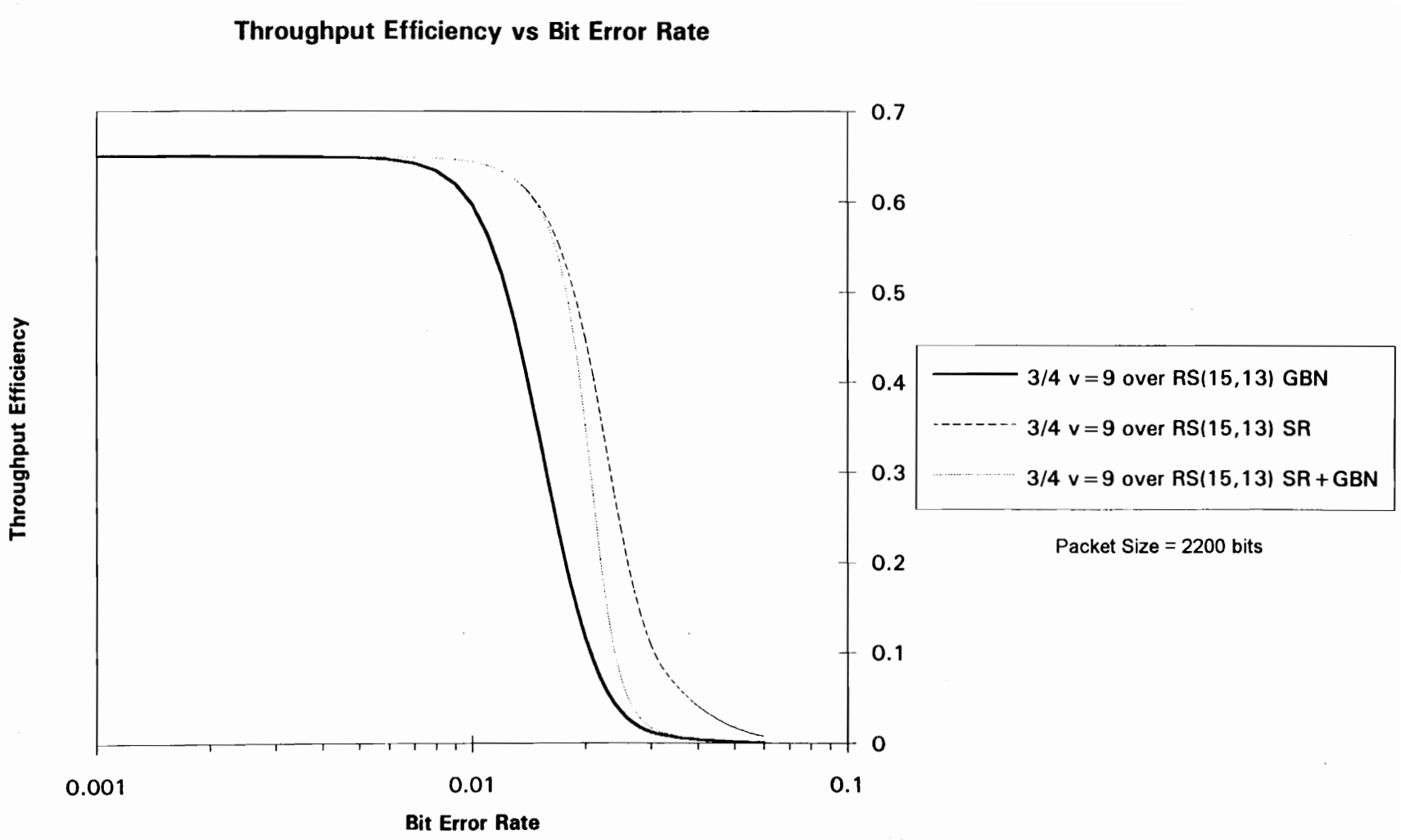

Figure 7-4 Plot of the throughput efficiency vs bit error rate for various ARQ schemes

7.2 ARQ With Mixed Mode of Re-transmission

As discussed in Chapter 3, it is possible to combine ARQ schemes in such a way as to improve the system's throughput performance. One such combination is the *selective-repeat plus go-back-N* (SR + GBN) ARQ. This ARQ method is presented in detail in Chapter 3 and achieves better throughput performance compared with the go-back-N ARQ because of the benefits gained from the use of the selective repeat mode.

7.2.1 Throughput Efficiency

The throughput efficiency for the SR + GBN ARQ is given by

$$\eta_{SR+GBN} = \frac{(1-P_{pe})}{1+(N-1)(P_{pe})^{\nu+1}}\left(\frac{k}{n}\right) \qquad (7\text{-}2)$$

where k/n is the rate of the code used, P_{pe} is the probability of packet error for the code, and ν is the number of re-transmissions in the SR mode allowed before the transmitter switches to the GBN mode. For the examples used here, the SR + GBN ARQ will be used with ν=2. The Figures 7.1 through 7.4 show the improvement which can be obtained in the throughput when the Selective Repeat plus Go-Back-N ARQ is used instead of the Go-Back-N ARQ.

Figures 7.1 through 7.4 show that the selective-repeat ARQ is the better of the ARQ methods, followed by the selective-repeat plus go-back-N with the go-back-N having the lowest performance. Therefore, on a system which is operating with a large channel error rate, it may be beneficial to use a different ARQ method in order to increase the system performance.

CHAPTER 8

CONCLUSION

8.1 Summary

This thesis has shown that the performance of the AX.25 packet radio system can be improved by adding forward error correction to the packet. There are many forward error correcting codes available, so several different types have been compared. Some of the different codes studied were BCH codes, the (23,12) Golay code, Reed-Solomon codes, and various convolutional codes. In order to form a basis of comparison there are several criteria which were used to make a final decision about what, if any, forward error correction should be added to the packet. These criteria include: throughput performance, code rate, and system complexity. As well as studying single codes, the performance of code combining and concatenation of the radio packet have been investigated.

After evaluating all the different codes, it was discovered that the Reed-Solomon and convolutional codes performed the best. These codes provide constant throughput over a large range of bit error rates. Thus either would provide good system reliability even over a noisy channel.

When code combining was used on the AX.25 radio packet, it was found that the overall performance was a combination of the two different codes combined. That means that the system throughput performance was better than

the throughput of the lowest performing code used in the combination, but it was not as good as that of the better performing code. Therefore, code combining was not chosen to incorporate with the AX.25 radio packet ARQ error control scheme.

Another option proposed was to use concatenation. Several different concatenated codes were reviewed and it was discovered that although this process reduced the probability of packet error at lower bit error rates, there was no significant improvement in the throughput. It was therefore concluded that concatenation should not be used on the packet.

It has been found that when coding is applied to an AX.25 radio packet, the system throughput performance can be dramatically increased. The codes which showed the best performance were the Reed-Solomon and convolutional codes, particularly the (255, 187) Reed-Solomon code and the 1/2 rate ν=9 convolutional code. Either of these codes would dramatically increase the system throughput performance when applied to a radio packet. If the channel is very noisy, the convolutional code would be the better choice because it maintains its peak throughput longer than the Reed-Solomon code. If the channel is not too noisy, then the Reed-Solomon code should be used because it has a maximum throughput efficiency of 0.7333 while the maximum throughput for the convolutional code is 0.5.

Both the convolutional and Reed-Solomon codes can be applied at various code rates, and for the convolutional codes, different constraint lengths

as well. The lower rate codes perform better over noisy channels, but when the channel is not too noisy, the higher rate codes pass more information to the user in a given amount of time. So, it is up to the system designer to evaluate the channel and apply the appropriate code parameters.

As AX.25 packet radio continues to expand into new areas of usage, the use of a hybrid ARQ system may be necessary on many channels. For example, as packet radio expands into satellite communications where delay times become longer and error rates become higher, the re-transmission of packets can severely decrease system throughput. By applying coding to the packet, the system throughput can be made constant over a noisy channel, thereby making AX.25 packet radio and satellite communications a more efficient system.

8.2 Further Work

One method which can be used which was not studied is the use of soft-decision decoding. Hard decision decoding occurs when binary demodulator output quantization is used (Q=2). For this case, the decoder has only binary outputs. If a binary-encoded system uses a Q-ary demodulator output (Q > 2), or the output is left unquantized, then the demodulator is said to make *soft decisions*. Soft-decision decoders are more complex than hard-decision decoders because an automatic gain control is needed and $\log_2 Q$ bits have to be manipulated for every channel bit. But soft-decision decoding can offer an

additional coding gain of about 2 dB over hard-decision decoding.[1] For convolutional codes which use Viterbi decoding, the coding gain can be as large as 6 dB if soft-decision decoding is applied.[2] Therefore, it may be worth studying system performance where the use of soft-decision decoders are employed.

Another powerful, but rather simple method which can improve system performance is interleaving. This method was discussed, but it was not applied to any of the codes or packets studied. Interleaving is effective for deriving long burst error correcting codes from shorter codes. It is used to protect the system from errors which occur in bursts. This method is easily implemented by arranging λ code vectors in the original code into λ rows of a rectangular array and then transmitting the array column by column. When interleaving is used, a code which can correct t errors will be able to correct λt errors. This is because no matter where the burst starts, a burst of length λ will not affect more than one digit in each row of the array. By interleaving the code words, the errors are spread over many code words instead of effecting only a few code words. Interleaving can certainly enhance the system error correction capabilities so it may be interesting to apply this technique and study the results.

Soft-decision decoding and interleaving are just two techniques which can be further investigated. Incorporating either of these with the AX.25 radio packet may certainly enhance system performance.

---

[1] Tri T. Ha, p. 460.
[2] Ziemer and Peterson, p. 450.

Appendix A

Golay Code

This appendix shows a listing of the code used to simulate the Golay encoder. This code asks for a twelve bit information word to be encoded and then shows the resulting code word. The program next asks how many bits you wish to corrupt in the code word and then their location (a position from 1 to 23). Up to three bits can be altered. The decoder then shows the altered code word and decodes the word.

To run a simulation of the Golay code, this program was modified to randomly generate packets, encode them, and then corrrupt them at a specified bit error rate. The coded packets were decoded looking for any errors in decoding. When any errors occured a count was made so that the probability of packet error could be found.

The following data is a representation of the first decoder showing an example where bits 1 3 and 5 of the code word are corrupted.

data to encode: 1 0 0 1 1 1 0 0 0 1 0 1

encoded word: 1 0 1 1 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 1 1 1 1

corrupted word: 0 0 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 1 1 1 1

corrected word: 1 0 1 1 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 1 1 1 1

```
*********************************************************************
*This is a program that will encode data using the (23,12) Golay code
*  The word to be encoded must be 12 bits long
*  Up to 3 bits can be corrupted in the code word
*
*********************************************************************

      PROGRAM GOLAYENC
      INTEGER CW(24),GENMAT(12,23),HMAT(11,23),I,
     .INFO(12),K,N,PARBIT, RCW(24),SYN(11),
     .RPARBIT, EV(24),TEMP(12,23),TEMP2(11,23),SUM,VAR, SYND(11,23),
     .CCW(11),RCWEV(23),WT,EFLAG,SHFT,ORG,NO,ONEBIT(3),ORG2

      OPEN(UNIT=12, FILE='ENCDAT',STATUS='OLD')
      OPEN(UNIT=13, FILE='GPOLY',STATUS='OLD')
      READ(12,*)(INFO(I), I=1,12)

      PRINT *, 'ENTER A 12 LENGTH INFO WORD'
      READ *, (INFO(I), I=1,12)
      READ(13,*)((GENMAT(I,J), J=1,23), I=1,12)

C     MULTIPLY THE INFO TO BE ENCODED BY THE GENERATOR MATRIX TO FORM
C     THE 23 BIT ENCODED WORD

      DO 30 I=1,12
        DO 25 J=1,23
          TEMP(I,J)=INFO(I)*GENMAT(I,J)
 25     CONTINUE
 30   CONTINUE

      DO 40 J=1,23
        CW(J)=TEMP(1,J)
        DO 35 I=2,12
          IF(CW(J) .EQ. 1 .AND. TEMP(I,J) .EQ. 1)THEN
            CW(J)=0
          ELSE
            CW(J)=CW(J)+TEMP(I,J)
          ENDIF
 35     CONTINUE
 40   CONTINUE

      PRINT *, 'The (23,12) Endcoded word is'
      PRINT 50, (CW(J), J=1,23)
 50   FORMAT (1X, 23I2)

     THE NEXT STEP WILL BE TO ADD AN EVEN PARITY CHECK BIT TO MAKE A
     (24,12) CODE WORD
      CALL PARITYCK(CW, PARBIT)
      CW(24)=PARBIT

c     The next step is to corrupt some bits
c     Form an error vector of length 24 and add it to CW

      PRINT *, 'Enter the number of bits to corrupt'
      READ *, NO
      PRINT *, 'Enter the positions of the bits to corrupt'
```

```
      READ *, (ONEBIT(I), I=1,NO)
      DO 5, I=1,23
        EV(I)=0
    5 CONTINUE
      DO 6 I=1,NO
       EV(ONEBIT(I))=1
    6 CONTINUE

      DO 75 I=1,24
       IF(EV(I) .EQ. 1 .AND. CW(I) .EQ. 1)THEN
         RCW(I)=0
       ELSE
         RCW(I)=CW(I)+EV(I)
       ENDIF
   75 CONTINUE

      PRINT *, 'THE RECEIVED CODEWORD IS'
      PRINT 76, (RCW(I), I=1,23)
   76 FORMAT(1X, 23I2)
*****************************************************************
c     The next step is to make the decoder

      OPEN(UNIT=15, FILE='HMAT', STATUS='OLD')
      READ(15,*)((HMAT(I,J), J=1,23), I=1,11)

      WT=3
      CALL EVECTOR(RCW,WT,HMAT,RCWEV,EFLAG,SHFT)
      EFLAG=0
      IF(EFLAG .EQ. 1)THEN
       GOTO 150
      ELSE
        DO 90 I=1,23
          ORG=RCW(I)
          IF(RCW(I) .EQ. 1)THEN
            RCW(I)=0
          ELSE
            RCW(I)=1
          ENDIF
          CALL EVECTOR(RCW,WT,HMAT,RCWEV,EFLAG,SHFT)
          EFLAG=0
          IF(EFLAG .EQ. 1)THEN
           GOTO 150
          ELSE
            RCW(I)=ORG
          ENDIF
   90   CONTINUE
      ENDIF

      DO 100 I=1,22
        ORG=RCW(I)
        IF(RCW(I) .EQ. 0)THEN
         RCW(I)=1
        ELSE
         RCW(I)=0
        ENDIF
        DO 95 J=I+1,23
```

```
        ORG2=RCW(J)
        IF(RCW(J) .EQ. 0)THEN
          RCW(J)=1
        ELSE
          RCW(J)=0
        ENDIF
        CALL EVECTOR(RCW,WT,HMAT,RCWEV,EFLAG,SHFT)
        RCW(J)=ORG2
 95   CONTINUE
      RCW(I)=ORG
100 CONTINUE

C     CORRECT THE RCW USING THE RCWEV COMPUTED ABOVE
C     CCW IS THE CORRECTED CODEWORD

150 DO 160 I=1,12
      IF(RCW(I) .EQ. 1 .AND. RCWEV(I) .EQ. 1)THEN
        CCW(I)=0
      ELSE
        CCW(I)=RCW(I)+RCWEV(I)
      ENDIF
160 CONTINUE
    DO 170 I=13,23
      CCW(I)=RCW(I)
170 CONTINUE


C    SHIFT THE CODEWORD TO THE LEFT SHFT TIMES
     IF(SHFT .EQ. 0)THEN
       GOTO 195
     ELSE
       DO 190 I=1,SHFT
         VAR=CCW(1)
         DO 180 J=1,22
           CCW(J)=CCW(J+1)
180      CONTINUE
         CCW(23)=VAR
190    CONTINUE
     ENDIF

195 PRINT *, 'THE CORRECTED CODE WORD IS'
    PRINT 200, (CCW(I), I=1,23)
200 FORMAT(1X, 23I2)
    END


**********************************************************************
*    SUBROUTINE THAT WILL CALCULATE THE EVEN PARITY CHECK BIT
*
**********************************************************************
    SUBROUTINE PARITYCK(CW, PARBIT)
    INTEGER CW(23), I, PARBIT
    REAL SUM

    SUM=CW(1)
    DO 10 I=2,23
```

```
      SUM=SUM+CW(I)
 10  CONTINUE
     SUM=SUM/2
     VAL=0
     DO 20 I=1,12
      IF(VAL .EQ. SUM)THEN
       PARBIT=0
       GOTO 25
      ELSEIF(VAL .LT. SUM)THEN
       GOTO 15
      ELSE
       PARBIT=1
       GOTO 25
      ENDIF
 15   VAL=I
 20  CONTINUE
 25  END



*********************************************************************
*    SUBROUTINE THAT WILL COMPUTE THE ERROR VECTOR
*
*********************************************************************
     SUBROUTINE EVECTOR(RCW,WT,HMAT,RCWEV,EFLAG,SHFT)
     INTEGER RCW(23),WT,HMAT(11,23),RCWEV(11),EFLAG,HAMWT,
    .SYN(11),L,M,SHFT,SYNCHK(11),CCWT(23),HAMWTCCW,VAR

     DO 50 L=1,23
      CALL SYNDROME(RCW,HMAT,SYN)
*
*      CALCULATE THE HAMMING WEIGHT OF THE SYNDROME
*
      HAMWT=SYN(1)
      DO 10 I=2,11
         HAMWT=HAMWT+SYN(I)
 10    CONTINUE

      IF(HAMWT .LE. WT)THEN
        DO 20 M=1,11
         RCWEV(M)=SYN(M)
 20      CONTINUE
        EFLAG=1
        SHFT=L-1
        GOTO 30
      ELSE
        EFLAG=0
        GOTO 30
      ENDIF
*
*       COMPUTE THE RIGHT CYCLIC SHIFT OF THE RECEIVED CODEWORD
*       ADD THE RCWEV TO THE WORD AND RECALCULTE SYNDROME
*
 30    DO 31 I=1,12
       IF(RCW(I) .EQ. 1 .AND. RCWEV(I) .EQ. 1)THEN
        CCWT(I)=0
```

```
      ELSE
        CCWT(I)=RCW(I)+RCWEV(I)
      ENDIF
 31   CONTINUE
      DO 32 I=13,23
        CCWT(I)=RCW(I)
 32   CONTINUE

      CALL SYNDROME(CCWT,HMAT,SYNCHK)
      HAMWTCCW=SYNCHK(1)
      DO 33 I=2,11
         HAMWTCCW=HAMWTCCW+SYNCHK(I)
 33   CONTINUE
      IF(HAMWTCCW .EQ. 0)THEN
*
*     shift the ccwt and this is the decoded word
*
       IF(SHFT .EQ. 0)THEN
        GOTO 37
       ELSE
        DO 36 I=1,SHFT
          VAR=CCWT(1)
          DO 35 J=1,22
            CCWT(J)=CCWT(J+1)
 35        CONTINUE
          CCWT(23)=VAR
 36       CONTINUE
       ENDIF
 37     PRINT *, 'THE CORRECTED CODE WORD CCWT IS'
       PRINT 38, (CCWT(I), I=1,23)
 38     FORMAT(1X,23I2)
       STOP

      ELSE
       GOTO 39
      ENDIF


 39    VAR=RCW(23)
      DO 40 J=23,2,-1
       RCW(J)=RCW(J-1)
 40    CONTINUE
      RCW(1)=VAR
      IF(L .EQ. 23)THEN
       DO 45 M=1,11
        RCWEV(M)=0
 45     CONTINUE
      ELSE
       GOTO 50
      ENDIF
 50   CONTINUE
 60   END
```

```
*********************************************************************
*    SUBROUTINE THAT COMPUTES THE SYNDROME OF THE RECEIVED CODEWORD *
*                                                                   *
*********************************************************************
      SUBROUTINE SYNDROME(MAT1, MAT2, PROD)
      INTEGER TEMP2(11,23), MAT1(23),MAT2(11,23),PROD(11),
     .SUM(11)

C     MAT1 IS RCW, MAT2 IS HMAT, AND THE SYMDROME IS PROD

      DO 10 I=1,11
        DO 5 J=1,23
          TEMP2(I,J)=MAT1(J)*MAT2(I,J)
    5   CONTINUE
   10 CONTINUE
      DO 20 I=1,11
        SUM(I)=TEMP2(I,1)
        DO 15 J=2,23
          IF(SUM(I) .EQ. 1 .AND. TEMP2(I,J) .EQ. 1)THEN
            SUM(I)=0
          ELSE
            SUM(I)=SUM(I)+TEMP2(I,J)
          ENDIF
   15   CONTINUE
   20 CONTINUE
      DO 30 I=1,11
        PROD(I)=SUM(12-I)
   30 CONTINUE
      END
```

Appendix B

List of Symbols

ARQ	Automatic Repeat Request
BER	Bit Error Rate
DISC	Disconnect
DM	Disconnected Mode
FCS	Frame Check Sequence
FRMR	Frame Reject
GBNARQ	Go-Back-N ARQ
I	Information
REJ	Reject
RNR	Receive Not Ready
RR	Receive Ready
S	Supervisory
SABM	Set Asynchronous Balanced Mode
SRARQ	Selective Repeat ARQ
SSIDS	Secondary Station Identifiers
SWARQ	Stop and Wait ARQ
U	Unnumbered
UA	Unnumbered Acknowledgement
UI	Unnumbered Information

Appendix C

References

[1] A. M. Rosie, *Information and Communication Theory*, London: Van Nostrand Reinhold Company, 1973.

[2] Djimitri Wiggert, *Error-Control Coding and Applications*, Dedham, Massachusetts: Artech House, Inc., 1978.

[3] E.F. Assmus, JR. and J.D. Key, *Designs and their Codes*, Cambridge: University Press, 1992.

[4] Franklin M. Ingels, *Information and Coding Theory*, Scranton: Intext Educational Publishers, 1971.

[5] Henry R. Reed and Carl M. Russell, *Ultra High Frequency Propagation*, London: Chapman & Hall LTD, 1964.

[6] J. Chen and P. Owsley, "A Burst-Error-Correcting Algorithm for Reed-Solomon Codes," *IEEE Transactions on Information Theory*, Vol. 38, No. 6, November 1992, pp. 1807-1812.

[7] J. Pieter M. Schalkwijk and Karel A. Post, "On a Method of Calculating the Event Error Probability of Convolutional Codes with Maximum Likelihood Decoding," *IEEE Transactions on Information Theory*, Vol. IT-25, No. 6, November 1979, pp. 737-743.

[8] Jeffrey L. Anderson, "On Minimal Decoding Sets for the Extended Binary Golay Code," *IEEE Transactions on Information Theory*, Vol. 38, No. 5, September 1992, pp. 1560-1561.

[9] Jiri Adamek, *Foundations of Coding*, Chinchester: John Wiley & Sons, Inc., 1991.

[10] Kyungwhoon Cheun and Wayne E. Stark, "Optimal Selection of Reed-Solomon Code Rate and the Number of Frequency Slots in Asynchronous FHSS-MA Networks," *IEEE Transactions on Communications*, Vol. 41, No. 2, February 1993, pp. 307-311.

[11] L. Van De Meeberg, "A Tightened Upper Bound on the Error Probability of Binary Convolutional Codes with Viterbi Decoding," *IEEE Transactions on Information Theory,* May 1974, pp.389-391.

[12] Leonard S. Schwartz, *Principles of Coding, Filtering, and Information Theory*, London: Cleaver-Hume Press, LTD., 1963.

[13] R. J. Deasington, *X.25 Explained*, Chichester: Ellis Horwood Limited, 1985.

[14] Robert K. Morrow and James S. Lehnert, "Packet Throughput in Slotted ALOHA DS/SSMA Radio Systems with Random Signature Sequences," *IEEE Transactions on Information Theory*, Vol. 40, No. 7, July 1992, pp. 1223-1230.

[15] Rodger E. Ziemer and Roger L. Peterson, *Introduction to Digital Communications*, New York: Macmillian Publishing Company, 1992.

[15] Shu Lin, *An Intorduction to Error-Correcting Codes*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1970.

[16] Shu Lin, Daniel J. Costello Jr., and Michael J. Miller, "Automatic-Repeat-Request Error Control Schemes," *IEEE Communications Magazine*, Vol. 22, No. 12, December 1984, pp. 5-15.

[17] Shu Lin and Daniel J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983.

[18] Stan Horzepa, *Your Gateway to Packet Radio*, Newington, CT: American Radio Relay League, 1989.

[19] Stephen B. Wicker, " Reed-Solomon Error Control Coding." *IEEE Trasnactions on Vehicular Technology*, Vol. 41, No. 2, May 1992, pp. 128-133.

[20] Stephen Horan, *PCM Telemetering Systems*, Boca Raton: CRC Press, 1993.

[21] Tri T. Ha, *Digital Satellite Communications*, New York: McGraw-Hill, Inc., 1990.

[22] Vera Pless, *Introduction to the Theory of Error-Correcting Codes*, New York: John Wiley and Sons, 1989.

[23] W. Wesley Peterson, *Error-Correcting Codes*, Cambridge Massachusetts: The M.I.T Press, 1961.

[24] William C. Lindsey and Marvin K. Simon, *Telecommunication Systems Engineering*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1973.

[25] William E. Ryan and Paul Conoval, "A Method of Analysis for Interleaved Reed-Solomon Coding with Erasure Decoding on Burst Error Channels," *IEEE Transactions on Communications*, Vol. 41, No. 3, March 1993, pp. 430-434.

VITA

The author was born in Fairfax, Virginia on June 25, 1970. He attended Park View High School in Sterling, Virginia where he graduated in June 1988. After high school, the author went to Virginia Polytechnic Institute and State University where he received a B.S. degree in Electrical Engineering in May 1992.

After graduating college, the author entered graduate school at Virginia Tech in August 1992, and is expected to receive a M.S. degree in Electrical Engineering in May 1994.