ANALYSIS AND DESIGN OF SOURCE / CHANNEL CODES FOR NOISY COMMUNICATION CHANNELS

A Thesis submitted by

Alan Douglas Clark

in partial fulfillment of the requirements for the

DEGREE OF DOCTOR OF PHILOSOPHY

of the

COUNCIL FOR NATIONAL ACADEMIC AWARDS

submitted in June 1987

School of Electronic and Electrical Engineering

LEICESTER POLYTECHNIC

Collaborating Establishment:

BRITISH TELECOM PLC

Acknowledgements

I would like to acknowledge the assistance and support provided by my supervisors, Dr A. A. Hashim of Leicester Polytechnic, and Dr A. G. Constantinides of Imperial College, and my advisors Mike Morse and Nick Frydas throughout the research programme.

The project was sponsored by the Datacomms Unit of British Telecom PLC, International Products Division under contract 589360. In addition, considerable technical advice was received for which I am grateful. Thanks are due to Mike Buckley, Gerry Comber, John Holdsworth, John Magill, Yannis Chirides, Dave Trouse, Rita Shah and Graham Steele, of BT Datacomms, and to John Brownlie, Peter May, and David Bond of BT Research Laboratories.

Advice and assistance from, and discussion with, one's colleagues are invaluable, and I would like to thank Said Amir, Swapan Parui, and other members of the former Digital Signal Processing Research Group at Leicester Polytechnic.

And finally, I would like to particularly thank my long suffering wife Glenis, and our children Adam and Elizabeth, who have been remarkably patient and tolerant of "Dad finishing his thesis".

ANALYSIS AND DESIGN OF SOURCE/CHANNEL CODES FOR NOISY COMMUNICATION CHANNELS

A. D. CLARK

ABSTRACT

The rapid expansion in the field of information technology in recent years, has led to an increased awareness of the need for efficient, reliable communications systems. Although digital networks are being introduced, modems are widely used for transmitting digital information over the Public Switched Telephone Network (PSTN). This approach suffers from two drawbacks, limited bandwidth and transmission errors. The objectives of this research program were to investigate methods for compressing digital information and for correcting transmission errors, to enhance the performance of a voiceband modem.

The application of source coding to a modem based communications system requires efficient compression and low implementation complexity. The type of data transmitted is unknown, hence the source encoder must adapt to the data during transmission. The design and performance of a number of robust source codes, in particular the problem of designing codes with constrained maximum length is discussed, and a number of solutions proposed. Adaptive variable length and string encoding techniques are compared. The Ziv-Lempel encoding algorithm is investigated, and a number of improvements suggested.

The distribution of transmission errors is affected by the type of disturbance causing the errors, and the design of the modem. The characteristics of PSTN transmission errors are discussed, and the design of error control systems considered. A number of automatic repeat request (ARQ) and hybrid error control schemes are discussed, and their performance evaluated under a range of channel conditions. The quality of a telephone channel is variable, and the adaptive selection of frame length and code rate can result in a performance improvement. The design of an adaptive hybrid ARQ scheme is discussed, and its performance compared to conventional methods.

A number of practical design considerations are given. The design of three source/channel coding systems is discussed, and their performance compared.

Original Aspects of the Research Program

The research program, although containing a substantial theoretical element, was essentially oriented towards a practical goal. A major part of the work comprised comparative performance analyses, and consideration of the practical implementation of coding algorithms. An element of the work not discussed in this thesis was the practical implementation of a communication system based on some of the ideas discussed below, involving both hardware and software design. This practical work is reflected throughout the thesis, with particular emphasis in Chapter 6 on system implementation and design.

The elements of the work that are to the best of my knowledge original are:-

- The quantitative discussion of the effects of concatenating run length and variable length encoding. (Section 2.6)

- The extension to the work of Gilbert (1971) and Van Voorhis (1974) in the development of constrained variable length codes. (Section 3.2.2)

- The extension to the work of Faller (1974) and Gilbert (1978) in the development of adaptive variable length codes, and consideration of the performance of the codes on non-stationary sources. (Section 3.3.1)

- Two modifications to the Ziv-Lempel (1976) compression algorithm, space synchronization and an improved data structure/ dictionary maintenance technique. (Section 3.3.2)

- Consideration of the feasibility of compressing a synchronous data stream, in which the symbol size is unknown. (Section 3.3.3)

- The qualitative discussion of the telephone channel error distribution, and the effects of modems on the error patterns. (Section 4.2, 4.3)

- The comparative performance analysis of ARQ and hybrid ARQ error control schemes under a wide range of channel conditions. (Sections 5.4-5.6)

- A specific adaptive hybrid ARQ scheme, and its performance on random and burst channel models. (Section 5.7)

- The interaction of data compression and error control elements of a communications system. (Section 6.3)

- The discussion of practical implications in the design of data compression and error control schemes. (Sections 2.7, 3.4, 4.2.2, 5.4, 5.5, 5.8, 5.9, Chapter 6).

CONTENTS

Symbols and Abbreviations

Abbreviations

ARQ	Automatic Repeat Request
BCH	Bose, Chaudhuri, Hocquenghem error correcting code
BER	Bit Error Rate
BISYNC	Binary Synchronous Communications Protocol (IBM)
BLER	Block Error Rate
BSC	Binary Symmetric Channel
CCITT	Comite Consultatif Internationale de Telegraphique et Telephonique
DMS	Discrete Memoryless Channel
EFS	Error Free Seconds
FEC	Forward Error Correction
GBN	Go Back N (ARQ)
GF	Galois Field
HDLC	High-Level Data Link Control protocol (ISO)
HF	High Frequency (radio channel)
iff	if and only if
ISO	International Standards Organization
LZ	Lempel-Ziv compression algorithm
OSI	Open Systems Interconnection model
pdf	probability density function
PSTN	Public Switched Telephone Network
SDLC	Synchronous Data Link Control protocol (IBM)
SR	Selective Repeat ARQ
STN	Switched Telephone Network
SW	Stop and Wait ARQ

VDU	Visual Display Unit
VL	Variable length (code)
wrt	with respect to
ZL	Ziv-Lempel (see LZ)

Symbols

The following symbols are generally in accordance with those in common use in the field of information theory. Although this has occasionally resulted in two definitions of the same symbol, this causes no ambiguity when in context.

${}_nC_r$	The number of combinations of r objects from a set of n
D	The radix of a code
D	End to end channel delay (in bits)
d	The distance of a codeword (e.g. Hamming distance)
d_{min}	The minimum distance of a code
F(s)	The s-th Fibonacci number
$\underline{\mathbf{G}}$	The generator matrix of a linear block code
$\underline{\mathbf{H}}$	The parity check matrix of a linear block code
H	The entropy of a source
$H_{i,j}$	The entropy of the first extension of a source
$H_i(j)$	The conditional entropy of a source
h	The size of header field in an ARQ frame
I_i	The information content of symbol s_i
$I_{i,j}$	The information content of the symbol pair (s_i,s_j)
$I_i(j)$	The conditional information content of symbol s_j
k	The number of information bits in an (n,k) codeword
L_i	The length of a source codeword
L'	The average length of a source code

L_{con}	The constrained maximum codeword length of a variable length source code
L_{max}	The maximum codeword length of a variable length source code
M	The marginal information content of a symbol
N	The number of symbols in a source alphabet
N	The acknowldgement delay of an ARQ system (in frames)
n	The length of a linear block code codeword.
p_t	The transition probability of a binary symmetric channel
$P_i(j)$	The conditional probability of event j, given event i
P(>m,n)	The probability that a block of length n contains more than m errors
s_i	The i-th symbol from a source
t	The number of errors correctable by an (n,k,t) linear block code.

1 INTRODUCTION

1.1 Background to the Research Project

During the past thirty years the importance of data communications to the business and scientific community has grown tremendously. Many businesses are now heavily dependent on their communications equipment to allow data to be transferred between computers or to provide remote access to common resources. It is therefore highly desirable that data transfer may be accomplished quickly, reliably, and at low cost.

A common medium of transmission is the Public Switched Telephone Network, which provides voice grade communication channels between virtually any two points in the world. A modem is used to convert the digital signal to, or from, a form which is compatible with the requirements of the voice channel in terms of power level and spectrum (Figure 1.a). Unfortunately, as the telephone network was not designed as a carrier of data, the modulated signal is subject to distortion and additive noise which may result in errors in the reconstructed digital signal.

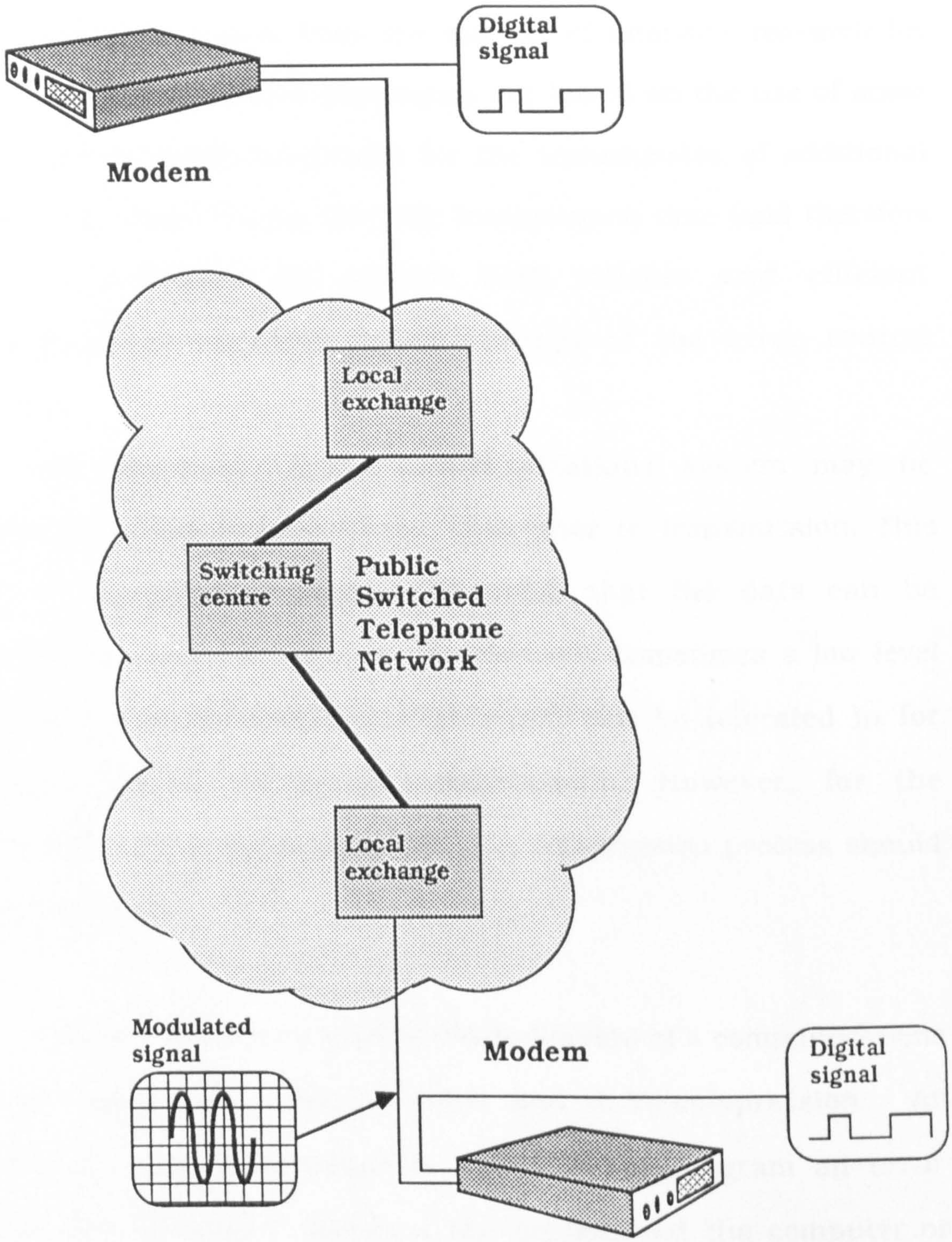

Figure 1.a The use of modems for transmitting digital information over the Public Switched Telephone Network

The detection and correction of transmission errors is an essential part of a data communications system, hence methods for accomplishing this have been the subject of intensive research for many years. Error control techniques are based on the use of some part of the available bandwidth for the transmission of additional information, which means that the transmission time (and therefore cost) are increased. To provide both reliable *and* efficient communications requires careful design of the error control function.

The efficiency of the communications system may be enhanced by compression of the data prior to transmission. This must be a reversible process in order that the data can be reconstructed without error by the receiver. Sometimes a low level of distortion of the reconstructed signal can be tolerated in for example, speech or image transmission. However, for the transmission of digital information, the compression process should be distortionless.

Figure 1.b shows a general block diagram of a communications system incorporating error control and data compression. An additional component, which is called in the diagram an *error control unit*, is placed between the modem and the computer or terminal at each end of the link. This type of device is available from a number of different manufacturers. All use automatic repeat request (ARQ) error control, with data compression to compensate for the loss in channel capacity.

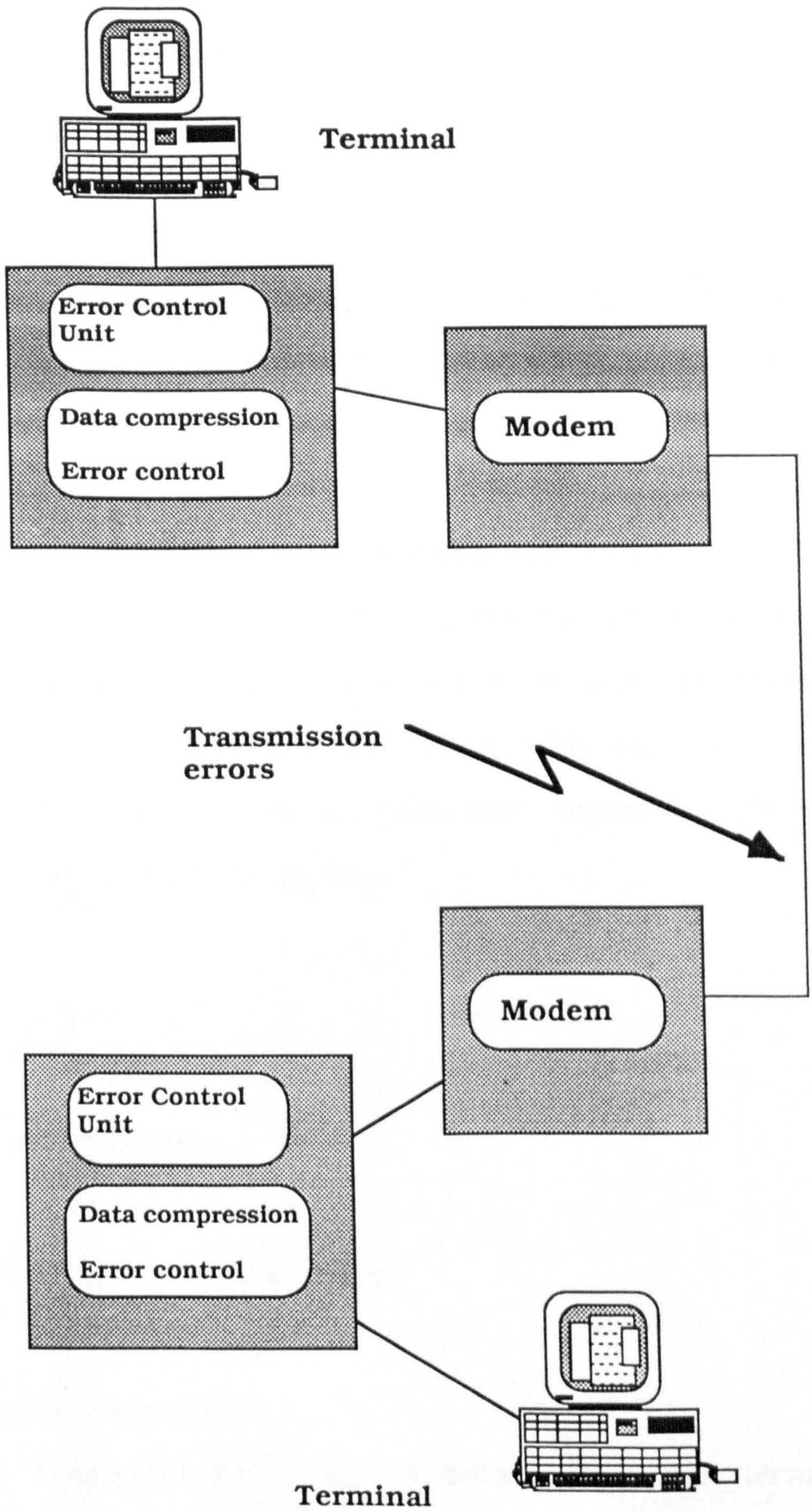

Figure 1.b The use of error control units to provide error correction and data compression over a modem link.

Some commercial error control units offer an additional multiplexing facility. For example, the Timeplex Datamizer provides four multiplexed channels, using an adaptive Huffman code for data compression, and ARQ for error control. Products offering only a single channel capability are the DaCom Error Controller, Modular Technology Interblaster, and others, mostly intended for use with low speed modems.

The project involved an investigation of the use of data compression and error control techniques for enhancing the performance of modem based communications, and consideration of the practical aspects of implementation. This was aimed at the eventual development of new products with improved performance, for an already competitive market.

1.2 Aims and objectives

The aims of the project were:-

(i) Data compression

To investigate the range of data compression techniques appropriate to the application, and propose new or improved methods. These techniques must be effective under realistic conditions, i.e. compression should be achieved with as little prior knowledge of the source data as possible. The complexity of

implementation was also very important, as the data compression component must operate in real time, with a restricted memory capacity.

(ii) Error control

To investigate the use of error control with high speed voice band modems. Although retransmission error control (ARQ) is widely used in this type of application, more powerful hybrid schemes incorporating forward error correction have been developed in recent years. As with data compression, complexity is an important consideration.

(iii) System integration and design

To place the data compression and error control components in context within a system, to investigate methods of implementation and examine any conflicting requirements.

1.3 Overview of the thesis

The general structure of the thesis follows Section 1.2., discussing in turn data compression, error control, and system design.

Data compression is discussed in Chapters 2 and 3, initially for the ideal case in which the characteristics of the data source are known, and then for the more realistic case of a partially known source.

Huffman coding, adaptive Huffman coding, and some variants are examined, with emphasis on the problem of reducing the maximum length of the codes. The more recent compression algorithm of Ziv and Lempel is discussed and a number of improvements proposed. In addition, data for which the symbol size is unknown is shown to be compressible in some cases, using the Ziv-Lempel compression algorithm; this has particular significance in the compression of synchronous data.

Chapter 4 discusses telephone channel characteristics, and error statistics and modelling. This is followed by a brief introduction to error correction and detection coding.

In Chapter 5, a number of error control schemes based on automatic repeat request (ARQ) are compared under a range of channel conditions. These include hybrid ARQ, parity retransmission and adaptive schemes. The choice of forward error correcting code rate and frame length are discussed, and the reliability of the error

detection code considered.

System design, encompassing a range of practical considerations, is discussed in Chapter 6. An ARQ protocol provides more than simple error control; the additional features include end-to-end signalling, flow control and signalling. Three alternative systems are proposed, each appropriate to a particular application, and their performance compared.

Finally, Chapters 7 and 8 give the conclusion of the thesis, and some suggestions for further study.

2. SOURCE MODELS AND SOURCE CODING

2.1 Introduction.

The efficient use of channel capacity is of immediate interest to users of data communications networks. A large wide area network, for example, may incur line rental costs of over ten million pounds per annum; the use of data compression can provide improved throughput and hence a more cost effective service.

Although source coding techniques date from early in the Nineteenth Century, recent developments in microprocessor technology, coupled with a demand for high speed low cost data transfer, have provided a new impetus to the development of powerful data compression systems.

The theoretical foundations for source coding were laid by Hartley (1928), and Shannon (1948). Hartley's paper entitled "*Transmission of Information*" discussed a number of then current issues in telegraphy, including the statement that the number of code symbols (for example, bits) required to encode a source symbol was proportional to the logarithm of the number of source symbols. Although Morse and Vail (Bylanski 1980) had realized that an improvement in efficiency could be obtained by using a variable codeword size, and assigning short codewords to frequently occurring characters, the nature of information and the performance bounds for this type of coding were not known until the work of

Shannon in 1948.

The source coding techniques addressed in this and the next chapter, are noiseless or distortionless, in that the encoding process is reversible without error. In general, there is some relationship between the instantaneous compression ratio or code rate, and the distortion introduced by the coding scheme. Rate-distortion coding attempts to achieve a constant code rate, allowing the distortion or error to vary. In noiseless coding systems however, the code rate is uncontrolled, which can lead to practical design problems such as buffer overflow (Humblet 1981).

The design of the source code is based on knowledge of, or assumptions about, the frequencies and ordering of the source symbols constituting messages. This knowledge is represented in the form of a mathematical model of the source. A number of source models are used, some of which will be described in more detail in Sections 2.2 and 2.4. For example, the discrete memoryless source (DMS) model is based on the assumption that source symbols are independently selected from some alphabet, the model parameters are the occurrence probabilities of the symbols.

In practice there are often local relationships between symbols within a message, for example "*the*" is a common group of symbols used in English text. These relationships can be used to advantage in a coding scheme, and several source models have been devised to represent them.

The performance of the source encoder will obviously depend on the correct choice of source model, and knowledge of the parameters involved. If an inappropriate model is used, or if the

parameters are inaccurately known or unstable, the compression achieved may be poor, in the extreme even expansion may result. In this chapter an ideal situation will be assumed, i.e. a stable source for which the parameters are known, the next chapter will consider the more usual case, in which the source is inaccurately known and time varying.

Although the following discussion is valid for codes based on non-binary code symbol sets (for example ternary), binary codes will be assumed. The term *bits* will therefore be used for *binary digits* and for the binary unit of information, and unless otherwise stated logarithms will be taken to base two.

The results in the following sections and in Chapter 3 were obtained by computer simulation or implementation of the source coding techniques, with a number of different sources. A sample from each of the sources is given in Appendix B for reference.

2.2 The Discrete Memoryless Source.

The Discrete Memoryless Source (DMS) is the simplest and most widely used model. It is assumed that the source emits symbols selected at random from an alphabet, each choice being made independently of any earlier selection. Associated with each source symbol s_j is a probability p_j which corresponds to the probability of selection for the symbol.

It will be assumed for the moment that the probabilities $\{p_0, p_1 , p_2 , .. p_{N-1} \}$ corresponding to the N symbols in the source alphabet are known, and stationary. Hartley (1928) proposed a logarithmic measure of information content, and hence of the number of code symbols required to represent the source, but did not consider the effects of unequal source symbol probabilities.

Shannon (1948) considered the case in which source symbols do have associated probabilities, and defined the information content of a symbol, I_j, and the average information content of the source - the entropy *H*.

For each source symbol s_j, the information carried by the symbol is:-

$$I_j = \log(1/ p_j) \quad \text{bits}$$

A measure of the average information per source symbol is the source entropy:-

$$H = \sum_{J=0}^{N-1} p_J I_J$$

For any given message *M*, consisting of a sequence of k symbols independently selected from the source alphabet, the total information carried by the message is the sum of the information carried by the symbols in the message. Thus for message *M*:-

$$M = (s_a, s_b, s_c, s_d, \ldots s_x).$$

where a, b, c, d, .. x are in the range {0 .. N-1}, the probability that message M will be generated is:-

$$P(M) = P(s_a).P(s_b).P(s_c).P(s_d) \ldots P(s_x)$$

and hence the information content of the message is:-

$$I(M) = \log(1/P(M)) \text{ bits}$$

$$= I(s_a)+I(s_b)+I(s_c)+\ldots+I(s_x) \text{ bits}$$

The average information per symbol contained in *M* is

therefore given by:-

$$I(\text{ average }) = I(M) \ / \ k \qquad \text{bits per symbol}$$

As k becomes large, then the average information per symbol will approach the source entropy (as k tends to infinity, the relative frequency of each symbol will by definition be given by the symbol probability), and may be regarded as an estimate of the entropy. The term *sample entropy* will be used to denote the estimate of the source entropy, obtained under the assumption that the occurrence frequencies of symbols within a message or sample are in direct proportion to the symbol probabilities.

Table 2.a shows values for the sample entropy for a number of sources. Most text samples have values of 4 to 4.5 bits per symbol, whilst numeric data generally have a lower information content (requiring usually eleven or twelve symbols rather than the 27 or more needed for text). Executable computer program code usually has a high information content, as a large number of symbols are used with similar frequency.

Sample type	Sample Entropy	Comment
English text	4.03 bits per symbol	Shannon(1951)
English text	4.16 bits per symbol	Measured
Portuguese text	3.92 bits per symbol	Manfrino(1969)
FORTRAN	5.29 bits per symbol	Measured
ALGOL	5.58 bits per symbol	Wells(1972)
Executable code	5.80 bits per symbol	Measured

Table 2.a Examples of sample entropy for various types of data.

Digram encoding, in which pairs of characters are represented by codewords, is often used. Under the assumption that the source is memoryless, the probability of some pair (s_i, s_j) occurring is given by the product $p_i.p_j$, and thus the information content by:-

$$I_{i,j} = I_i + I_j$$

The entropy of the digram source is twice that of the original source, and the extended source alphabet contains N^2 symbols.

2.3 Variable Length Codes.

A source encoder accepts symbols or sequences of symbols from a source, and generates codewords. As no error can be allowed in this application, the encoding must preserve the information content of the message. For the discrete memoryless source, the information content of a message and the source entropy have been defined above. If the DMS model is assumed, then an ideal source encoder would encode the message in a number of bits corresponding to its information content (Shannon 1948).

The encoding process may accept variable length sequences of symbols, outputting a fixed length codeword for each, or fixed length sequences of source symbols, outputting a variable length codeword for each symbol. The latter technique is usually termed variable length coding.

A variable length encoder assigns a codeword of length L_j to each symbol s_j contained in the message. In a long message, or series of messages, the source symbols s_j would occur with a frequency corresponding to the associated probabilities p_j. Thus the average encoded symbol length would be the weighted sum of the codeword lengths. This value is usually referred to as the average length, L', of the code:-

$$L' = \sum_{j=0}^{N-1} p_j L_j$$

An ideal encoder would achieve an average length equivalent to the source entropy, thus ideally:-

$$L' = \sum_{j=0}^{N-1} p_j L_j \quad = \quad \sum_{j=0}^{N-1} p_j I_j$$

to which a solution is $L_j = I_j$. As the information content of a symbol may have non-integer values, this solution would require codewords with fractional length. Shannon showed that this could be circumvented by encoding groups of symbols, and hence that the average length of a code could be made arbitrarily close to the entropy.

The efficiency of a variable length code represents the degree to which the average length of the code approaches the entropy:-

$$\text{Efficiency} = \frac{H}{L'}$$

The redundancy of a code is also used to measure performance:-

$$\text{Redundancy} = 1 - \text{Efficiency}$$

The source code must be uniquely decodable, which means that an encoded message has a single unique possible interpretation. It is also desirable that the codewords are constructed so that any codeword may be decoded immediately it has been completely

received, i.e. instantaneously decodable.

A necessary condition (Abramson 1963) for a variable length code to be instantaneously decodable is given by the Kraft inequality:-

$$\sum_{i=0}^{N-1} D^{-L_i} \leq 1$$

where D is the radix of the code, and L_i is the length of the i-th codeword.

Instantaneously decodable codes (one class of uniquely decodable codes) within which lie prefix codes, suffix codes and others. Prefix codes are one of the most important classes of variable length code, and are defined by the prefix condition, which states that no codeword may be a prefix of any other codeword.

The design of a variable length source code consists of finding some set of codewords that meet the above criteria for decodability and can achieve an average length close to the source entropy. There may be additional design criteria, some of which will be discussed in later sections.

Shannon (1948) described a method for code generation in which a codeword is determined arithmetically using the following algorithm:-

(i) Arrange the symbols in order of decreasing probability, so that s_0 has the highest and s_{N-1} the lowest probability.

(ii) Determine the cumulative probability $P(\leq j)$ for each symbol s_j, i.e. the sum of the probabilities p_k for values of k from 0 to j.

(iii) The j-th codeword is given by the expansion as a binary number of $P(\geq j)$, the expansion being carried out to L_j places, where L_j is given by:-

$$I(\leq j) \leq L_j < 1 + I(\leq j)$$

An equivalent method is described by Fano (1949), hence the code is often referred to as the Shannon-Fano code.

An optimal prefix coding method was given by Huffman (1952), which is simple to implement, and generally achieves an average length very close to the entropy. The algorithm is generally related to the construction of a code tree, in which each node has q or less dependants, for a q-ary code. The method of construction is as follows:-

(i) Merge the q symbols or nodes having the lowest probabilities, to give a new node with a probability equal to

their sum.

(ii) Repeat (i), until one node, the root, remains.

(iii) Assign code symbols arbitrarily to the branches of the tree; codewords consist of the sequence of code symbols on the path between root and source symbol.

The excellent performance of this source code is demonstrated in Table 2.b. For one non-text, and four text samples the symbol frequencies were measured, and a Huffman code generated using the method given above. In every case the average length of the Huffman code is very close to the sample entropy.

Sample type	**Sample entropy (bits)**	**Average length (bits)**	**Efficiency**
Text 1	4.504	4.530	0.994
Text 2	4.641	4.670	0.994
Pascal 1	5.022	5.047	0.995
Numeric 1	3.988	4.049	0.985
Image 1	4.734	4.759	0.995

Table 2.b Comparison of average length of Huffman code with sample entropy.

2.4 Sources with memory.

Many real sources exhibit local dependence between message symbols. Various models have been used to represent this class of source, of which two will be considered. The first extends the principles used for the memoryless source, and is derived from the work of Shannon (1948, 1951), whilst the second model is based on the more recent development by Lempel and Ziv (1976) of a complexity measure for finite sequences.

The discrete memoryless source is defined in terms of the probabilities $p_0 .. p_{N-1}$ assigned to the symbols $s_0 .. s_{N-1}$. The model may be extended by considering the joint probabilities $p_{i,j}$ and the conditional probabilities $p_i(j)$ i.e. the probability of s_j occurring given that the preceding symbol was s_i. The set of conditional probabilities are equivalent to the transition probabilities of a Markov chain (Bartlett 1978), and hence this model is often referred to as Markovian.

The vector of symbol occurrence probabilities $p_0 .. p_{N-1}$ will, for an ergodic source, be the stationary vector of the Markov chain. A measure of the information carried by s_j given that s_i is the previous symbol is $I_i(j)$ which is defined as:-

$$I_i(j) = I_{i,j} - I_j$$

since $p_{i,j} = p_i \cdot p_i(j)$

and hence $I_{i,j} = I_j + I_i(j)$

The joint entropy $H_{i,j}$ measures the average information content of a digram source, whilst the conditional entropy $H_i(j)$ measures the equivalent for a Markov source. These two entropies are related by the expression:-

$$H_{i,j} = H_j + H_i(j)$$

For the memoryless source, it is assumed that $p_j = p_i(j)$. In general however, the conditional probability $p_i(j)$ is more than p_j due to dependence between characters, and hence the conditional information content $I_i(j)$ will be less than I_j . If the information content is reduced, the average code length may be shorter, hence it is generally advantageous to design source codes based on this type of model. The principal drawback is the larger number of parameters, N^2 rather than N.

Table 2.c shows the values obtained for information content and conditional information content from a sample of English text.

and illustrates the potential advantage of conditional encoding over that discussed in Sections 2.2 and 2.3.

In the table, $p_{i,j}$ gives the probability (in fact the observed frequency) of the symbol pair (s_i,s_j), I_j, the information carried by the second symbol s_j if considered independently, and $I_i(j)$ the conditional information content of s_j. The pair (*t*,*h*) for example, can be encoded ideally in 5.39 bits; the second symbol *h* would be encoded in 4.55 bits if the source were assumed memoryless, but in 1.63 bits if the code were based on the conditional probability, a saving of 2.92 bits.

Character pair (s_i,s_j)	$p_{i,j}$	I_j (bits)	$I_i(j)$ (bits)
(e,)	0.0341	2.77	1.77
(,t)	0.0264	3.76	2.48
(t,h)	0.0239	4.55	1.63
(h,e)	0.0223	3.11	0.94
(s,)	0.0197	2.77	1.57
(r,e)	0.0154	3.11	1.60

Table 2.c Comparison of independent and conditional information content of characters from English text.

Measured statistics have been published for a wide range of source types. Shannon (1951) gave first, second and third order conditional entropies for samples of English text. Barnard (1955) gave first order letter entropies for English, French, German and Spanish text. The entropy of Arabic was measured by Wanas (1976), of Portuguese by Manfrino (1969), and of Malay by Tan (1981). Equivalent statistics for a television image were published by Schreiber (1956).

Table 2.d gives the first, second and third order entropies for Arabic, English, and the television image, from the sources given above. A point of interest is the obviously high correlation of adjacent points in the television image, which results in fairly small increases in entropy, with increasing order.

Table 2.e compares the sample entropy, the entropy per symbol (the joint entropy represents two symbols), and the conditional entropy. It can be seen that the joint entropy per symbol is substantially less than the first order entropy, an average gain of 0.6 bits. The conditional sample entropy gains further, and achieves an average improvement of 1.21 bits over the first order entropy. The results indicate that an encoding scheme based on the conditional probability should be more efficient than one based on the symbol or digram probability.

Sample type.	Sample entropy		
	1st	2nd	3rd order
	bits per (1)	(2)	(3) symbols
Arabic	4.21	7.98	10.47
English	4.03	7.35	10.45
Television signal	4.39	6.30	7.80

Table 2.d Sample entropy for source models based on p_i, $p_{i,j}$, and $p_{i,j,k}$

Sample type.	Sample entropy (per symbol)		
	1st order	joint	conditional
	H	$H_{i,j}/2$	$H_i(j)$
Arabic	4.21	3.99	3.77
English	4.03	3.67	3.32
TV signal	4.39	3.15	1.91
Average	4.21	3.61	3.00

Table 2.e Comparison of conditional and joint sample entropy.

The source model may be based on conditional or joint probabilities and, as shown above, it is generally advantageous to use a higher order model. In the case of text and many types of sampled analogue data, the context of a symbol is usually not of fixed size, as this model assumes. It would seem therefore that an alternative model, which allows a variable context size, would be more appropriate.

Shannon (1951) discussed the use of a word based, rather than letter based encoding for text. Estimates of the word entropy for a sample of English text indicated that a word based source encoder should achieve an average length of 2.1 to 2.6 bits per letter. The encoding would map variable length sequences of source symbols (words) onto variable length codewords.

One important class of source model which incorporates a variable symbol context size, is known as fragment encoding or variety generation (Cooper 1982, Yannakoudakis 1982). A fixed number of equiprobable strings of symbols are found; as these are equiprobable and hence have equal information content, they may be efficiently encoded using codewords of equal length. The number of fragments is generally selected to be some integer power of two, to avoid loss due to the need to round up fractional codeword lengths.

For English text, typical fragments from a set of 256, are "*in*" "*the*", "*that*", "*atio*", "*with*", and "*ght*". They consist of frequent sections of words, and common words or groups of words. The main

problem found with the model is the complexity of the process of building the fragment set. Cooper (ibid) discusses several approaches to set production, for example processing a large sample of text to determine the occurrence frequencies of characters, character pairs (digrams), triples (trigrams), up to some limit, and then selecting the set from amongst these. For a 64 character alphabet, there are 4,096 digrams, 262,144 trigrams, and 16,777,216 tetragrams; the processing involved is obviously not trivial.

An alternative approach to source modelling devolves from the approach to the measurement of sequence complexity suggested by Lempel and Ziv (1976). The complexity of the sequence of source symbols is evaluated with a simple learning machine, which scans the sequence once, matching strings of symbols to those stored in its memory, to which is appended any new string of symbols encountered en route. The size, and rate of growth of the compiled vocabulary form the basis of the complexity measure.

The initial vocabulary of the machine consists of the source alphabet; additional entries will be strings of two, three or more symbols. In its simplest form, the machine finds the longest match to the current subsequence of symbols, and then forms a new vocabulary entry by appending the next symbol in the sequence to the matched subsequence. This process has been termed *incremental parsing*, and provides an automatic context gathering method ideally suited to source modelling.

2.5 Coding schemes for sources with memory.

Two classes of code, corresponding to the two types of source model, are commonly used. The first extends the principles of variable length coding, as described for the DMS, to the Markov model, whilst the second group of coding techniques are applied to fragment encoding.

In the preceding section it was shown that the use of conditional and joint probabilities, rather than symbol probability, offers some advantage in an ideal encoding. This leads to the use of two alternative encoding methods:-

(i) Digram encoding, in which variable length codewords are constructed (using Huffman's algorithm for example) using the joint symbol probabilities $p_{i,j}$.

(ii) Conditional encoding, in which variable length codewords are constructed using the conditional symbol probabilities $P_i(j)$.

As the joint symbol probability $p_{i,j}$ consists of the product $p_i \cdot p_i(j)$, the first symbol is encoded using approximately I_i bits, the second with $I_i(j)$ bits. This is less efficient than the conditional encoding scheme, in which only the first symbol of the message is encoded inefficiently, all succeeding symbols being encoded with

$I_1(j)$ bits (on average). Table 2.e showed that definite gains could be made through the use of conditional encoding on a range of data samples, and the same number of codewords are required to encode a message using either joint or conditional schemes, it would therefore seem preferable to use conditional encoding.

Lavelle (1981) proposed an adaptive variable length coding scheme using conditional encoding. Results given for a text sample indicate that a Huffman code based on a DMS model achieved an average length of slightly below 5 bits per symbol, whereas the adaptive conditional coding scheme gave an average length of less than 3 bits per symbol.

The second type of source code, used for fragment or string coding, aims to encode equiprobable sequences of symbols which, having equal information content, may be encoded with equal codeword length. In the discussion of this type of source model (Section 2.4), the problem of context gathering was mentioned, and the Lempel-Ziv complexity measure outlined. In fact, the complexity measure provides the basis for a powerful family of source codes. In Ziv and Lempel's 1977 paper "*A Universal Algorithm for Sequential Data Compression*", the method is given.

The basic Ziv-Lempel encoder has a dictionary, in which each entry has an associated index number. Initially the dictionary contains only the basic alphabet of the source; during the encoding process new dictionary entries are formed by appending single

symbols to existing entries.

Let *E*(i,n) be a string of source symbols exactly matching the dictionary entry with index i and length n, and let s_l be the next source symbol in the input sequence. Symbol s_l is read and appended to *E*(i,n), giving an extended string *E*(x,n+1). The dictionary is searched and, if *E*(x,n+1) is matched with some entry with index j, then with *E*(j,n+1) the next source symbol is read. If *E*(x,n+1) is not found, the pair (i, s_l) is transmitted and the string *E*(x,n+1) added to the dictionary.

For each transmitted pair (i, s_l), an average of n' symbols (where n' is the average encoded string length) will be read from the source, and the dictionary size increased by one. The compression obtained is therefore:-

$$\text{compression ratio} = \frac{T + s}{n'.\ s}$$

where T is the number of bits required to identify a dictionary entry, i.e. the logarithm of the dictionary size, and s is the number of bits required to identify an uncompressed symbol.

Assuming fixed values for T and s, the rate of dictionary growth with number of input symbols is equal to the compression ratio. This algorithm will be discussed more fully in the next chapter.

2.6 Run Length Encoding.

This is a very widely used method of data compression, which makes few assumptions about the nature of the source (Gottlieb 1975). It is assumed that the sequence of source symbols contains runs of some symbols (i.e. subsequences containing only one type of symbol repeated several or many times). The encoder outputs a short, fixed length sequence of codewords corresponding to a variable length input run, but otherwise does not affect the message contents.

Various models have been proposed, which attempt to model this type of source. For example, a Markov process having large probabilities on the major diagonal of its transition matrix will produce runs. As an alternative, a discrete distribution (for example geometric or Poisson) may be used to directly specify the probabilities of given run lengths occurring.

The encoding process may be of three types:-

(i) An explicit codeword for each run length.

(ii) A codeword for certain run lengths. For simple encoding, the lengths could be integer powers of two, although more correctly the length distribution should be determined by the probability distribution. This would be suitable for a binary source which produces only runs of 0's and 1's.

(iii) A single control character *run*, which is used to indicate

that the following codeword is to be interpreted as a numerical run length value. For example, "xxxxxxx" could be encoded as ("x",*run*,7). This method is more appropriate to data containing occasional long runs, perhaps for the removal of trailing nulls or spaces on computer files.

For short run lengths the first two methods are preferable, although the three element method (iii) performs better for longer sequences.

Run length encoding may be concatenated with other coding schemes. For example the CCITT facsimile encoding standard T.6 specifies run length encoding followed by a modified (predefined) Huffman code. Codewords are allocated to run lengths from 0 to 64, and then in steps of 64 up to 1728. Any run length in the range 0 to 1728 can be encoded with, at most, two codewords.

Concatenation may increase the entropy of the message, both by reducing the frequency of the symbols encoded, and the introduction of the additional codewords needed. Table 2.f illustrates the effect of run length encoding on the number of symbols, sample entropy, and encoded message length for eight samples. The sample *Numbers* illustrates the effect described above particularly well, a large gain from run length encoding is offset by an increase in entropy, resulting in a lower degree of overall compression.

In general however, run length encoding is a simple and practical compression method which may be concatenated with other coding techniques.

Sample	Sample size (number of symbols)			Sample Entropy		Effect of concatenating run length encoding with variable length encoding (% reduction in encoded message length)
	Before Run Lngth Encoding	After Run Lngth Encoding	Reduction in sample length	Before Run Lngth Encoding	After Run Lngth Encoding	
Text1	40598	40268	0.8%	4.504	4.520	0.5%
Text2	44685	35348	20.9%	4.641	5.061	13.7%
Text3	13873	13768	0.8%	4.456	4.430	1.3%
Text4	48753	46920	3.8%	4.722	4.815	1.9%
Fortran1	5387	5370	0.3%	5.281	5.290	0.1%
Fortran2	1971	1896	3.8%	4.773	4.893	1.4%
Numbers	2736	1981	27.6%	3.988	5.041	8.5%
Image	65664	60803	7.4%	4.734	4.864	4.9%

Table 2.f The effects of Run Length Encoding on the number of symbols length, sample entropy and encoded length of eight samples.

2.7 Discussion.

The preceding sections have introduced source modelling and coding under certain assumptions. Firstly that the source is known and stationary, secondly that the transmission channel is error free, and thirdly that the complexity of the encoder/decoder is not important. The first of these complications will be dealt with in the next chapter, but the remaining issues are discussed below.

Three basic types of source model have been introduced, the Discrete Memoryless Source, the Markov source, and the string producing source (with particular reference to the Lempel-Ziv complexity measure). Consideration of these different source models showed how a number of different coding schemes could be developed, and some idea of performance was given.

The Discrete Memoryless Source is based on the occurrence probabilities of the source symbols. Shannon (1948) showed that a good source encoder could be designed for this class of source, whilst Huffman (1952) developed an algorithm for generating an optimum code.

Sources which exhibit dependence between symbols may be modelled as Markov processes, which allows the design of more efficient encoders than those based on the DMS. Two alternative coding schemes based on the Markov model were examined, and it was shown that the conditional probability formed the best basis for code generation.

Although Markov models achieve good performance for sources with memory, the dependence between symbols in a message is usually over a variable number of symbols. For example in text, there is strong dependence between characters in words, the word size however is not fixed. Variable to fixed encoding schemes were discussed, and the Ziv-Lempel compression algorithm introduced.

Another compression technique that is widely used is run length encoding, in which repeated occurrences of some symbol are replaced by a two or three codeword sequence. The method is suitable for concatenation with other source codes, but there is some degree of interaction.

Several important points have been omitted from the discussion so far.

(i)Transmission errors.

Errors introduced between source encoder and decoder will cause corruption of one or more decoded symbols. For variable length encoders, if the transmitted and corrupted codewords are of the same length, then only one symbol will be affected. If however, the transmitted and corrupted codewords are of different lengths, the decoder will lose synchronization with the encoder, resulting in a series of incorrect output symbols. Careful design can produce codes which will resynchronize quickly, as discussed by Stiffler (1971) and Ferguson and Rabinowitz (1984).

Maxted and Robinson (1985) developed a finite state model for the analysis of synchronization recovery for a variable length code. They found that most codes do resynchronize quite quickly after an error; for two different Huffman codes generated for a 26 symbol English character source, recovery occurred within three to seven symbols of an error. It is stated that "*one must work diligently to construct codes with a long recovery span*", however an example of a poor code is given, which took up to 62 symbols to resynchronize.

The conditional coding scheme proposed in Section 2.4 is likely to result in greater error extension, as the decoding of each codeword is dependent on the correct decoding of its predecessor. This is similar to the problem of error propagation in predictive encoding systems, as discussed by Maxemchuk (1979).

Variable to fixed length codes, such as the string encoding methods, are less susceptible to errors. Transmission errors will result in the incorrect decoding of a single codeword and, although this will result in the corresponding string being corrupted, no loss of synchronization will occur. Loss of synchronization could however occur if a bit were inserted or deleted, due to timing instability in some part of the transmission path. This would extend almost indefinitely, whereas the variable length codes would resynchronize fairly quickly.

(ii) Complexity.

Although the performance of some coding schemes may be excellent, the memory, hardware, or processing time requirement may be excessive. A variable length code, designed for a source with an N symbol alphabet, may require codewords from 1 to (N-1) bits in length.

Software and hardware implementation of a variable length encoder/decoder is not simple, due to the bit oriented nature of the data. Wells (1972) discusses hardware implementation of a Huffman encoder and decoder, whilst Schwartz and Kallick (1964) give an algorithm for software implementation.

For digram encoding, requiring N^2 codewords, the possible codeword length range is from 1 to (N^2-1) bits, and it becomes necessary to limit the maximum length of the code. Garten (1985), Humblet (1981), Lavelle (1981) and Van Voorhis (1974) discuss ways in which this can be accomplished; these techniques will be further discussed in the next chapter.

Variable to fixed length codes have some advantage, as the processing tends to be character rather than bit oriented. Other problems are encountered however, such as the need for extensive memory capacity; for example, the scheme proposed by Cleary and Witten (1984) required up to 1.4 megabytes of storage. In addition, the encoder performs a string matching operation based on a dictionary search, which is generally complex.

A number of other codes exist, amongst which the most prominent are *arithmetic codes* (Langdon 1984). These treat codewords as binary fractional values, generated by successive subdivision of the interval (0,1) using the cumulative probabilities of the symbols. The performance of these codes is bounded, as with Huffman codes, by the entropy.

This chapter has considered some of the basic source coding techniques and their background in information theory. Comparative results have been given, and some of the practical problems discussed. The next chapter will consider some of the more practical issues, in particular the encoding of non-stationary sources.

3. CODING PARTIALLY KNOWN SOURCES.

3.1 Introduction.

Many of the practical problems associated with source coding are related to the degree with which the source model matches the actual source; hence the design of coding schemes which are well behaved, for sources that are not, is of considerable importance. Several features of real sources must be considered:-

(i) The source may not match any realizable model well enough for a practical encoder/decoder to be designed.

(ii) The source model parameters are not known, and must be estimated from previous messages.

(iii) The source may be non-stationary.

(iv) The source may only exist for a finite period of time, i.e. produce a single output sequence.

Some of these points have been considered by Gilbert (1971), who proposed a number of variable length coding techniques for inaccurately known sources. These and other techniques for designing codes that are not sensitive to source instability are discussed in Section 3.2.

Non-stationary sources cannot usually be effectively compressed using a static coding scheme. The use of an adaptive encoder, which continuously estimates the source parameters and hence maintains a near optimal code, is discussed in Section 3.3. Faller (1974), for example, suggested an adaptive variable length coding scheme, which is discussed together with the later method of Gallager (1978).

The coding scheme of Ziv and Lempel (1977) is inherently adaptive and, although more complex than the adaptive variable length encoders, can usually achieve significantly better performance. Other adaptive string encoding schemes have been suggested (by Cleary and Witten, 1984 and others).

The main theme of this chapter is the selection of practical source coding techniques for partially known sources, i.e. sources for which a model is known or assumed, but the model parameters are unknown or inaccurate. Considerations such as performance on non-stationary sources, memory requirements, and ease of implementation will be discussed.

3.2 Designing robust variable length codes.

If the symbol occurrence probabilities for the source are known, an optimal code can be generated using Huffman's algorithm (Section 2.3). If however, the probabilities are not precisely known, the optimality of a code becomes difficult to measure. Some of the characteristics of a robust code are low average length, small change in average length with deviations in source symbol probability, and an absence of transient effects which may cause buffer overflow.

In Section 2.3 it was stated that the length of a codeword should ideally be close to the information content of the symbol represented. For the Shannon-Fano code, the following relation holds (by definition):-

$$I_x \leq L_x < I_x + 1$$

In the case of a symbol with small probability, the corresponding length will be large. A source of N symbols whose probability follows a negative exponential distribution would, if encoded using a Huffman code, result in a maximum codeword length of (N-1) bits.

If the probabilities of the symbols are estimated from previous (finite length) messages, the smaller values, which will generate long codewords, will be less accurately known than the larger probabilities. Some symbols may not have occurred in the samples used for measurement, and will be represented as having zero

probability (and a theoretically infinite codeword length).

Hamming (1980) discusses the effects of uncertainty in source probabilities on Huffman code performance. A source with known symbol probabilities p_i, is used to generate a Huffman code with corresponding codewords of length L_i. The actual source has symbol probabilities p_i' . The average symbol length obtained is:-

$$L' = L + \Delta L = \sum_{i=0}^{N-1} L_i.p_i + \sum_{i=0}^{N-1} L_i.e_i$$

where L is the average length of the code for the original source, and

$$e_i = p_{i'} - p_i$$

This is developed by Hamming to show that:-

$$\Delta L = \sqrt{\text{variance of } L_i \;\; x \;\; \text{variance of } e_i}$$

hence showing that a larger variance of the codeword length distribution will exacerbate the effects of errors in the estimates of symbol probabilities.

A code may be made robust by limiting the maximum

codeword length. This assists in several respects; the worst case performance is limited by the predefined maximum length, the zero frequency problem is circumvented, and the maximum length can be selected to suit buffer or register widths (for example 16 bits), or other hardware considerations (Garten 1985). Unfortunately, it is fairly difficult to design codes with minimum average length subject to a maximum length constraint. The solution has been formulated as an integer programming problem by Karp (1961), although the approach is computationally complex.

As an alternative, the maximum length may be reduced, but not constrained. This does not provide the same level of security as the former method, nor does it give a definite bound on buffer size, beyond the trivial case (of N-1 bits for an N symbol code). The two classes of technique will be considered separately, and the compromise between average and maximum length examined.

3.2.1 Generating variable length codes with reduced maximum length.

Huffman's algorithm can produce a number of different codes for a given probability distribution, due to arbitrary decisions that are made at certain stages. An improved algorithm was given by Schwartz and Kallick (1964), which produces the Huffman code with the minimum longest codeword length. The original algorithm, given an ordered list of probabilities, repeatedly finds the lowest D probabilities (where D is the code radix), merges them, and places

the merged probability in order. Given a number of equal probabilities, Huffman's algorithm gives no rule for selecting the values to be merged. Schwartz and Kallick found that the minimum longest codeword resulted if the following rules were applied:-

(i) Select probabilities from the bottom (low probability end) of an equiprobable set.

(ii) Place merged probabilities above (at the higher probability end of) any existing probabilities of equal value.

An alternative approach given by Gilbert (1971), is to modify the source probability distribution. This may be carried out in several ways, by merging two probability distributions, or by altering the estimated distribution. Gilbert suggests that the source is regarded as composite, and the overall distribution of symbol frequencies calculated from the proportion of time spent in each source state. For example, the source may spend five percent of the time with a uniform, and ninety five percent with a negative exponential distribution.

The source may be modified without altering the entropy, by assigning infinitesimal probabilities to those symbols with zero frequency. As the product p.log(1/p) tends to zero with decreasing p, the entropy will not be affected by the operation.

Table 3.a shows two different sets of Huffman codes, for four

data samples. The first set of codes is generated using the basic source probability distribution, whilst the second set is based on modified distributions obtained using the method given above. The reduction in maximum length obtained by the modification is considerable, whilst no change in average length is observable.

Sample	Huffman code performance			
	Using source PDF		Using modified PDF	
	Average length	Maximum length	Average length	Maximum length
Text1	4.53	61	4.53	21
Text2	4.67	63	4.67	20
Text3	4.48	73	4.48	21
Image1	4.76	98	4.76	23

Table 3.a Average and maximum codeword lengths for Huffman codes based on source and modified source probability distributions.

If upper and lower bounds can be given for the symbol probabilities, a method given by Smith (1974) can be used to find a code that minimizes the largest average codeword length for all probability distributions within the bounds.

The code is designed using Huffman's algorithm on a modified *compromise* probability distribution. The distribution is obtained from the upper and lower bounds p_l^U, p_l^L of the probability for each

symbol s_i:-

$$p_i = Z \qquad \text{if } p_i^L < Z < p_i^U$$

$$= p_i^L \qquad \text{if } p_i^L \geq Z$$

$$= p_i^U \qquad \text{if } p_i^U \leq Z$$

where Z is determined from the condition that the p_i's must have a sum of 1.

Humblet (1981) formulated an iterative algorithm to produce a prefix code that minimizes the moment generating function of the codeword length distribution. The algorithm of Huffman is modified slightly, by introducing a scaling operation applied after each merge step.

3.2.2 Variable length codes with constrained length.

If a length constraint, L_{con}, can be applied to a variable length code, the worst case performance is limited, the robustness improved, and hardware or software implementation made easier. Inevitably, the constraint can only be applied at the cost of increased average length. A loose upper bound for the performance of a code of this type is given by Gilbert (op cit), this is given as:-

Upper bound on average length

$$L^U \leq H + 1 + N\, 2^{(1-Lcon)}\ (\log(N) - H)$$

The code may be generated in several ways, Karp's (ibid) integer programming solution is not attractive due to the inherent complexity of the algorithm, although an optimum solution will be found. Some alternative approaches are:-

(i) modify the symbol probability distribution.

(ii) use an iterative method such as dynamic programming.

(iii) use a standard codeword length distribution, assigning codewords to symbols on the basis of symbol probability.

3.2.2.1 An approach based on a modified source pdf.

This method has been briefly examined in Section 3.2.1, where it was shown that the replacement of zero frequencies by very small values substantially reduced the maximum length of a Huffman code without significant effect on the average length. If the information content of the symbols with lowest probability is less than or equal to the maximum allowable length (L_{con}), it should be possible to generate a code with average length close to the entropy,

and maximum length equal to the constraint.

If some number m of the least probable symbols are selected, such that the average information content of the m symbols is equal to, or less than the constrained length, these may be encoded in at most L_{con}+1 bits. The entropy of this source is therefore:-

$$H_m = \sum_{i=0}^{N-m-1} p_i \log\left(\frac{1}{p_i}\right) + m.P(av).\log\left(\frac{1}{P(av)}\right) \quad \cdots$$

where the smallest m is selected such that

$$P(av) = \frac{1}{m} \sum_{i=N-m}^{N-1} P(i)$$

and $$\log\left(\frac{1}{P(av)}\right) \leq L_{con}$$

If a Shannon-Fano code is generated on the modified probability distribution, the maximum length obtained will be in the range L_{con} to (L_{con} +1), as the length of a codeword is within one bit of the information content. For a Huffman code, an upper bound to the maximum codeword length (for the symbol with the minimum probability p_{min}) is given by Katona and Nemetz (1976); the maximum length is bounded by s-1 where:-

$$\frac{1}{F(s+1)} \leq p_{min} < \frac{1}{F(s)}$$

where F(s) denotes the s-th Fibonacci number, generated from the

relation F(1)=1, F(2)=1, F(s)=F(s-1)+F(s-2), (Hardy 1938).

This method does not guarantee that the maximum codeword will be L_{con} , although it does constrain the codeword length to within some known bound. Ideally however, a code is required that has no codeword longer than L_{con} . An iterative approach may be used, in which the value of m is successively increased, and a code generated, until the value L_{con} is not exceeded. Although the number of iterations is likely to be small, the repeated code generation will be computationally expensive.

3.2.2.2 An iterative approach.

Integer programming, as suggested by Karp (ibid), will produce an optimum code, i.e. a code with a minimal average length subject to a length constraint. However the complexity of integer programming, even with the branch and bound method of solution, is known to be high. An alternative approach, similar to the method suggested by Van Voorhis (1974), is to use dynamic programming (Cooper 1981).

The code generation process may be implemented in an iterative fashion, at each stage optimizing a single variable subject to the given constraints. The variable in this instance is the number of codewords of some given length, which are selected to give a minimum average code length, subject to the maximum length

constraint and the Kraft inequality (sect 2.3)

The probabilities of the symbols s_k are arranged in decreasing order. The initial codeword length is set to one bit, and the following procedure used repeatedly until all symbols have been assigned codewords.

(i) Select the number of codewords of length i bits, n_i, to minimise the expected average length L'', subject to (ii); given that the j symbols already assigned codewords are encoded using the allocated codeword lengths, the n_i codewords are encoded in i bits, and unassigned symbols are encoded under the assumption that they are equiprobable.

(ii) n_i is constrained by two requirements, firstly that the Kraft inequality is satisfied, and secondly that the number of bits required to encode all remaining symbols must not be greater than the given maximum length L_{con}.

(iii) Assign n_i of the x_i available codewords of length i bits, set j to j+ n_i , i to i+1 bits, and repeat (i).

The number of codewords available of length i bits is:-

$$x_i = 2.\ (\ x_{i-1} - n_{i-1}\ )$$

If n_i codewords, of the x_i available, are assigned, it is possible to encode the $(N - j - n_i)$ remaining symbols with codewords of length i+r bits, where r is given by:-

$$2^r . (x_i - n_i) \geq (N - j - n_i)$$

this gives

$$r \geq \log_2\left(\frac{(N - j - n_i)}{(x_i - n_i)} \right)$$

Provided that (i+r) is not greater than the required maximum length then the code will be able to satisfy the length constraint. The optimizing function is the expected average length L''_i, given by:-

$$L_i'' = \sum_{k=0}^{j-1} p_k.L_k + \sum_{k=j}^{j+n_i-1} p_k.i + \sum_{k=j+n_i}^{N-1} p_k.(i+r)$$

$$= \sum_{k=0}^{j-1} p_k.L_k + \sum_{k=j}^{N-1} p_k.i + \sum_{k=j+n_i}^{N-1} p_k.r$$

As the minimization procedure will take significant processing time, a simple precondition may be added to check that an optimal stage solution is likely. The test compares the marginal information content of the y-th symbol (defined below), with the

marginal information capacity of the x_i available codewords.

Marginal information $M_y = \log_2(R_y/p_y)$ bits

content of y-th symbol

N-1

where R_y is the sum $R_y = \sum_{k=y} p_k$

The marginal information carrying capacity

of an i bit codeword is:- $M_y' = \log_2(x_i)$ bits

Attempt optimisation if $M_y < M_y' - \partial$

where ∂ is determined experimentally.

This method cannot be shown to provide an optimal solution unless the problem can be classified as ṡeparable (Cooper op cit), although it will be shown to provide reasonable performance.

3.2.2.3 A simple code with a fixed codeword length distribution.

An alternative technique, using a predetermined set of codeword lengths, is extremely simple to implement. For example, a code length set for a 128 symbol alphabet, satisfying the Kraft

inequality is:-

length of codeword	4bits	8 bits
number of codewords	8	120

To generate a code, it is necessary only to allocate the eight most frequent symbols 4 bit codewords, and allow 8 bits for the remaining symbols. Encoding consists of searching a short table, each entry containing one of the eight most frequent characters. If the current character is not found, it's binary equivalent (for example ASCII code) is prefixed by a '0' and sent. If the character is found in the table, the number of the entry prefixed by a '1' is transmitted. Decoding is even simpler, if the received codeword is prefixed by '0', the remainder of the codeword is the character code; if the codeword is prefixed by '1', the remainder of the codeword forms an index to an equivalent table to that used by the transmitter.

3.2.2.4 Performance comparison of the constrained VL codes.

Table 3.b shows the average lengths of codes with a maximum length of 8 bits, generated using the three methods outlined above, the iterative technique, a Huffman code generated on a constrained source, and the simple 4/8 code. The average length obtained for the different samples do not show that one of the methods is markedly better than the others, however the iterative technique

provides a more consistent performance. Table 3.c shows the ranking of the three methods over all twelve samples, and supports this conclusion.

Sample name.	**Sample entropy.**	**Constr. source.**	**Iterative method.**	**4/8 bit code.**
Text1	4.504	5.680	**5.355**	5.507
Text2	4.641	5.603	**5.440**	5.593
Text3	4.456	5.664	**5.324**	5.526
Text4	4.722	5.898	**5.530**	5.719
Text5	4.019	5.375	5.283	**5.202**
FORTRAN1	5.281	6.122	**6.074**	6.415
FORTRAN2	4.773	5.861	**5.576**	5.944
Pascal1	5.022	**5.690**	5.799	5.938
Pascal2	4.420	**5.344**	5.471	5.602
Prolog	4.747	5.822	**5.640**	5.818
Numbers	3.988	5.355	**5.142**	5.246
Image	4.734	6.008	**6.000**	6.149

Table 3.b. Average length (bits) of three variable length codes with a constrained maximum length of 8 bits.

Ranking	Number of occurrences of rank: Constrained source.	Iterative method.	4/8 fixed structure.
1	2	9	1
2	3	3	6
3	7	0	5

Table 3.c. Summary of scores attained by coding schemes from Table 3.b

The performance of the iterative and fixed structure codes is compared with that of a Huffman code in Tables 3.d to 3.f., for four data samples of two different types. A code is generated on one sample of data from the four, and used to encode all four samples. This illustrates the tolerance of the code to minor (i.e. different sample of the same type of source), and major (i.e. sample from a different type of source) changes in source parameters. Table 3.g. combines the results from Tables 3.d. to 3.f., and generally supports the preceding discussion on robustness. The Huffman code achieves best performance for sources of the same type, but worst performance for sources of different types. The two robust coding schemes, both with a constrained maximum length of eight bits, are less efficient than the Huffman code for sources of the same type, but are more tolerant to larger changes.

The performance difference between codes generated by the

iterative (dynamic programming) method, and the simple fixed structure code is not large, and the simplicity of code generation and ease of handling four and eight bit codewords render the latter attractive for small or fast systems.

Code applied to sample	FORTRAN		Pascal	
	1	2	1	2
Code generated on sample:				
FORTRAN 1	**0.99**	0.93	0.65	0.61
FORTRAN 2	0.88	**0.99**	0.44	0.42
Pascal 1	0.71	0.71	**0.99**	0.94
Pascal 2	0.64	0.64	0.90	**0.99**

Table 3.d Efficiency of Huffman code when applied to sources other than that used for code generation.

Code applied to sample	FORTRAN		Pascal	
	1	2	1	2
Code generated on sample:				
FORTRAN 1	**0.87**	0.79	0.72	0.64
FORTRAN 2	0.87	**0.86**	0.73	0.67
Pascal 1	0.69	0.64	**0.87**	0.77
Pascal 2	0.69	0.65	0.82	**0.81**

Table 3.e Efficiency of code generated using the iterative method, when applied to sources other than that used for code generation.

Code applied to sample	FORTRAN		Pascal	
	1	2	1	2
Code generated on sample:				
FORTRAN 1	**0.82**	0.75	0.74	0.69
FORTRAN 2	0.79	**0.80**	0.73	0.70
Pascal 1	0.69	0.64	**0.85**	0.76
Pascal 2	0.69	0.64	0.82	**0.79**

Table 3.f Efficiency of the 4/8 bit fixed structure code when applied to sources other than that used for code generation.

	Efficiency for source:-		
Code type:	**Used to generate code**	**Same type**	**Different type**
Huffman	**0.99**	**0.95**	0.61
Iterative method	0.85	0.83	0.68
Fixed structure	0.81	0.80	**0.69**

Table 3.g. Average value of efficiency of coding schemes for different classes of source (from previous Tables).

3.3 Adaptive Source Coding.

Although a robust code will perform well for slight variations in source symbol probabilities, it will not provide acceptable performance for gross source changes. The use of adaptive source coding is practical for non-stationary sources, if the rate of change of source model parameters is low.

In principle, adaptive coding is a logical extension of the static coding schemes already discussed. A source model is defined, and the parameters of the model estimated from observed data. A code is generated, based on the model parameters. The parameters are continually updated, and new codes generated at intervals. Thus the code is maintained near to the optimum for the source model.

Many of the inherent problems of source coding are exacerbated by the use of adaptive encoders. For example, a code which is complex to construct may be practical for a static encoder, but the processing overhead in periodically regenerating it may be excessive, rendering the method impractical for an adaptive system.

Two types of adaptive source code will be discussed, the adaptive variable length code, and the adaptive string (or variable to fixed) code.

3.3.1 Adaptive Variable length coding.

An adaptive variable length encoder consists of the basic blocks shown below. Source symbols are read and encoded using a *current* code. As the symbols are read from the source, they are used to update a frequency table. In the case of a discrete memoryless source model, the symbol frequencies are used to estimate the symbol probabilities, whilst for a Markov model, the joint or conditional probabilities are estimated.

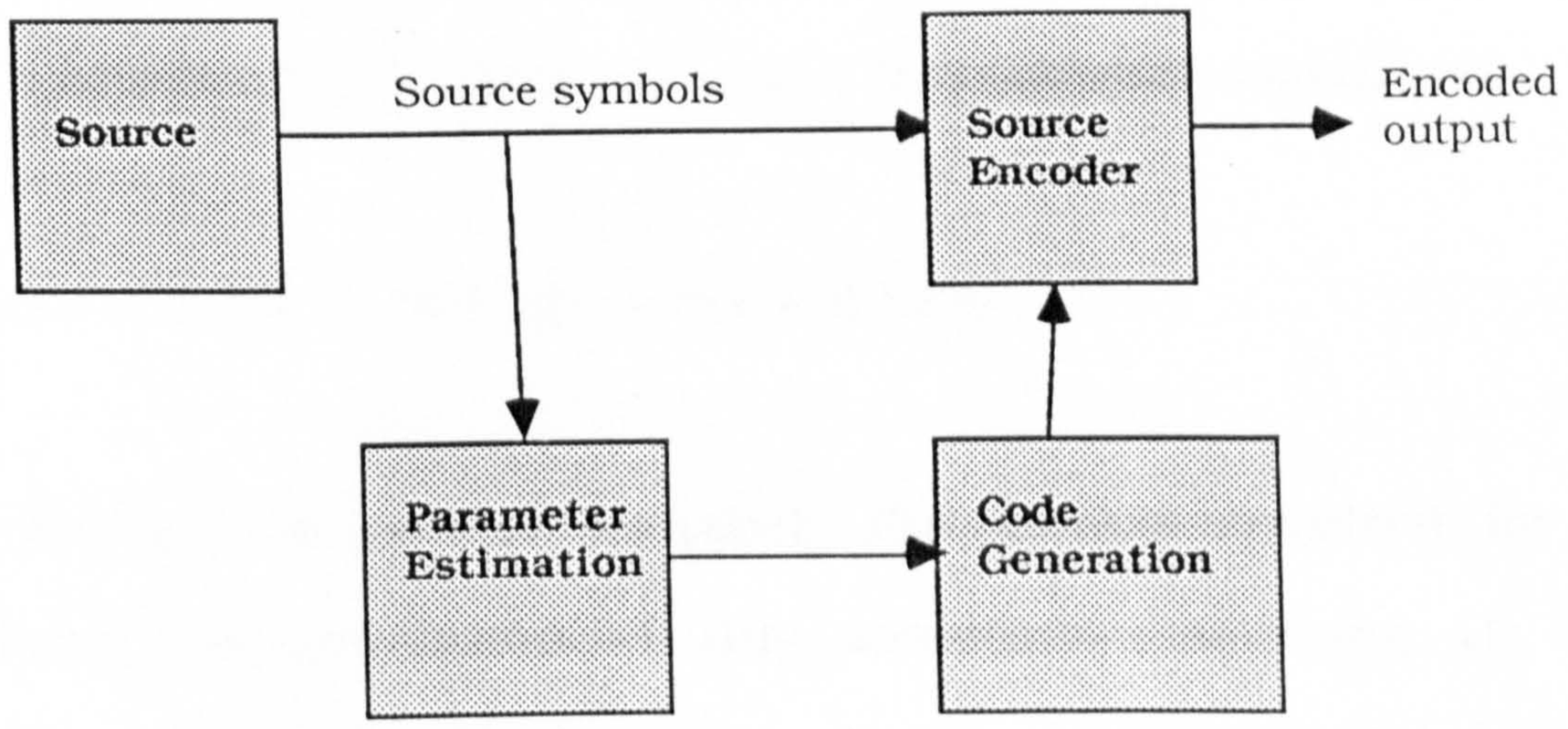

Figure 3.h An adaptive source encoder

A number of variations are possible but complexity, performance, rate of change of source characteristics, and other practical design constraints need to be considered. Initially, two designs proposed by Gallager (1978) and Faller (1974) will be discussed, some alternative schemes will then be proposed, and the design considerations outlined.

Gallagers adaptive Huffman encoder employed a tree form of Huffman's algorithm. A node consists of the 6-tuple:-

(i) pointer to parent node,

(ii) 0 or 1 bit element of a codeword,

(iii) pointer to left dependent node,

(iv) cumulative frequency for left node,

(v) pointer to right dependent node,

(vi) cumulative frequency for right node

In addition, for each source symbol there is a pointer to the node in which the symbol is represented, and a bit to indicate whether the symbol is on the left or right hand branch. The total storage required for the data structure, for N source symbols is:-

N.(1 word + 1 bit) + (N-1).(5 words + 2 bits)

For a 128 symbol alphabet, the storage required for a reasonable representation of this structure (assuming 16 bit addresses) would be 1654 bytes. For digram or conditional encoding based on a 128 symbol alphabet, the memory requirement would be 114,683 words (32 bit words would be needed, giving 459 kilobytes total requirement).

As a symbol is read from the source, the pointer associated with the symbol is used to find the appropriate node, and the left/right-hand bit used to determine which pointer/ frequency pair to update. The frequency count is incremented, and then the pointer to the parent used to propagate the count increase up the

tree. The nodes are resorted into frequency order (the sum of left and right hand frequencies), and paired. By the sibling property (Gallager 1978), the resulting tree structure forms a Huffman tree, and can be used to represent the corresponding Huffman code.

To prevent frequency counter overflow, and give a fairly fast response to source changes, a scaling operation is periodically applied to the measured frequencies. Gallager suggests that, every M symbols, the frequencies are multiplied by a factor ß, typically 0.5.

The code in this case is updated once for every symbol encoded, which results in a non-trivial processing requirement. For a fairly stable source the code will change infrequently, and the re-sorting steps will take little time.

Faller's (1974) encoding scheme is broadly similar to the later method described above. The scaling operation, division by two, is applied when the average length has increased on v occasions. Faller gives results for four data samples, and for three values of v. The average length, normalized to the static Huffman code performance, taken over the four samples was calculated by Faller to be 0.993, *less* than the average length of a static Huffman code on the same data. An adaptive code may perform better than a static code if a source sequence can be subdivided such that the average subsequence sample entropy (weighted by the subsequence length), is less than the entropy of the whole sample; or if the static code is not optimal for the source.

The fixed structure 4/8 code may also be used within an adaptive encoder. This requires a table of character frequencies as with the Huffman codes described above, however the accuracy of the frequencies has less effect on the code performance, hence one byte per character is sufficient. The total memory required for a 128 symbol alphabet is 128 bytes for the frequency table, and eight bytes for the table of short codewords (described in Section 3.2.2).

It was assumed by both Faller and Gallager that the receiver and transmitter independently track the source parameters and generate codes, however other approaches may be adopted:-

(i) Independent transmitter and receiver code generation.

This method, described above, has the advantage that the channel capacity is used only for transmitting encoded data. If transmission errors occur however, the receiver, which is relying on the decoded data for the information necessary to update its code, may lose synchronization. In addition, since both transmitter and receiver must maintain parameter estimation and code generation, the complexity is fairly high.

(ii) Code generated at the transmitter and downloaded to the receiver.

This method is fairly practical as long as the code download is infrequent, as it occupies useful channel capacity. For example, the transmitter may generate a code and download it to the

receiver. The average length of the code is monitored, and if the performance deteriorates beyond some threshold, a new code is generated. If the source is highly variable, the code download will become more frequent and the resulting loss in efficiency may be unacceptable, therefore this technique will not be considered further.

(iii) Code generated at the transmitter and modifications sent to the receiver.

Certain types of code can be explicitly modified, rather than regenerated. For example, the 4/8 prefix code can be modified by swapping code length allocations for pairs of codewords. This has the advantage over the method in (i) that the complexity is low, and the advantage over (ii) that only modifications rather than the whole code are transmitted.

Coding schemes of types (i) and (iii) are compared in Table 3.j. Three sets of data, type 1 - upper case text, type 2 - lower case text, and type 3 - numeric data, were combined to form four composite samples. The pattern of occurrence of the data types within the samples, shown below, gives an increasing frequency of change. The test was intended to show the performance of the coding schemes on mixed sources, and the rate of degradation with increasing source instability.

Sample A = (1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,3)

Sample B = (1,1,1,2,2,2,3,3,3,1,1,1,2,2,2,3,3,3)

Sample C = (1,1,2,2,3,3,1,1,2,2,3,3,1,1,2,2,3,3)

Sample D = (1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3)

The samples had an identical entropy of 5.383 bits per symbol, and a joint entropy of 7.679 bits per symbol pair. This gives a lower bound on the performance attainable using a static coding scheme designed for the source.

The first result, for an adaptive Huffman code, shows an average length for samples A, B and C that is below the sample entropy. This may be compared with the lower bound obtained by calculating the weighted average of the entropies of the individual data samples 1, 2 and 3, of 4.039 bits per symbol. The average length increases markedly with increasing frequency of source change, the average length for sample D is 20 percent greater than that for sample A.

The second type (i) adaptive code is based on the 4/8 bit fixed structure code discussed in Section 3.2.2. This has several advantages over the adaptive Huffman code, principally the ease of code generation, and the constrained maximum length. The robust nature of the code is shown by the small (seven percent) degradation in performance from sample A to sample D. The average length is however significantly larger than that attained by the Huffman code, even in the worst case. If performance is a primary consideration, the adaptive Huffman code would seem preferable.

The third result in Table 3.j shows the average length obtained for a code of type (iii). A 4/8 bit fixed structure code is maintained at the transmitter. A block of symbols are encoded (128

in this example). If the encoded block length would be reduced by modifying the code and incorporating information about the change in the block, the code modifications are transmitted before the block, and the block encoded using the modified code. This allows 'look ahead' to be used to incorporate information from the block into the code before the block is encoded.

The performance of this code is better than the type (i) adaptive 4/8 code. The most probable reason for this is the look-ahead feature which will give fast response to changes in the source characteristics. With increasing frequency of change however, the performance degrades more quickly than the type (i) 4/8 code, the average length for sample D is nine percent greater than that for sample A. If the source changes very slowly, it is possible that the code would also perform poorly, as the criterion for transmitting a *swap* command is that the current block of data would be more efficiently compressed. The overhead of transmitting the *swap* command places a form of threshold on the gain that must be realized as a result of the operation.

Sample	Adaptive code:-		
	Huffman type(i)	**4/8 type(i)**	**4/8 type(iii)**
A	**4.699**	5.715	5.485
B	**5.098**	5.804	5.534
C	**5.379**	5.882	5.589
D	**5.634**	6.128	5.993

Sample entropy 5.383 bits Joint entropy 7.679 bits

Table 3.j. Average length (bits) of adaptive codes when applied to samples with identical content but increasing frequency of change of basic data type.

3.3.2 Adaptive String Encoding

String based, or variable to fixed encoding has been regarded as complex to implement in adaptive form. Some implementations have required large amounts of memory and processor time, and have only been viable on mini- or mainframe computers; in addition, the learning times are often protracted. The techniques have not been suitable for real-time or on-line compression systems.

An example is given by Cleary and Witten (1984), of an adaptive string matching system. Their method achieved high compression, a file of English text could be compressed to less than 3 bits per symbol average length, but the processing and memory requirements were high. For a file of 44,871 English characters, an average encoded symbol length of 2.75 bits was achieved, but the memory required was 500 kilobytes. Cleary and Witten state that it should be possible to achieve better performance with 1.4 megabytes of storage, and give an expected encoding time of 120 microseconds per symbol. A maximum throughput of the order of 1000 symbols per second on a VAX 11/780 minicomputer is given as a realistic target.

The Ziv-Lempel compression algorithm was briefly outlined in Section 2.5. The algorithm is basically a string learning system. It has a dictionary of known substrings of the input sequence; the longest known substring matching the input sequence is found and

its index output as the codeword for the substring. The next symbol in the input sequence is appended to the matched substring and the resulting extended substring added to the dictionary. The next symbol is output in uncompressed form. The receiver has an equivalent dictionary, finds the substring corresponding to the received index, and performs an equivalent dictionary update to the transmitter.

For example, consider a source of symbols { a,b,c }, which emits an output sequence - a,b,c,a,a,b,c,a,b,c,b

Initial dictionary - 1. a
2. b
3. c

Appended - 4. ab
entries. 5. ca
6. abc
7. abcb

Operations -

match "a" with dictionary - 1..... output "1"
next symbol is "b" output "2"
add "ab" to dictionary
match "c" with dictionary - 3..... output "3"
next symbol is "a" output "1"
add "ca" to dictionary
match "ab" with dictionary - 4.... output "4"
next symbol is "c" output "3"
add "abc" to dictionary
match "abc" with dictionary - 6.. output "6"
next symbol is "b" output "2"
add "abcb" to dictionary

resulting output sequence - {1,2,3,1,4,3,6,2}

Obviously for a long input sequence, the substrings learnt by the algorithm can be quite large, and are limited in size mainly by the rate of learning (data dependent), and the memory available.

The preceding example showed how a sequence of input symbols may be parsed into substrings. A number of practical points need to be considered when implementing the algorithm.

(i) Coding of index values.

The output codewords must be expressed to some finite precision. The range of values will correspond to the maximum dictionary size, typically 12 bit codewords giving a dictionary size of 4096 entries.

(ii) Data structure and memory requirement.

The data structure needs careful design, as the encoding/ decoding speed and memory requirement depend on it. Early implementations used a simple array or tree structure, with each element providing storage for the maximum size of string. Quoted memory requirements have been in the range 200 kilobytes to 1 Megabyte.

(iii) Substring recognition

An important part of the encoding process is the parsing of the input sequence with respect to the substrings contained in the dictionary. A linear search would obviously be very inefficient and some faster technique is necessary.

(iv) Dictionary purging.

When the dictionary has been filled, some strategy is necessary to remove infrequently used strings, in order that the system continues to adapt to the input data. If the data is consistent however, it is not necessary to purge the dictionary.

(v) Learning phase.

When the dictionary contains few entries the efficiency of the algorithm is low, as each codeword represents only a short string.

(vi) Uncoded symbols.

The original Ziv-Lempel algorithm specifies that the output stream consists of codeword/next-character pairs. This reduces efficiency as part of the output stream consists of uncompressed data.

(vii) Early string termination.

If the stream of characters is halted, the encoder will wait for further characters without transmitting any further codewords. A control character may be added to the basic source alphabet, which causes termination of the string matching process. The transmitter sends the encoded string, followed by the control character; the receiver decodes the string but does not attempt to add it to its dictionary. This would typically be used at the end of a message, or when some given time has elapsed since the

receipt of the last character from the source.

Miller and Wegman (1982) and Welch (1984) have suggested improvements to the Ziv-Lempel algorithm which render it both memory efficient and fast, solving problems (i) to (iii) above.

Miller and Wegman describe a data structure that provides very efficient use of memory. The dictionary is held in the form of a tree, with each node containing a single character and a pointer to the parent node which represents the prefix string. A hash table (Knuth 1973) is used to determine, given a matched substring and the next input character, whether the extended substring is in the dictionary.

The implementation difficulties of this method are discussed by Welch (op cit). The hash table requires a significant amount of memory (Welch suggests 8 kilobytes for a 4096 entry dictionary), in addition to that needed for storage of the basic tree structure used to encode the dictionary. Use and maintenance of the hash table, if implemented in software, is slow, and Welch suggests that a hardware implementation or the use of associative memory would be much faster.

An improved data structure with a modest memory requirement is shown below. This follows Miller and Wegman's tree structure, but rather than linking each node only to its predecessor, which necessitated the use of a hash table, a parent node is linked to a list of dependants. Thus a node contains a character, a *down* pointer to the list of nodes representing the dependent strings, and

a *right* pointer to an alphabetically ordered list of alternative strings having the parent string as a prefix. The nodes are held in an array, and the array index is used to represent the dictionary index of the string formed by the sequence of characters on the path from the root of the tree to the last matched node. A node is defined therefore as:-

node = (character, *down*-pointer, *right*-pointer)

Figure 3.k illustrates the modified data structure, and shows the strings "q", "qu", "qua", and "qui" encoded using four nodes. To match the input sequence "quiet", the ordinal value of the first character "**q**" is used to find the initial node index (113 assuming ASCII encoding), and the *down*-pointer used to find the index of the next dictionary entry, the second input character, "**u**" is read and immediately matched with the character in the current dictionary entry. The *down*-pointer is again used to determine the index of the next dictionary entry, corresponding to the first three character string "**qua**".

The next input character is "**i**", whilst the next dictionary entry currently contains character "**a**", these do not match and a search is instigated on the *right* list of the current node. There are three possible outcomes to this search, the character may be matched in which case the next input character is read, or the search may fail because the *right*-list ends, or a character of greater ordinal value is encountered.

Index	Character	Down pointer	Right pointer
113	'q'	197	null
197	'u'	692	null
314	'i'	null	null
692	'a'	null	314

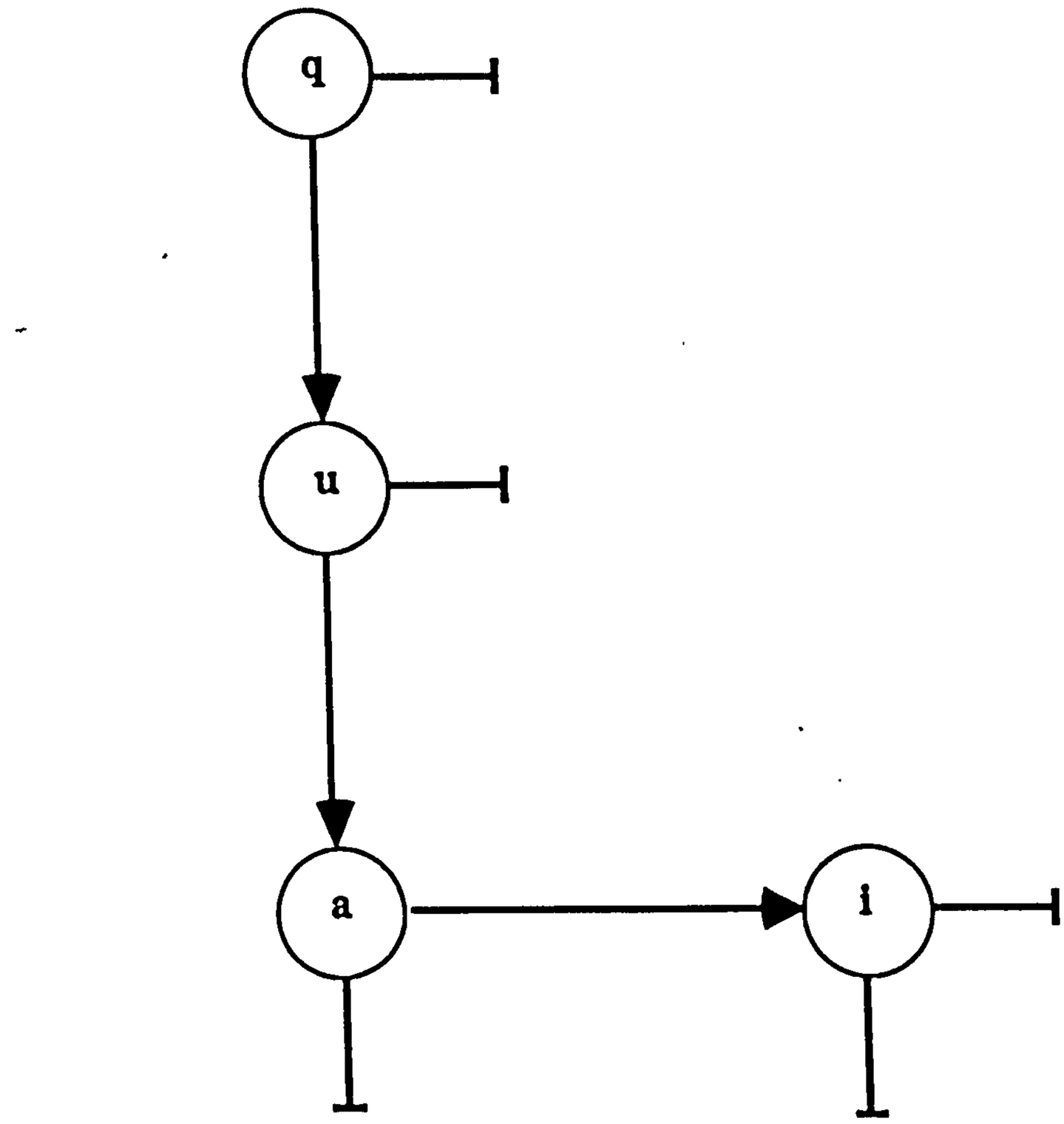

Figure 3.k The data structure for the improved Ziv Lempel compression algorithm; the dictionary entries and associated tree for the strings "q", "qu", "qua", "qui".

If the search/match procedure fails, then a string has been matched completely, and the index corresponding to the last entry is transmitted. The last (unmatched) character is used to create a new dictionary entry linked onto the last (matched) string.

The string matching process consists essentially of following a simple progression of pointers, searching is only required amongst alternative known characters for a given position within a string. The memory requirement is reduced to the order of five to twenty kilobytes.

The decoder has a similar data structure to the encoder for storage of the dictionary, but requires an additional pointer to allow backtracking up the code tree. Given some received index, the corresponding entry is immediately found, and the *parent* pointers followed until the string is completely decoded. This needs no searching at all, but produces the characters in reverse order. A last in first out (LIFO) stack is used to correct the character order, characters are pushed onto the stack during decoding and recovered afterwards.

A further development suggested by Miller and Wegman, is termed character extension. This addresses (vi) above, which stated that the output of the encoder consists partially of uncoded characters. Miller and Wegman suggest that when a string has been matched, the index is transmitted as before, but the next character, instead of being sent, is used to start the next string matching operation. The decoder adds to its dictionary, a new entry consisting

of a received string with the first character of the next received string appended. The encoder has to delay its dictionary updates by one step, to allow the decoder to maintain synchronisation.

Welch discussed a method by which initial performance may be improved, solving problem (v) above. Initially, the dictionary is almost empty and a small number of bits suffice to encode the index values. As the dictionary grows, the codeword size is increased up to some maximum. This improves performance during the first ten to twenty thousand symbols encoded, but at the cost of increased complexity. Figure 3.m. shows the improvement obtained from this technique.

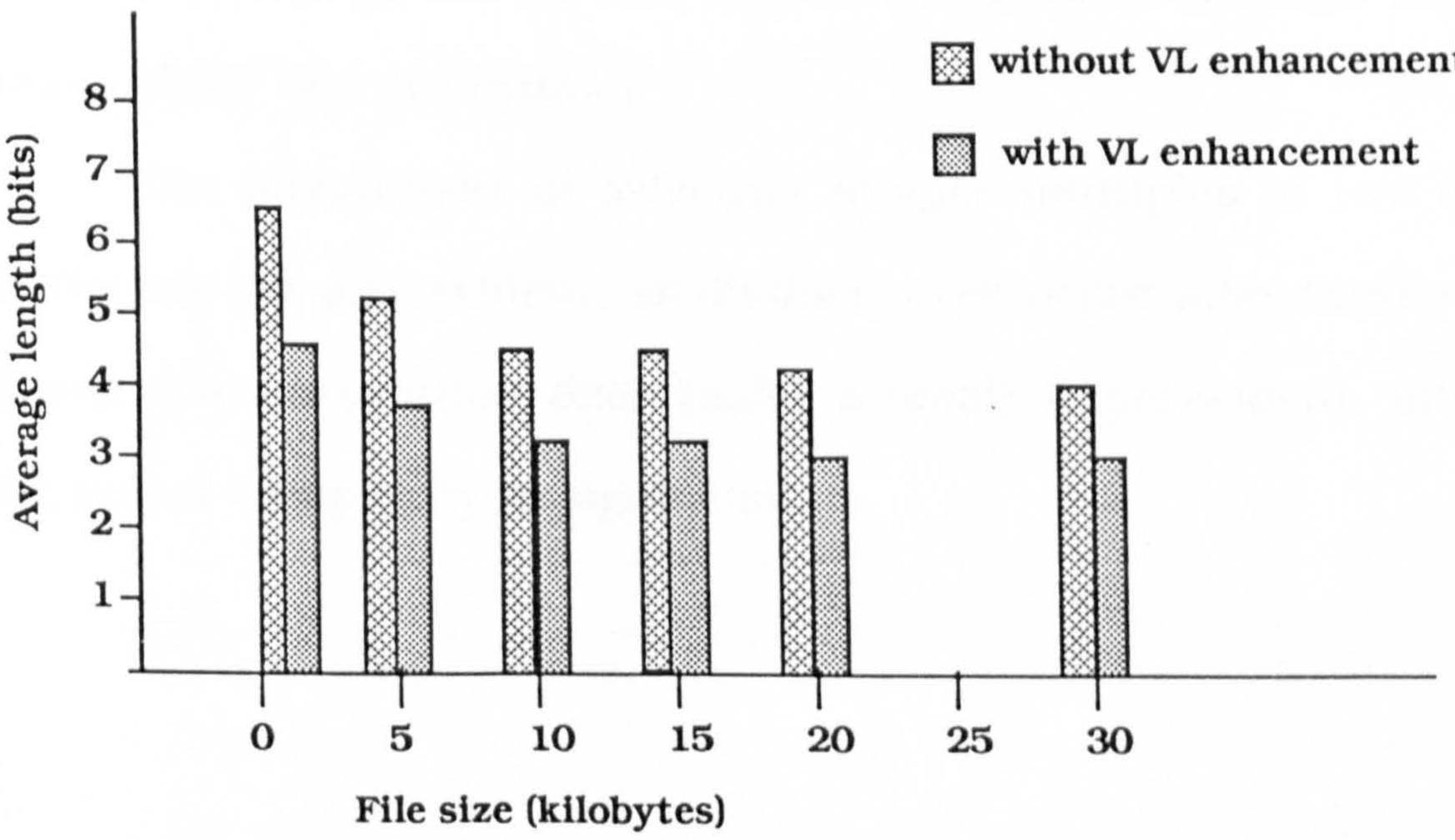

Figure 3.m Comparison of Average Length attainable by the Ziv-Lempel Algorithm, W ith and Without the Variable Length Enhancement., for file Text5.

A further modification which may have some benefits in text compression, is space synchronisation. If the string learning process is terminated when a space/character pair is found, the encoder will tend to learn words rather than arbitrary fragments of text. The drawback to this technique is that common fragments such as *and* *the* would be split into their component words.

The choice of dictionary size has a major influence on performance, as demonstrated in Figure 3.n., for an encoder with and without space synchronization. The improvement in average length with increasing dictionary capacity is quite dramatic, a dictionary with as few as 512 entries achieves an average length of nearly three bits per symbol.

The advantages of selecting integer multiples of two for the dictionary are also evident, as distinct steps occur after these values. Space synchronisation does make a small improvement, although the added complexity is significant.

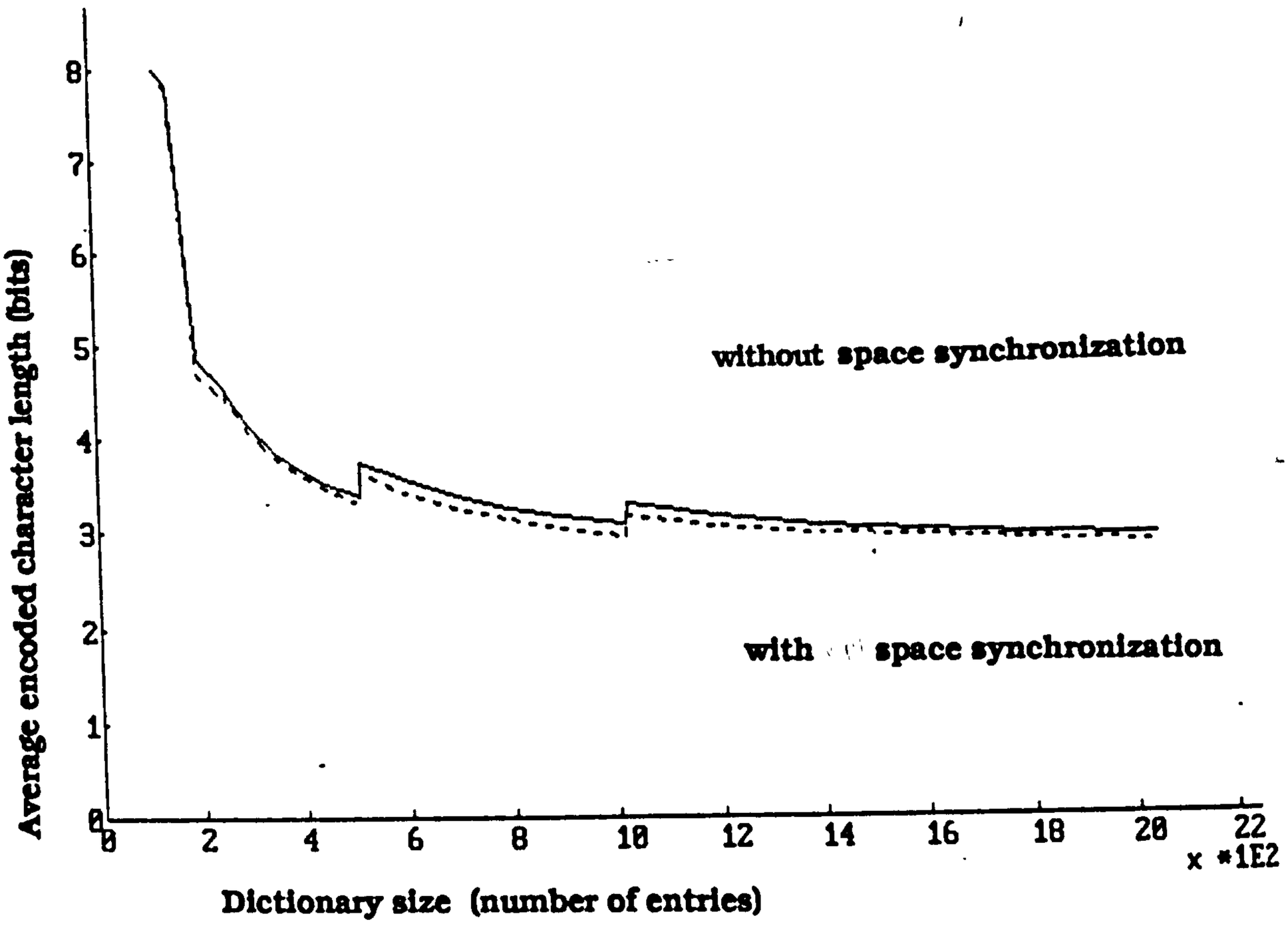

Figure 3.n The Relationship between dictionary size (number of entries) and average encoded character size, for the Ziv Lempel algorithm.

One area which may become complex is the dictionary reduction or *purging* that must be performed when there is no capacity for new entries. Welch (op cit), and Ziv and Lempel (op cit), do not discuss this problem, however Miller and Wegman suggest that a frequency count is associated with each dictionary entry, which is incremented each time the associated substring is used. When the dictionary is full, the least frequently used entries are deleted. This method is however expensive in memory, and the process of finding the least used entries is complex.

For example, the node structure shown below could be

used for a frequency count based system. Each node contains in addition to the basic components described above, a frequency count, and two pointers, which will require approximately 11 bytes of storage for the transmitter and 13 bytes for the receiver. The pointers are used to maintain the list of nodes in frequency order, in the form of a doubly linked list.

Whenever a node is accessed, the frequency count is incremented, and if necessary the node is moved up the list. The least frequently used strings will move to the bottom of the frequency list, and may be found without searching. It would be necessary to periodically scale the frequency counts to prevent overflow. A drawback of the method is the need to perform a frequency count update **for each encoded source symbol.**

node for frequency count variant:-

(character, *down*-pointer, *right*-pointer,
parent-pointer (receiver only),
frequency count, *up*-link, *down*-link)

An alternative method is based on the premise that the most frequently used strings will grow quickly. When the dictionary is full, all strings that are not prefixes of other strings are reduced in length by one character. This means that commonly used strings will keep increasing in length, whilst disused strings are progressively shortened. This has the slight drawback that the strings which only increased by one character since the last purge will be returned to their earlier state. A boolean *new* variable can be added to the node

descriptor, which is set to *true* when a node is created, and to *false* when a purge is carried out.

The method is a significant improvement on the frequency based method in terms of memory requirements, as it needs only one bit of additional information to be held with each node, and in terms of processing time as there is no need to update a count when each character is matched. The purge operation is comparable in complexity to the scaling operation needed for the frequency count method.

The performance of the two methods is compared in Table 3.p . The frequency count method achieves from 0.018 to 0.038 bits improvement over the string reduction method.

File size	**Purge method:-**	
k.bytes.	**frequency count**	**string reduction**
5	4.422	4.422
10	3.751	3.751
15	3.440	3.458
20	3.303	3.327
30	3.094	3.132

Table 3.p. Effect of dictionary maintenance strategy on average length (bits) of Ziv Lempel encoding. (File Text5)

The improved Ziv-Lempel encoder, using the string reduction method of dictionary maintenance, the variable length improvement, and the compact data structure, is an efficient data compression technique for text files. The degree of compression attainable appears generally good, although obviously dependent on the characteristics of the encoded sample.

Table 3.q. shows the compression obtained for the full range of samples. In practice, the space synchronization technique did not achieve a consistent improvement in performance, and the algorithm used to obtain the results shown did not incorporate this modification. The results clearly indicate that the algorithm achieves a consistently better performance than the earlier variable length codes, whilst the memory requirements are lower and the encoding process simpler than that needed for joint or conditional encoder implementations.

The worst performance is shown for the FORTRAN2 sample; however this is undoubtedly due to the size of the sample, less than 2 kilobytes in length.

Sample name.	Average length for a dictionary size (number of entries) of 512	1024	2048	4096
Text1	4.48	4.06	3.79	3.60
Text2	3.49	3.02	2.75	**2.58**
Text3	4.54	4.11	3.86	3.80
Text4	4.26	3.82	3.56	3.42
Text5	3.75	3.36	3.14	3.01
FORTRAN1	4.32	3.99	3.90	3.92
FORTRAN2	**4.69**	**4.57**	**4.57**	**4.54**
Pascal1	4.45	4.17	4.07	4.08
Pascal2	3.38	3.08	2.95	2.96
Prolog	3.73	3.47	3.45	3.45
Number	3.62	3.45	3.44	3.44
Image	4.49	4.13	3.96	3.86

Table 3.q. Average length (bits) achieved by the improved Ziv Lempel encoder for a range of data samples.

3.3.3 Encoding sources with unknown character size.

In designing source encoders, it is usually assumed that the input to the encoder consists of discrete symbols from the source alphabet, and hence that the symbol size is known. Many modern communication systems employ synchronous transmission, in which data is treated as a continuous bit stream rather than individual characters.

When designing a source encoder for a synchronous source which outputs non-binary symbols in binary form, several approaches may be taken. If the symbol size is constant but unknown, the character length that results in optimum source code performance can be found by search. For example, a sample of data can be taken, the assumed character size stepped from one to some maximum number of bits, and the entropy estimated for each case; the ratio of entropy to character size should be minimal for the best compression ratio to be achieved.

An alternative approach is possible if a string encoding is used. A symbol size is selected arbitrarily, the assumption being that frequent strings of source symbols will not lose their identity when parsed into bit strings of different lengths. There are obvious points to consider when selecting the segment size (where a segment is an assumed symbol).

For a segment of length C, as there is no synchronisation between the position in the source bit stream of segments and of symbols, any regular string of symbols may coincide with any of the C bit positions in the segment. This implies that, for any frequent

string of symbols, at least C variations of the string are possible, and hence short segments are likely to result in more economical use of the dictionary space.

As the encoded binary string length is proportional to the segment size, but the codeword length is proportional to the logarithm of the number of known strings, short segments will make less efficient use of the dictionary space than long segments. This is particularly relevant to the Ziv-Lempel algorithm, in which a large proportion of the memory space is devoted to pointers rather than symbol storage.

The two observations above imply that an optimum segment length may be found. Figure 3.r shows the normalized average length obtained for a sample of seven bit ASCII encoded text, for segment sizes from one to eight bits. Several features may be seen, a slight valley at 3-4 bits, and a minimum at 7 bits corresponding to the true character size. In addition, a low value occurs at a segment length of eight bits.

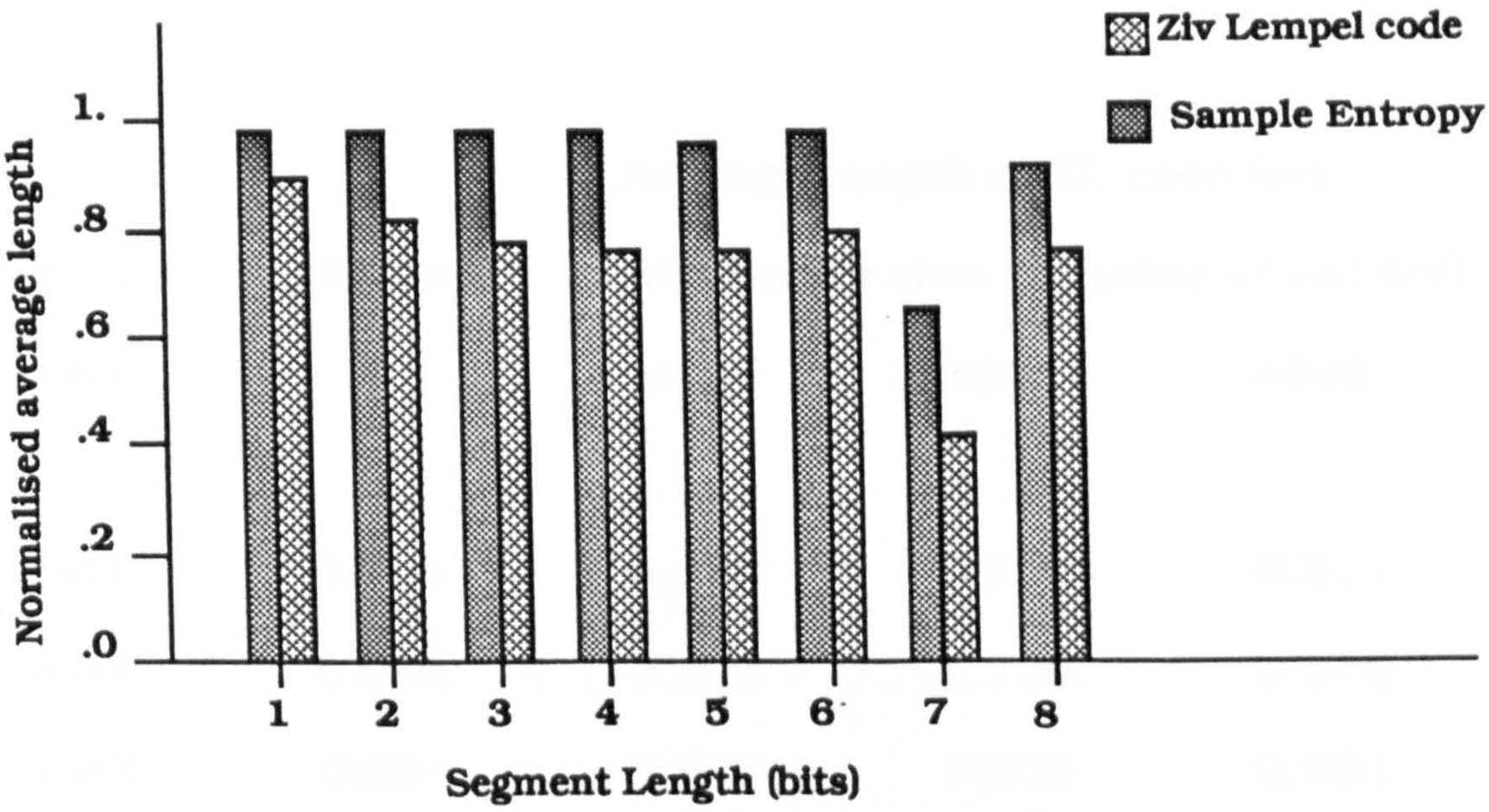

Figure 3.r The efficiency obtained on synchronous data of unknown character size, for assumed character (segment) sizes from 1 to 8 bits. Average encoded character size normalised wrt segment length.

Table 3.s shows the normalized average length obtained using the Ziv-Lempel algorithm for six samples, for a segment size of four bits. The highest compression obtained was 32 percent, however for a dictionary size of 2048, the average compression was 13.2 percent. As one would expect, the use of a large dictionary provides more reliable compression. On the Pascal sample slight expansion was obtained with a dictionary size of 1024 entries.

Sample name.	Entropy	Average length of ZL code for: Dictionary size (number of entries) 1024	2048	4096
Text1	0.994	0.970	0.898	0.841
Text2	0.986	0.809	0.736	0.676
Text3	0.994	0.973	0.915	0.881
Text4	0.996	0.923	0.848	0.794
Pascal1	0.991	**1.004**	0.965	0.949
Image1	0.993	0.911	0.841	0.791

Table 3.s Normalized average length of ZL code on synchronous data samples, for an assumed character size of four bits.

It appears feasible to compress synchronous data with unknown character size, and compression of 10 to 20 percent may be obtained. There is room however for further development in several areas.

(i) The effects of synchronous protocols on the data stream, for example the SDLC bit stuffing mechanism discussed in Section 6.2.1

(ii) Packet networks carry mixed format data, and it is not possible to predict from the evidence given, the likely performance.

(iii) The choice of a four bit segment size is based on only one sample, and should be supported by further analysis.

(iv) The use of variable rate encoding requires some means of regulating the flow of source data. This may be accomodated using a buffer, although there is a risk of buffer overflow (Humblet 1981), or flow control (Section 6.2.2). These techniques are more suited to asynchronous than synchronous traffic.

Synchronous data streams are almost always maintained using some form of link protocol capable of error control. In many cases the additional efficiency gained through compression of the synchronous stream may outweigh the loss in efficiency caused by errors due to buffer overflow.

3.4 Discussion

This chapter has considered three approaches to the encoding of partially known sources. If the source is fairly stable, but the symbol probabilities are inaccurately known, a robust code may be used, for which a number of design methods have been discussed. In practice, a communications channel may carry a wide variety of data types, and a static coding scheme would not be expected to perform well. Under these conditions, an adaptive source code is more likely to achieve good performance.

The choice of compression scheme is heavily reliant on the memory and processor constraints imposed by the communications subsystem. Although memory requirements can be accurately estimated, the processing requirements (time complexity) of most of the schemes discussed are very data dependent.

Robust codes were discussed in Section 3.2 above. The tolerance of a code to changes in symbol frequencies may be improved by limiting the maximum codeword length. A number of methods for achieving this were outlined, but it was generally found that the average length of the code was increased as a result of the constraint. A simple fixed structure code, having only four and eight bit codewords was suggested. This differs from a Huffman code, as the number of codewords of each length is predetermined. Two other approaches were also described, one of which achieved a consistently better performance than the fixed structure code,

however the complexity of these two methods is considerably greater and hence the simpler method is more appropriate to the application.

Adaptive codes are ideally suited to communications systems in which the source is non-stationary. In Section 3.3.1 several types of code were discussed, adaptive Huffman codes such as those described by Faller (1974) and Gallager (1978), and two types of adaptive fixed structure code. These were tested on a simulated non-stationary source, and the effects of rate of change of source type, observed. The robust fixed structure codes were more stable under these conditions, than the Huffman code, however the average length of the Huffman code was still significantly lower.

In Section 3.3.2, adaptive string encoding was discussed. The Ziv-Lempel (1977) algorithm was shown to be capable of achieving good compression, but subject to a number of practical problems. An improved algorithm was then developed, depending partially on the work of Miller and Wegman (1982) and Welch (1984), that is simple to implement, and efficient in memory and processing requirements. The novel aspect of the algorithm is the use of string reduction, rather than a string frequency based approach to dictionary maintenance.

Table 3.t. compares six of the adaptive source coding schemes discussed above, and shows the average length for the mixed data samples described in Section 3.3.1. and the estimated encoder and

decoder memory requirements.

Adaptive Huffman coding provides reasonable performance for a slowly changing source. The degradation with increasing source variability is due to the large maximum codeword length and to the time delay in updating the code.

The adaptive fixed structure codes provide moderate compression, but are extremely simple to implement. The second method, based on the transmission of changes to the code, is better than the first in terms of both complexity and performance.

An estimate of the performance and memory requirement of an adaptive conditional encoding scheme is included for comparison. The estimated average length is based on the conditional sample entropy.

The average length of the improved Ziv-Lempel algorithm, for dictionary sizes of 1024 and 2048 entries, is given. In both cases, the average length is almost half that obtained from adaptive Huffman coding, whilst the memory requirement is quite acceptable for a small microprocessor system.

If the source is supplying synchronous data, with unknown or mixed character sizes, as may be found within a public or private data network, variable length coding is unlikely to provide acceptable performance. Section 3.3.3 showed that compression of this type of data was possible, using the Ziv-Lempel algorithm.

The effects of transmission errors on the operation of the source decoder, and the maintenance of synchronization between

encoder and decoder, were introduced in Sections 2.7 and 3.3.1 Adaptive codes are particularly vulnerable to transmission errors, and may need resetting to the default code if an invalid codeword is received. Variable length codes may be constructed such that any binary sequence can be parsed into valid codewords. This means that the decoder could not detect errors, and hence some other mechanism must be added in order that synchronization is checked.

The modified Ziv-Lempel encoder does however have some limited error detection capability. The dictionary is maintained partially full, as immediately it is full a purge operation is carried out. Thus, if a received codeword is found to correspond to an empty dictionary entry it can be deduced that an error has occurred. This check would only indicate that some error had occurred, not whether the received codeword was incorrect or the dictionaries had lost synchronization.

Any of the methods described above may be considered suitable for on-line compression within a communications system. The 4/8 fixed structure code is attractive for small systems, particularly if used in conjunction with run length encoding (Section 2.6). For moderate or large systems, the high degree of compression attainable by the Ziv-Lempel encoder is well worth the additional memory and processing time. There are however circumstances in which the adaptive Huffman code may perform better than the Ziv-Lempel method, for example on a data set in which symbols have a well defined probability distribution, but have a high conditional entropy.

Sample	Average length for coding scheme					
	Huffman	4/8 Fixed struct.	4/8	Condit.1 Huffman (est)	Improved Ziv Lempel 1024	Ziv Lempel 2048
A	4.70	5.71	5.48	2.7	2.47	2.46
B	5.10	5.80	5.53	2.9	2.78	2.58
C	5.38	5.88	5.59	3.1	2.77	2.60
D	5.63	6.13	5.99	3.2	2.83	2.63
Memory requirement (bytes)						
Encoder	1654	136	136	200k	5k	10k
Decoder	1654	136	8	200k	7k	14k
Adaption type	(i)	(i)	(iii)	(i)	(i)	(i)

Table 3.t. Comparison of average length and memory requirements of adaptive coding methods.

4 TRANSMISSION ERRORS AND ERROR CONTROL.

4.1 Introduction.

The switched telephone network (STN) is subject to a wide variety of interference, of both man made and natural origin. As a result, when digital information is transmitted over the network, errors may occur in the form of corrupted, deleted or inserted message symbols. It is desirable to reduce the rate at which errors occur, however this involves some increase in message transmission cost due to the higher quality transmission medium that must be used or the overhead imposed by the error correction technique. The ideal error control system would achieve either a stated reduction in error rate, or some specified residual uncorrected error rate, with minimum increase in message transmission cost or time.

Shannon (1948) introduced the concept of channel capacity as the maximum rate at which symbols may be transmitted over a channel with arbitrarily small probability of error. A mathematical model which represents the channel error process is used to calculate the channel capacity, and to provide a basis for the design of an error control system. This implies that a model must be matched to the known characteristics of the channel, either to the sources of interference or to the measured distribution of the observed errors. Although a number of telephone channel error models have been proposed, by for example Berger and Mandelbrot (1963) and Elliott (1965), practical experience over dialled

connections would support Burton and Sullivan (1972), who point out that, in their opinion, *no one knows how to determine the capacity of a real telephone channel, warts and all.*

Burton and Sullivan were comparing various error control techniques for use over the the telephone channel. They conclude that, although Shannon showed that an optimum forward error correction code may be devised, block retransmission schemes (ARQ) provide a more practical solution for this application, and argue further that emphasis should be placed on the improvement of ARQ techniques rather than the development of forward error correction codes. Recently the falling costs and increasing power of integrated circuits have allowed powerful error control techniques to be implemented, both forward error correction and improved block retransmission schemes.

Effective error control is highly desirable to users of data communications systems. If data compression is used, error control becomes essential; the preceding chapter discussed the effects of errors on adaptive data compression, concluding that errors would cause loss of synchronization between source encoder and decoder. This would result in a (possibly long) series of symbol errors, and necessitate re-initializing the source encoder and decoder, with a consequent loss in efficiency.

ARQ error control techniques are ideal for this application, in terms both of efficiency and uncorrected error rate. Retransmission of data only occurs when errors have been detected, therefore under error free conditions the efficiency of ARQ is high. Further, ARQ

schemes depend on the error detecting rather than the error correcting properties of the code used, which provides a low residual error rate. The emphasis will therefore be placed on ARQ and hybrid ARQ schemes.

This, and the following chapter, aim to determine the most appropriate error control technique for use on the telephone channel, and to contribute to the understanding of ARQ error control techniques by making an objective comparison.

This chapter provides essential background material for the following more detailed analysis of error control systems (in Chapter 5). Initially the sources and characteristics of errors on the telephone channel are discussed; the objective is to identify properties relevant to the later discussion, and to establish a suitable channel error model for performance comparison. This is followed by a brief review of the relevant areas of classical coding theory; including the use of linear block codes for error detection and correction.

4.2 Characteristics of the telephone channel.

The telephone channel, in common with most other transmission paths, introduces some distortion and noise into the signal carried. Modern modems are designed to transmit data at very high rates (up to 19200 bits per second) over a channel with a nominal bandwidth of 3 kilohertz, which is made possible through the use of complex signal processing techniques (Lucky 1968, Watanabe 1978, Clark 1977, Brownlie 1984). The modulated signal transmitted over the telephone channel is subjected to distortion and additive noise, and the receiving modem attempts to reconstruct the original digital signal. With high levels of interference, some receiver decisions will inevitably result in differences between the reconstructed and original signal elements (source symbols), termed transmission errors. In this section, the relationship between signal path disturbance and transmission errors will be discussed, allowing subsequent sections to view the channel as purely digital.

4.2.1 Channel impairments.

A number of studies of channel impairments have been undertaken in recent years, on major telephone networks. The surveys reported by Williams (1966) of the British telephone network, Duffy and Thatcher (1971), Batorsky et al (1984) and Carey et al (1984) of the American Bell telephone network, form the basis

for the following discussion.

The telephone channel suffers from a variety of impairments, due to transmission line effects and to the switching and multiplexing schemes employed. Whilst the types of impairment are well known, and the cause of the impairment usually known, only certain effects are predictable enough to allow modem design to compensate for them.

(i) Frequency response.

The telephone channel exhibits a characteristic amplitude and phase response (subject to variation between grades of line and national networks). The insertion loss is fairly flat between 300 and 2600 Hertz, increasing rapidly beyond these limits. The phase response is less often given than envelope delay distortion, which is the derivative of phase with respect to frequency.

(ii) Propagation delay.

The overall propagation delay increases monotonically with distance at approximately 7 microseconds per mile; for typical terrestrial links this will lead to delays of 5 to 30 milliseconds, but satellite links introduce much greater end to end delays of the order of 350 milliseconds per hop.

(iii) Echoes.

Impedance irregularities on the channel (for example mismatched 2-4 wire conversion hybrid transformers) cause some part of the signal to be reflected. On telephone channels

with round trip delays exceeding about 20 milliseconds echo suppressors are introduced into the transmission path. These are disabled by the modem during transmission.

(iv) Frequency shift or offset.

The use of frequency division multiplexing schemes (high order multiplexing used on trunk lines) in which the carrier is generated locally and not transmitted with the signal, introduce a steady frequency offset due to the difference in frequency of the send and receive local carriers.

(v) Amplitude and phase disturbances.

Hits, rapid temporary changes of gain and phase with a duration of between 4 and 32 milliseconds are generally small in magnitude, and are often associated with impulsive noise (Carey 1984). Permanent changes of phase or amplitude, known as jumps, may also occur.

(vi) Amplitude and phase jitter.

Jitter is usually repetitive in some systematic way, and may be due to power supply ripple voltages, or a number of other sources (Bell 1971).

(vii) Interruptions.

Short interruptions in the transmission channel, of widely varying duration may occur, these are often termed *dropouts*. In addition, the call may terminate prematurely.

(viii) Crosstalk.

Unwanted coupling from other telephone links can introduce some component of voice or data signals.

(ix) Noise.

A large number of different noise sources may contribute to the overall received noise, depending on the path taken by the circuit. (Batorsky 1984).

(x) Impulsive noise.

Short, high amplitude impulses are a well known feature of the telephone channel (Enticknap 1961). They have generally been regarded as stemming from the mechanical noise associated with selectors and relays in the older type of telephone exchange although other explanations have been given (Fano 1977). Carey (1984) gave impulsive noise counts obtained from a number of different types of exchange, and showed that the level of impulsive noise was far lower on electronic exchanges than electro-mechanical ones, but was still present.

(xi) Other sources.

A number of other types of impairment are found, amongst which are intermodulation distortion due to non-linearities, and quantization noise which may result from part of the transmission path being routed through a digital PCM network.

4.2.2 The effects of impairments on demodulation.

A modern high speed modem (over 1200 bits per second full duplex) incorporates several features (Clark 1977, Watanabe 1978, Proakis 1983) which affect the error performance (Figure 4.a).

The binary data is initially fed through a scrambler, the function of which is to prevent the prolonged short periodicities in the transmitted signal which could starve the carrier recovery, timing recovery, and adaptive equalizer of the spectral components needed for correct operation. The scrambler is a simple shift register device with feedback, similar to a pseudorandom sequence generator.

The scrambled binary data is then modulated, usually using differential phase shift (DPSK) or quadrature amplitude (QAM) modulation, and transmitted. For a two wire modem, as used on dial-up rather than leased lines, some form of signal combining will take place as the transmitted and received signals share the same pair of wires. The signal is then subject to the impairments described in the previous section.

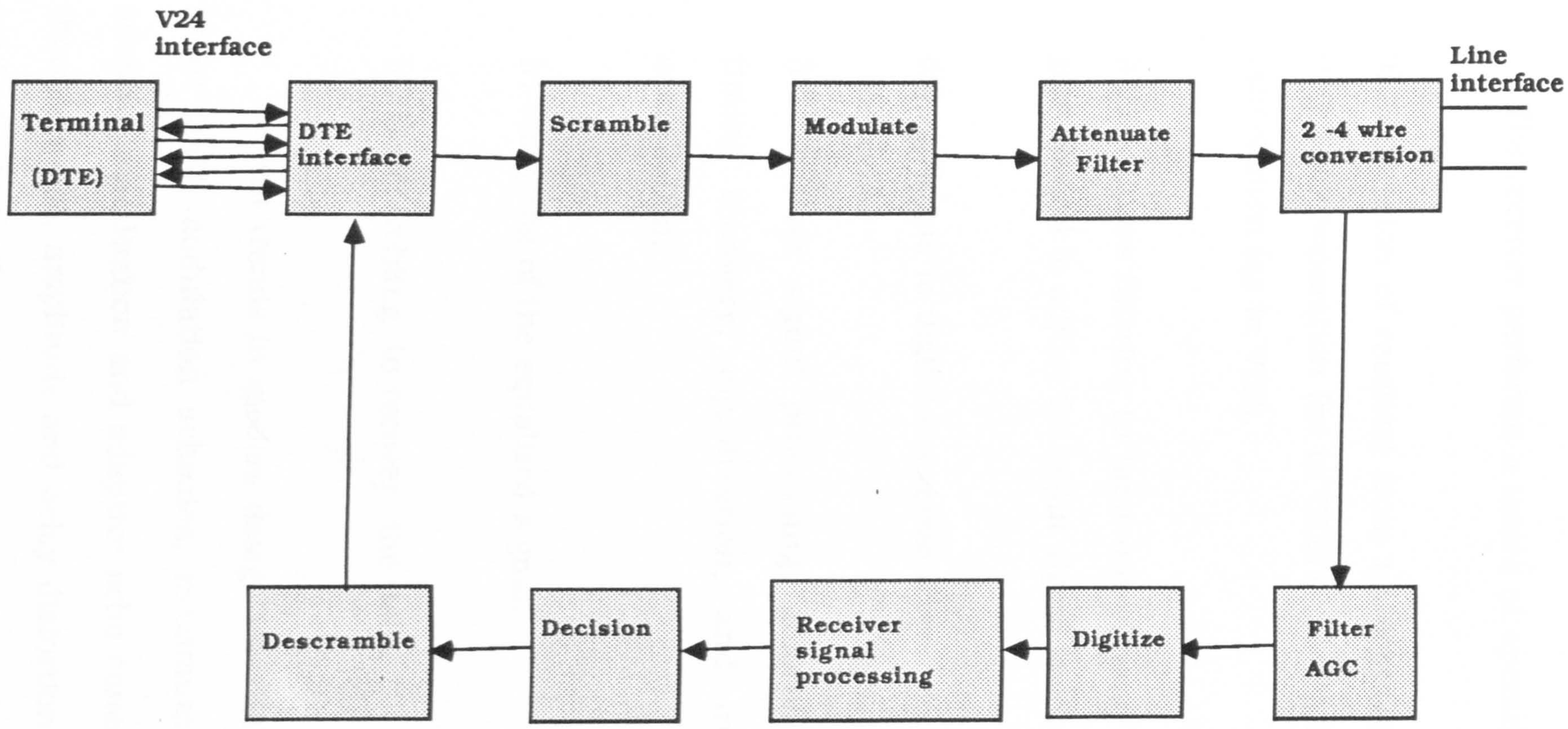

Figure 4.a Block Diagram of a High Speed Modem

The receiver performs a series of operations:-

(i) Separation of received from transmitted signal, using either frequency separation (as in CCITT recommendation V22) or echo cancellation (as in V32).

(ii) Bandpass filtering to limit out-of-band noise, and automatic gain control to adjust the input signal level.

(iii) Analogue to digital conversion.

(iv) Receiver signal processing, incorporating demodulation, timing recovery, equalization, and in some cases echo cancellation.

(v) Sampling of the equalized signal.

(vi) Descrambling, to recover the data.

Recent trends in modem design have involved the use of high order QAM modulation schemes, maximum likelihood decoding, adaptive equalization and adaptive echo cancellation. The effects of thermal noise, amplitude and delay distortion are reduced by these methods, but other impairments can still introduce errors.

Watanabe (1978) discussed the design and performance of a 4800 bit per second V27 modem using 8-PSK and adaptive equalization. For phase hits of 90 degrees, from five to nine error

bits resulted, due to error propagation in the equalizer.

The use of convolutional codes with Viterbi decoding in the modem receiver has resulted in good performance at data rates of over 9600 bits per second. This type of decoder makes use of soft decision information when mapping received symbols onto the QAM signal constellation. Under poor line conditions, the Viterbi decoder may result in considerable error extension (Lin 1983).

Studies of high speed error performance by Balcovic et al (1971) and Carey et al (1984), show that errors tend to occur in groups of size and weight corresponding to the scrambler employed. A single error from the demodulator is passed through the descrambler resulting in error extension. As some modem standards specify long scramblers (23 bits for CCITT Recommendation V29), this represents one of the more serious effects introduced by the receiver.

In summary, low speed modems would tend to introduce isolated errors, mainly due to impulsive noise, whilst high speed modems introduce burst errors generally of low weight, and with length dependent on the scrambler polynomial. Longer bursts of errors are likely to result from periods of heavy noise, incorrect equalization, or dropouts.

4.3 Transmission errors.

The errors affecting the digital message may be characterized by relating their time distribution to some stochastic model, in a similar way to that used in source modelling (sections 2.2,2.4). This must be approached with some caution however, as the error distribution is very dependent on the design of the modem used, and on the properties of the individual telephone channel. For leased line applications, the line/modem pair remain unchanged for extended periods of time. If dial-up modems are used however, the channel characteristics will change with each call, and the resultant error processes will be subject to variation.

Initially some of the results of telephone channel error measurements carried out on the American Bell, and German networks will be reviewed, followed by a discussion of a number of error models. The aim is not to find a unique model for the telephone channel for, as explained above, this is dependent on the modem, channel, and conditions of use; rather the objective is to discuss the suitability of the models to the evaluation of error control schemes and identify a number of representative models which cover a range of conditions.

4.3.1 Telephone channel error statistics.

A number of measures of channel error rate are used, which highlight different aspects of the error distribution.

(i) Bit error rate.

This is used to measure the average rate at which bit errors occur, or the probability of a bit error occurring (more correctly the bit error probability).

(ii) Block error rate.

As transmission systems often send data in fixed or variable size blocks, the probability of a block of known length containing any errors is more meaningful than the bit error rate. The block error rate is the measured proportion of errored blocks, the block error probability the corresponding probability that a block will be received in error.

(iii) Burst length and weight.

As errors often occur in distinct bursts, particularly on the telephone channel, some method of characterizing bursts is necessary. A burst is defined as a group of two or more bit errors, the distance between any two consecutive errors being less than some limit (the *guard space*). The principal measures applied to the characterization of bursts are *length*, (the distance between the first and last error in the burst), and *weight*, (the number of errors contained within the burst).

(iv) Gap length distribution.

Another measure applied to channels characterized by burst errors is the frequency distribution of the length of gap

between errors. This is a useful measure for block oriented transmission systems, as it gives an idea of the error free intervals as well as the *burstiness* of the channel.

(v) Error free or errored seconds.

A channel performance measure specified by the CCITT, Recommendation G821 (defined for high speed digital transmission channels), includes three error parameters:- the percentage of periods of one second containing no errors, (%EFS), the percentage of periods of one second with bit error rate less than 0.001. , the percentage of periods of one minute with bit error rate less than 1E-6.

Information on typical channel error conditions is available from the surveys carried out by the American Bell Telephone company at regular intervals. Results are available from 1960 (Alexander, Gryb and Nast,1960), 1962 (Townsend and Watts, 1964), 1969-70 (Fleming and Hutchinson, 1971; Balcovic et al,1971) and 1982-83 (Carey et al, 1984).

Lewis and Cox (1966) included a set of measured gap lengths in a discussion of channel models. Figure 4.b(i) shows the gap length distribution, in which a strong component of approximately 120 bits is observable. This is attributed by Lewis and Cox to dial pulses, as 120 bits corresponds to the interval between pulses of 100mS, at the transmission rate of 1200 bits per second used in the tests. This is confirmed by Figure 4.b(ii), which is a scatter diagram showing

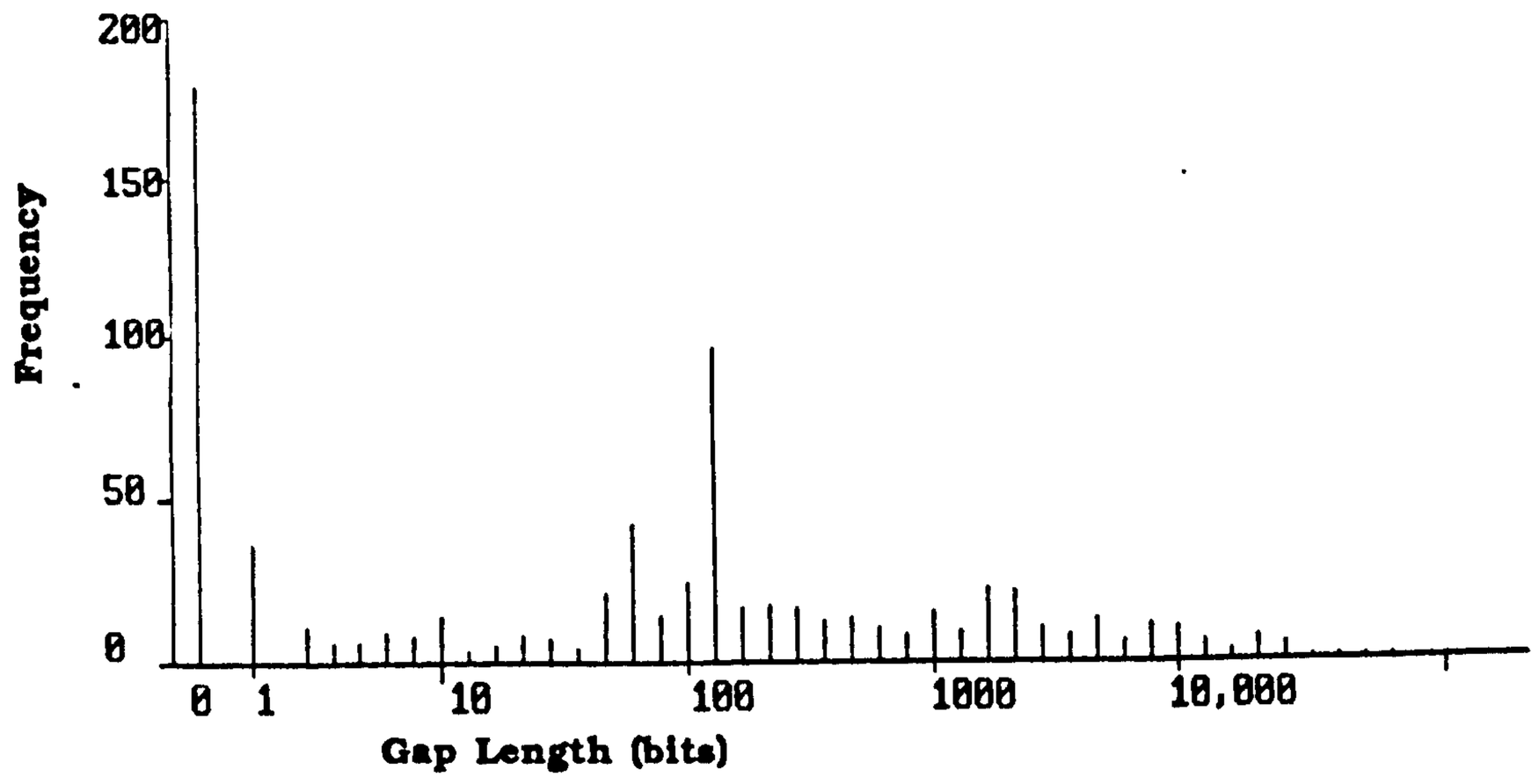

Figure 4.b(i) Histogram of the Gap Lengths of the channel error data given in Lewis and Cox (1966)

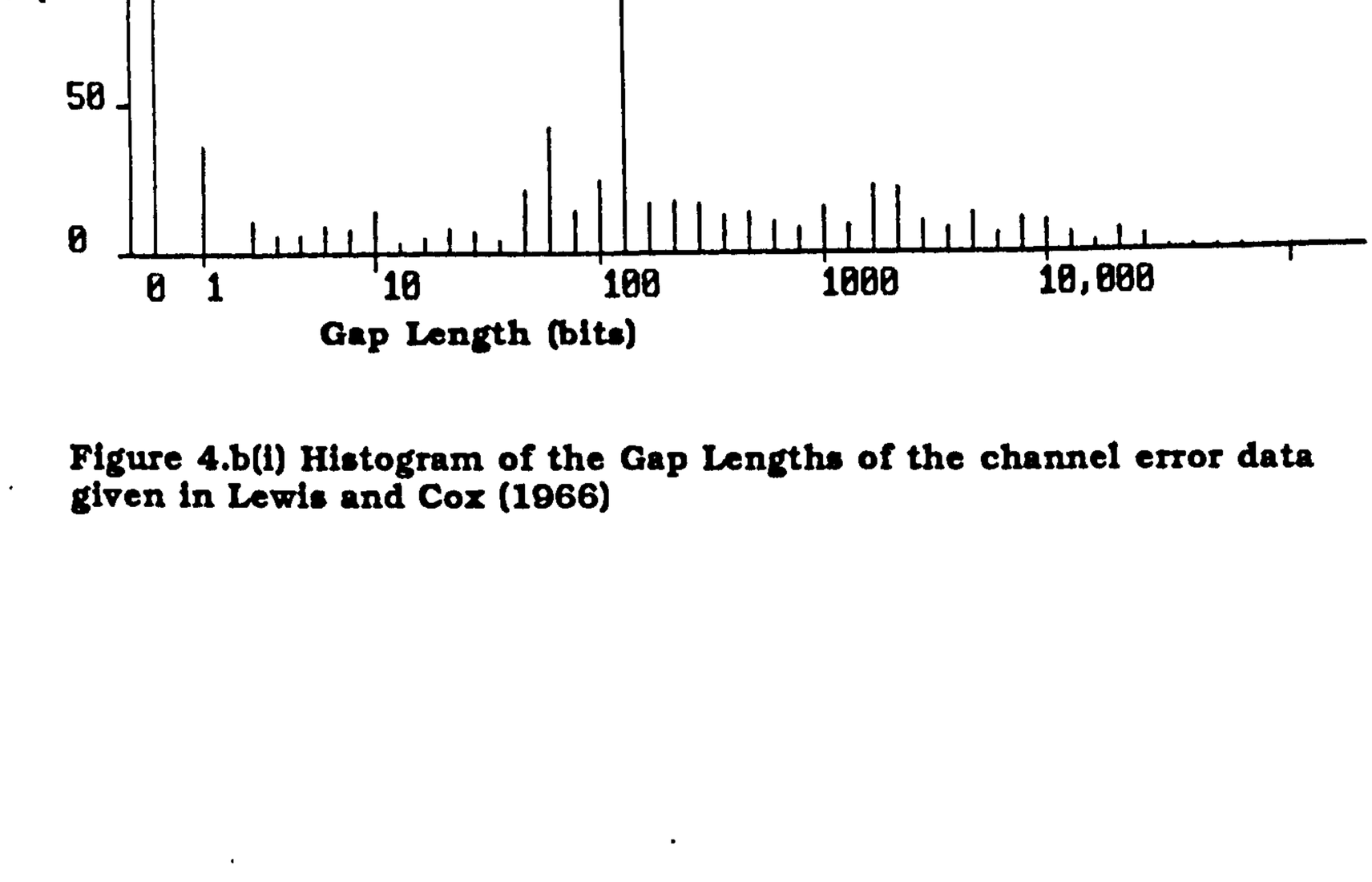

Figure 4.b(ii) Scatter diagram showing the Gap(i) / Gap(i-1) correlation for the Lewis and Cox data.

the correlation between adjacent gaps (the x axis is the i-th, the y axis the (i-1)th gap); clusters are apparent around the points (0,120),(120,0) and (120,120).

The effects on the error distribution of the modem receiver descrambling the received data, were simulated by convolving Lewis and Cox's data with a 23 bit descrambler polynomial. Figure 4.c(i) shows the gap length distribution for the descrambled data, which shows high occurrence rates for gaps of length five and fifteen corresponding to the tap weights on the descrambler. The number of errors (and hence gaps) has increased by a factor of three.

Figure 4.c(ii) shows the multigap distribution for the descrambled data. There are strong structural components within the scatter diagram corresponding to the distance between the scrambler taps.

Figures 4.d(i) and 4.d(ii) show scatter diagrams of burst length against burst weight, for the unscrambled and descrambled data, for a guard space of 100 bits. Other than illustrating the predominance of single and double errors and low weight bursts, Figure 4.d(i) shows no strong trends. Figure 4.d(ii) shows that the error extension introduced by the descrambler results in error bursts with weights that are multiples of 3 (the number of taps). Some interaction between error and scrambler polynomials does occur, as shown by the occasional burst of weight 4 or 7.

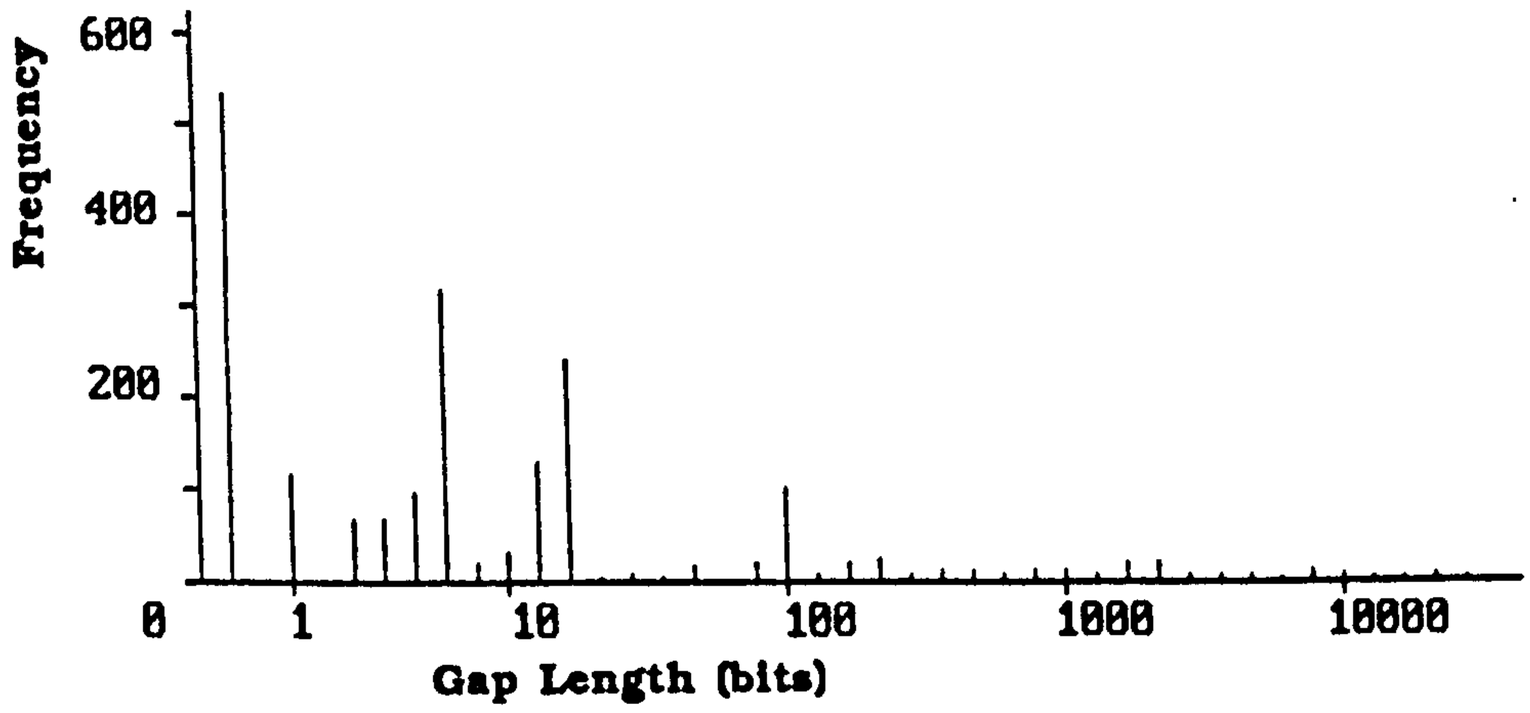

Figure 4.c(i) Histogram of the Gap Lengths of the channel error data given by Lewis and Cox, after descrambling

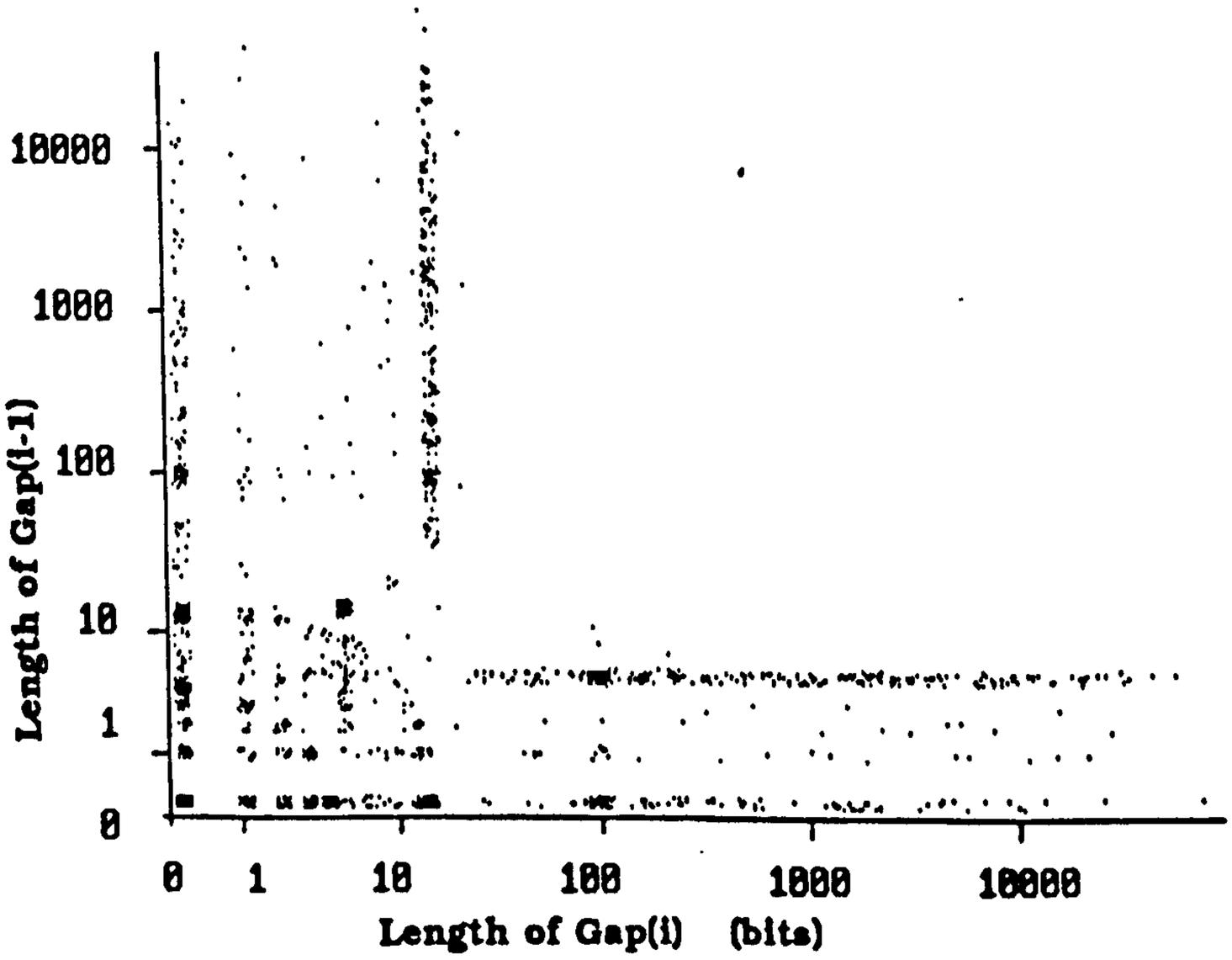

Figure 4.c(ii) Scatter diagram showing the Gap(i) / Gap(i-1) correlation for the Lewis and Cox data., after descrambling

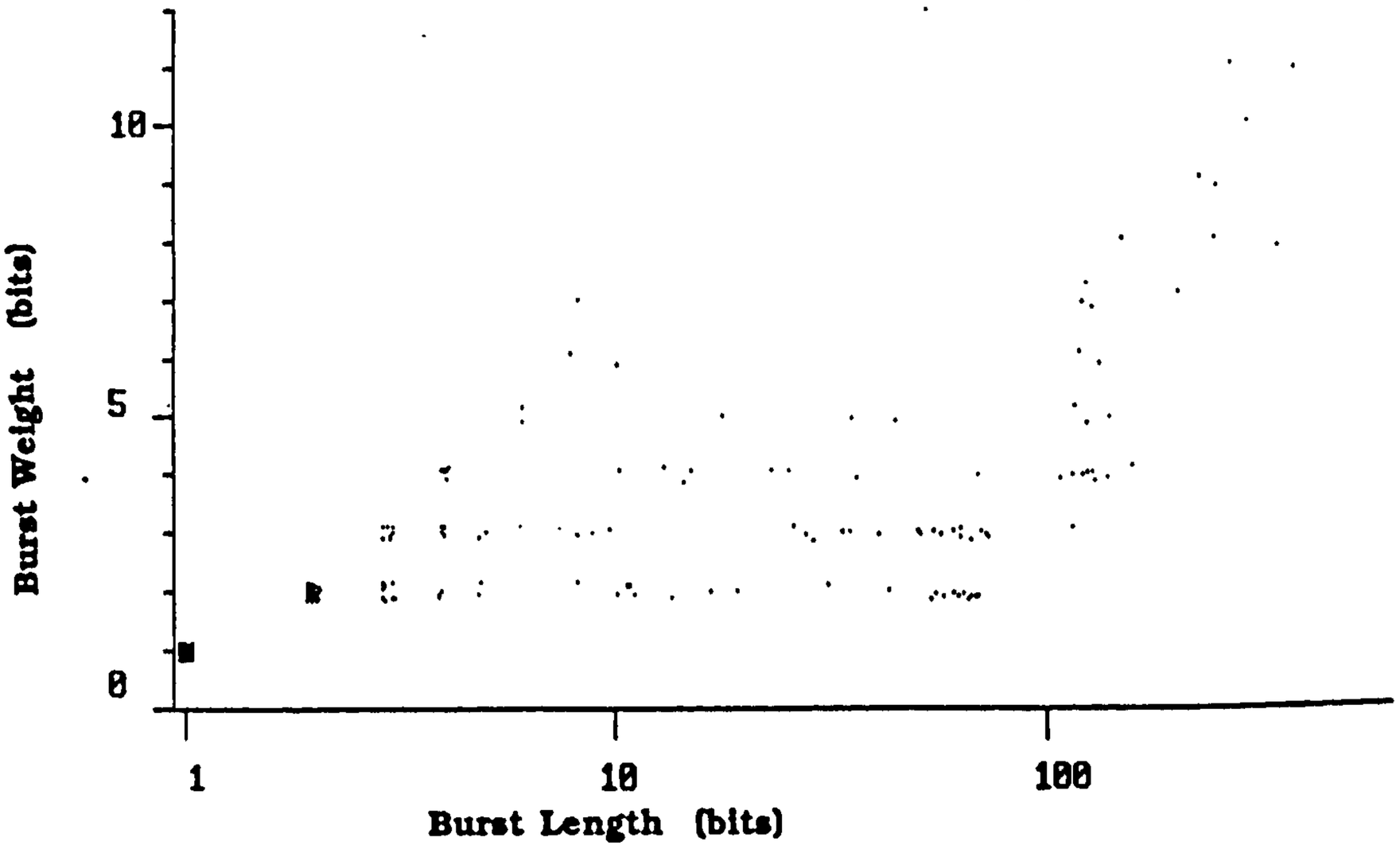

Figure 4.d(i) Scatter diagram of Burst Length against Burst Weight for the Lewis and Cox data, with a guard space of 100 bits

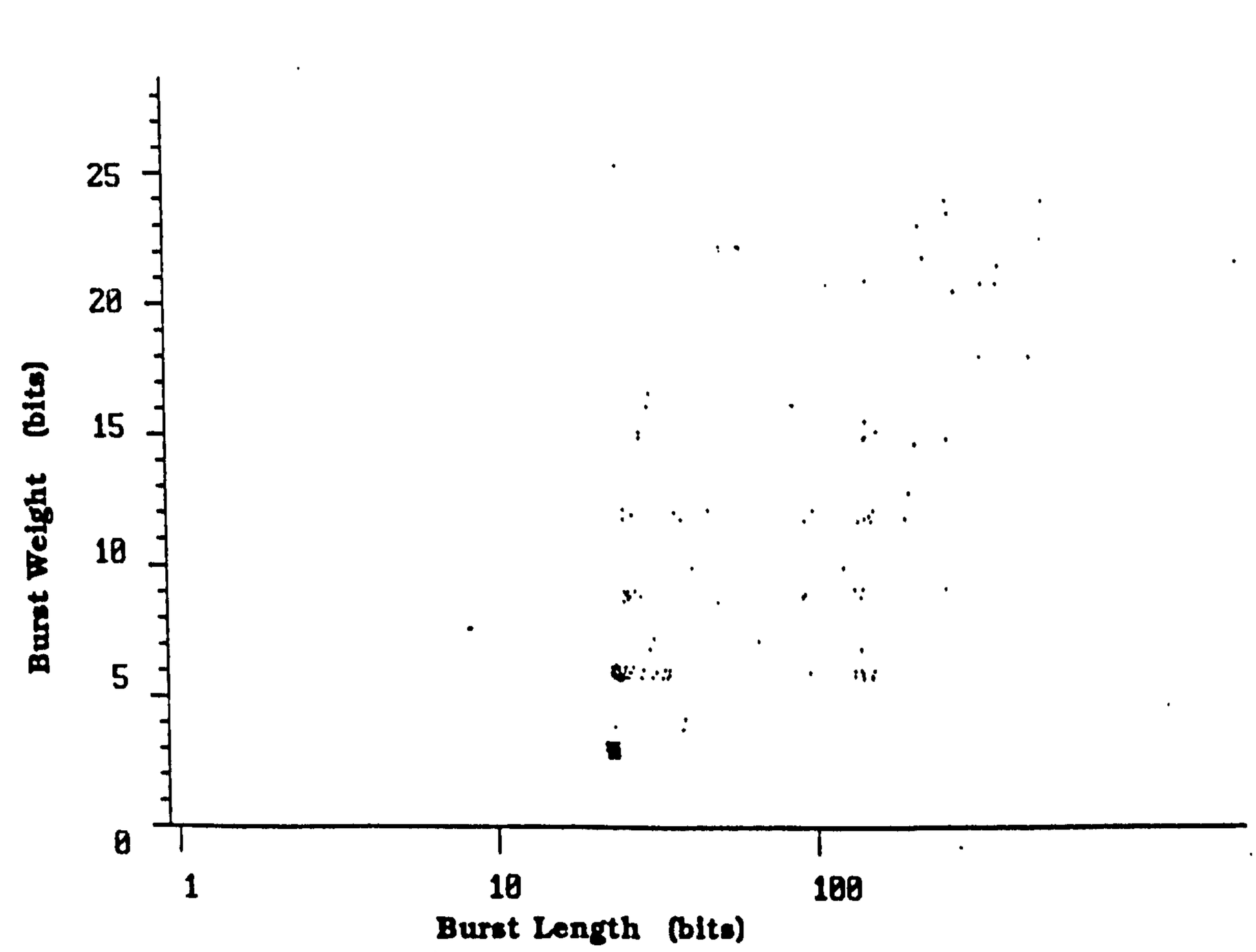

Figure 4.d(ii) Scatter diagram of Burst Length against Burst Weight for the Lewis and Cox data after descrambling, with a guard space of 100 bits

4.3.2 Channel error models.

The simplest and most widely used channel model is the binary symmetric channel (BSC). In this model, each bit interval is considered independently of any other, and the state of the bit altered according to a simple probabilistic model. The transition probability p_t gives the probability of a bit being incorrectly received as a result of channel noise.

The error distribution resulting from the binary symmetric channel model is usually termed random, by which is meant typical of a channel in which bit error events are independent, and is characteristic of a channel subject to additive white Gaussian noise (AWGN). For a BSC with transition probability p_t, the probability of m errors in a block of length n is:-

$$P(m ,n) = {}_nC_m \, p_t^m (1-p_t)^{n-m}$$

for the case in which m=0:-

$$P(0 ,n) = (1-p_t)^n$$

$$P(>0,n) = 1 - (1-p_t)^n$$

Although the binary symmetric channel does not model the telephone channel satisfactorily, it is a standard with which most error control systems are tested, and its properties are well

understood.

A model which is more suited to burst channels such as the telephone channel, was proposed by Gilbert (1960). The Gilbert model is based on a two state Markov process, which makes each error event dependent on its predecessor. One state is assumed to represent error free channel conditions, whilst the other state is a binary symmetric channel with transition probability p_t. At each bit interval, the process may change states, depending on the current state and state transition probabilities. Figure 4.e(i) provides an illustration of this model.

The model produces error patterns with a geometric distribution of burst and gap lengths. The error characteristics produced by the model are:-

$$\text{Bit error rate} = \frac{P_{1,2}}{(P_{1,2} + P_{2,1})} \cdot p_t$$

$$\text{Mean burst length} = \frac{P_{1,2}}{P_{2,1}}$$

$$\text{Mean gap length} = \frac{P_{2,1}}{P_{1,2}}$$

Probability of a burst length of length m

$$P_b(m) = P_{1,2} \cdot P_{2,1} \cdot P_{2,2}^{(m-1)}$$

A more realistic telephone channel model, which can produce

mixed random and burst errors, was suggested by Elliott (1963). The Gilbert model is extended by allowing the channel states to represent low and high error conditions (Figure 4.e(ii)) rather than the original no error/ error states.

If p_{t1} and p_{t2} represent the transition probabilities for states 1 and 2 respectively, the bit error rate is given by:-

$$\text{Bit error rate} = \frac{P_{2,1}}{P_{1,2}} p_{t1} + \frac{P_{1,2}}{P_{2,1}} p_{t2}$$

Cain and Simpson (1969) described a method for calculating the probability distribution of burst lengths within a block of defined length for the Elliott channel, and applied this to the performance analysis of burst error detecting codes. Others (for example Blank and Trafton 1973) have produced models based on Markov processes with more states. Complex models require the estimation of a number of parameters, and require considerable computation per bit if used for simulation.

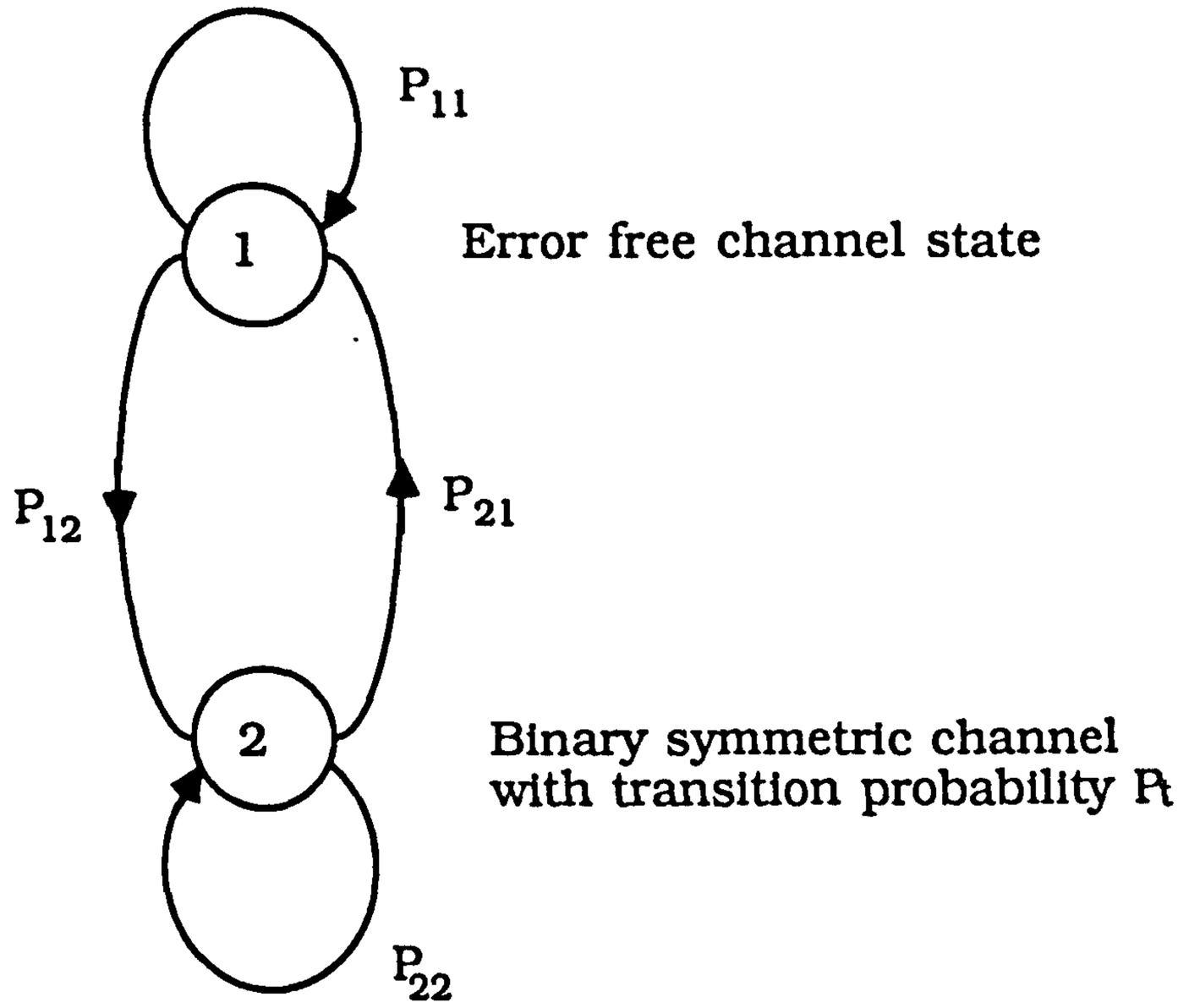

Figure 4.e(i) Gilbert model of a burst channel.

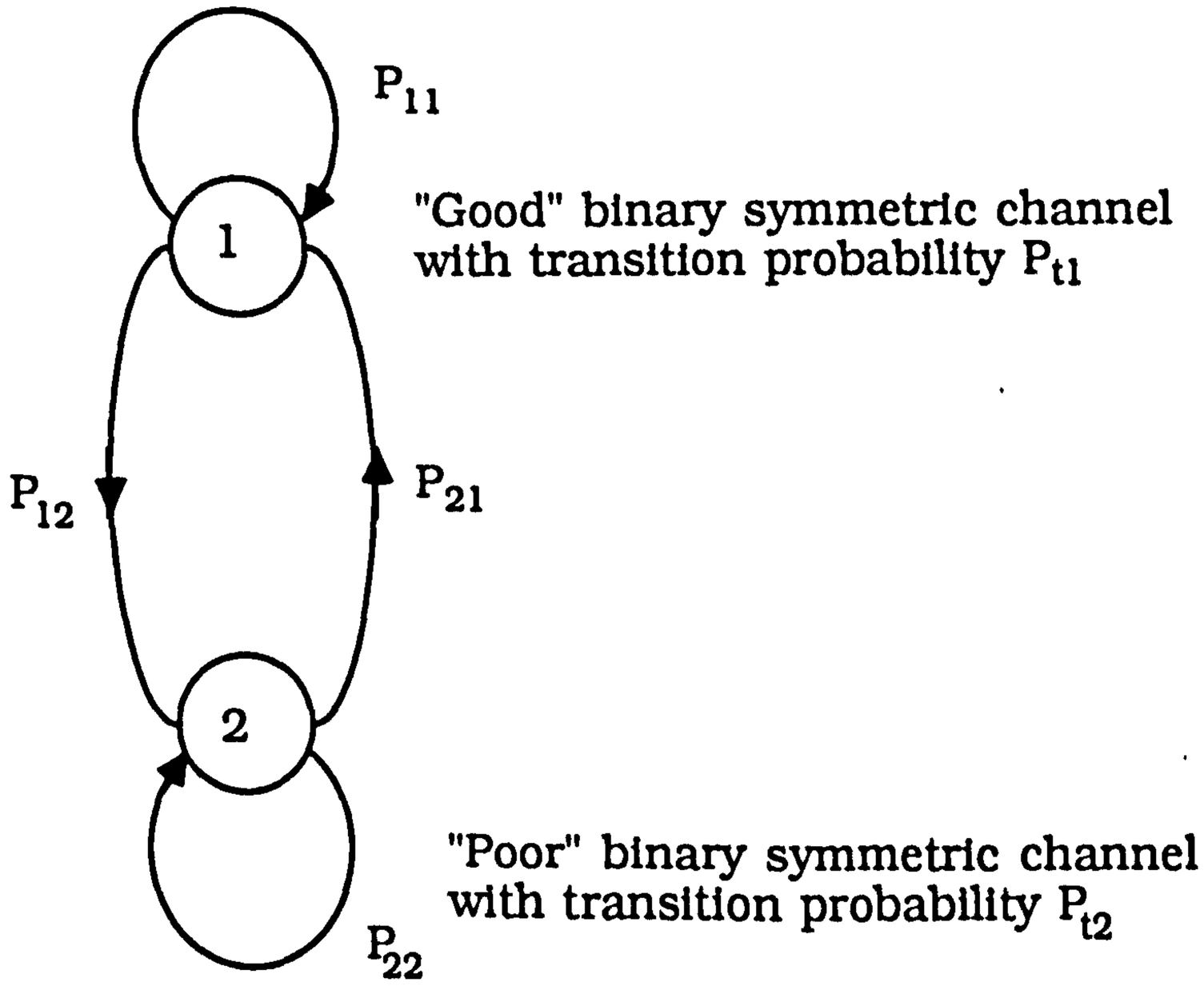

Figure 4.e(ii) Elliott model of the telephone channel

An alternative approach to the Markovian model, which is rather more convenient for simulation work (Drajic 1984), relies on a stochastic process which models inter-error gaps. This method can model a burst error channel, and has the advantage that each calculation stage corresponds to a number of transmitted bits, whilst the Gilbert class of models produce only one bit per calculation stage. Both approaches are analytically tractable, and have been shown capable of closely approximating recorded error data.

Mertz (1961a, 1961b) found that the inter-error gap lengths on the telephone channel could be approximated by a hyperbolic distribution. Mertz proposed a model of the form

$$P(\text{gap of length } g) = \frac{h}{(g+h)}$$

This has no mean value but a mean can be calculated under the assumption that measurements are taken over a finite period. Mertz fitted curves to two sets of data and obtained a reasonable fit.

A Pareto distribution of gap lengths was proposed by Berger and Mandelbrot (1963). The model is based on the assumptions that successive gaps are independent, and are produced by a renewal process (Cox 1962). The channel error data used for the study was obtained from joint IBM / Deutche Bundespost experiments in 1961, using 1200 bit per second modems employing phase and frequency modulation.

This model was also studied by Sussman (1963), who derived expressions for the probability of m errors in n bits, allowing comparison with the results of Alexander et al (1960) and Fontaine and Gallager (1961).

Lewis and Cox (ibid) however, tested Berger and Mandelbrot's model on further data samples from the IBM/ Deutche Bundespost experiments, and found that the hyperbolic/Pareto distribution does not hold for the whole range of gap lengths. In addition, a strong positive correlation between adjacent gaps was observed.

A composite renewal process was suggested by Muntner and Wolf (1968), and applied to the performance analysis of error correction schemes.

It is often sufficient to model one aspect of the error distribution rather than to produce a model which completely emulates the channel. Two particular properties of the channel are of interest in evaluating error control schemes, the *local* error distribution, which is described by the pattern of errors, burst length and weight, whilst the global distribution relates more to the probability of errors within a block, and the gap length distribution.

A block error model consists essentially of a function relating block length and probability of error for some given minimum error weight. Two models will be used in later sections, the first obtained from data given in Balcovic et al (1971), and the second based on the data given in Lewis and Cox (ibid).

The model based on results from the A.T. & T. 1969-70 survey, is based on the measured probability of m or more errors in a block of length n at 1200 bits per second (Figure.14 in Balcovic). From observation, the curves given for the logarithm of the probability against the logarithm of the block length for small values of m, appear straight with approximately equal steps between curves for increasing values of m, and hence can be reasonably approximated by a function of the form

$$\log(P) = k + a \log(n) + b \log(m)$$

The parameters k, a and b, were estimated using the method of least squares on values of log(P) and log(n) obtained from clearly identifiable points on the graph of $P(\geq m,n)$ for 1200 bits per second from Balcovic (ibid). The resulting function is:-

$$P(\geq m, n) = n^a m^b c$$

where a = 0.87, b =-1.66, c = 1/98700 for 1200 bits per second operation. This will be referred to as the Bell model.

The channel error models discussed above offer a wide range of mathematical approaches for the simulation and analysis of error control schemes. The telephone channel has however been described as difficult to model with any degree of certainty, and the selection of a model on which to base the following analysis is not easy. One approach to the evaluation of error control schemes, is to

consider their performance on several different channels. A scheme that performs well on all the models could reasonably be expected to behave well on the telephone channel.

The following models will be used in the following sections for performance comparison:-

(i) The Binary Symmetric Channel, which has well known properties and permits comparison of the results obtained with many published results. Transition probabilities of 0.001 and 0.0001 will be used, as these represent fairly poor telephone channel conditions. Higher error rates would result in loss of synchronization by the modem, and hence loss of the line or automatic fallback to a lower transmission rate.

(ii) The Bell model based on the A.T.& T. 1969-70 survey results, which represent a low error rate channel.

(iii) Lewis and Cox (ibid) data, recorded gap length information which may be used directly. Although the data was obtained using an early modem, and would not be identical to the results obtained with a modern modem, it has the advantage of **not** being the result of a model. The effects of scrambling the transmitted data may be introduced by descrambling the error data, as described above in Section 4.3.1.

Figure 4.f shows the relationship of block error probability to block length for the four models. The Bell model shows a very low

block error probability for the range of block lengths shown, whilst the high error rate binary symmetric channel model exceeds a block error probability of 0.5 at a block length of 693 bits. The difference between the high error rate BSC and the Lewis and Cox data is surprisingly large, as the bit error rates differ by only thirty percent. This is due to the burstiness of the latter.

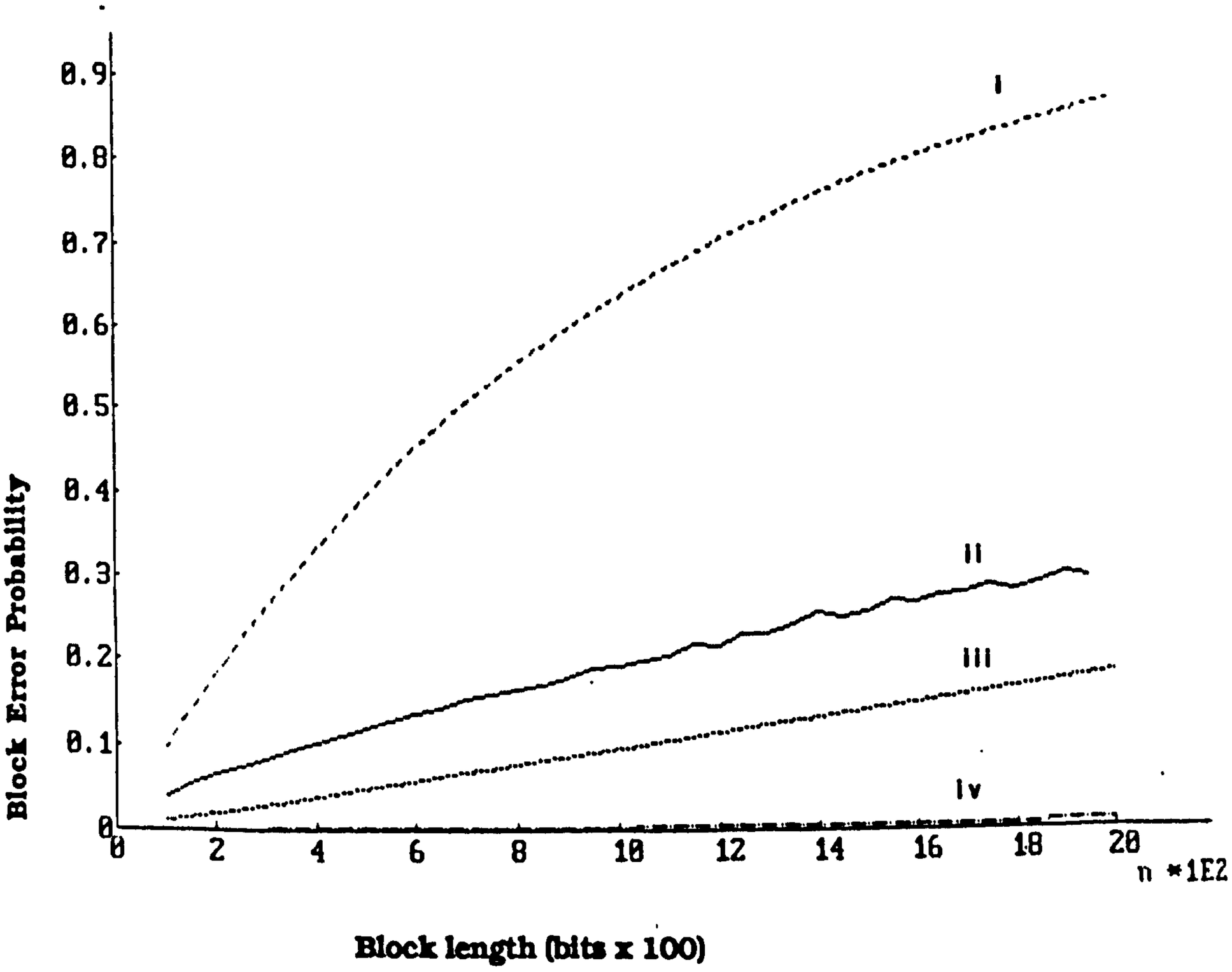

Figure 4.f Relationship between block error probability and block length for the four channel models

(i) binary symmetric channel .. $p_t = 0.001$

(ii) channel data from Lewis & Cox (1966)

(iii) binary symmetric channel .. $p_t = 0.0001$

(iv) model based on A.T.&T survey results

4.4 Channel capacity.

Shannon (1948) derived an expression for the capacity of a transmission channel perturbed by a Markovian noise model. The maximum rate at which information may be transmitted with arbitrarily small probability of error, is given by:-

$$R = H(x) - H_x(y)$$

where $H(x)$ is the entropy of the source X, and $H_x(y)$ the conditional entropy of the received signal y, given x was transmitted. Shannon equates $H_x(y)$ to the capacity of a secondary channel through which additional information is transmitted to allow the correction of errors at the receiver.

For a binary symmetric channel with transition probability p_t, and a source emitting equiprobable binary symbols:-

$$H(x) = -p(0) \log(p(0)) - p(1) \log(p(1)) = 1$$

$$H_x(y) = I_1(0) + I_0(0)$$

$$= I_1(1) + I_0(1)$$

$$= -p_1(0) \log(p_1(0)) - p_0(0) \log(p_0(0))$$

$$= -p_t \log(p_t) - (1-p_t) \log(1-p_t)$$

$$R = H(x) - Hx(y)$$

$$= 1 + p_t \log(p_t) + (1-p_t) \log(1-p_t)$$

This function is plotted in Figure 4.g, for a range of transition probabilities from 0.00001 to 0.1.

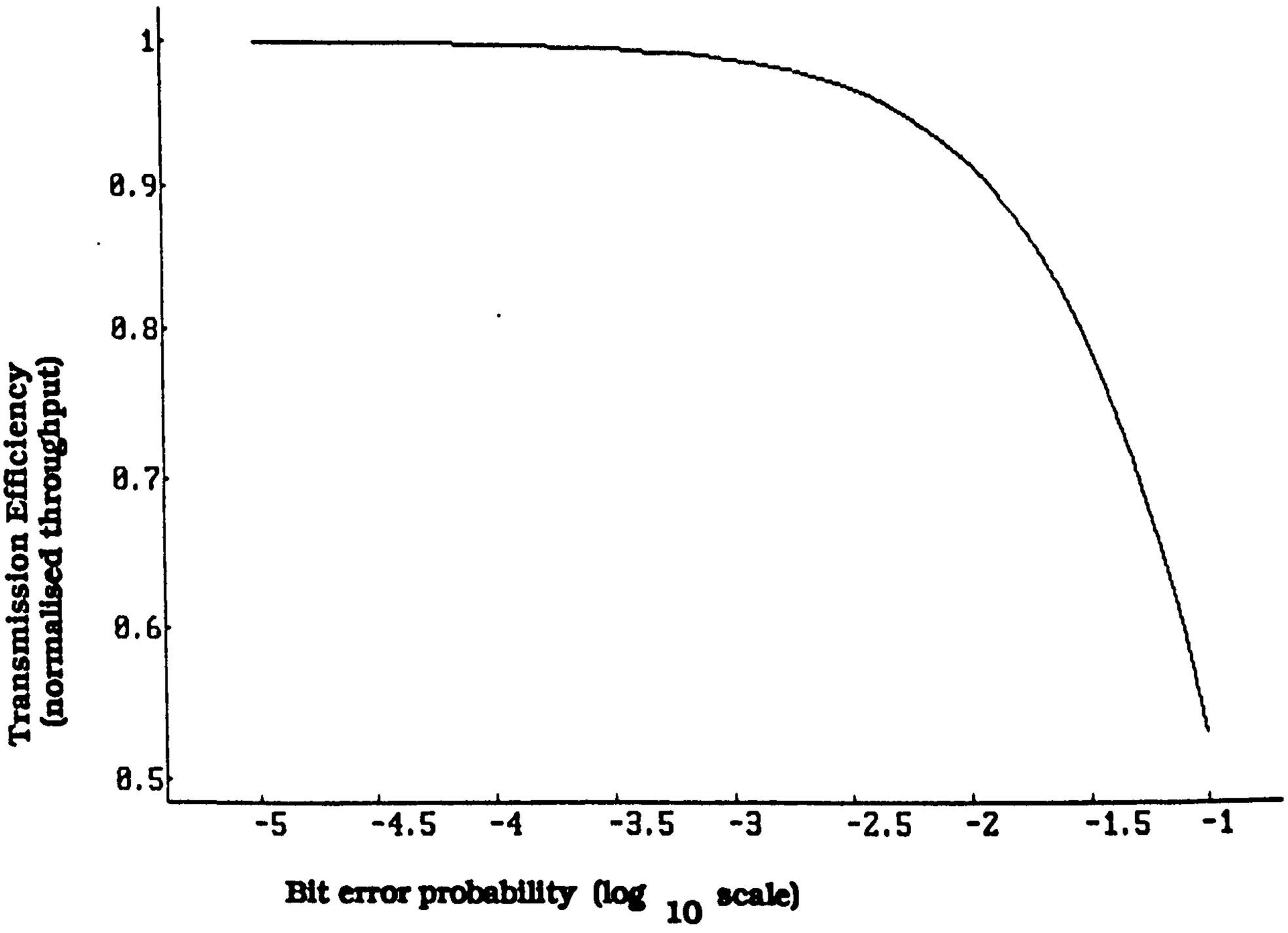

Figure 4.g The Capacity of the Binary Symmetric Channel (as defined by Shannon (1948))

The term *transmission efficiency* will be used in the following discussion to mean the ratio between the measured or simulated system transmission speed and the capacity of the channel when no errors are present. This is perhaps slightly misleading in that the

efficiency should be measured with respect to the true channel capacity, as defined above. The justification is, however, contained in the comment of Burton and Sullivan (ibid) quoted in Section 4.1, namely that the capacity of the real telephone channel is not known.

4.5 Error control.

The aim of error control is to minimize the effect of channel errors on transmitted data. Unfortunately, this is usually accomplished at the expense of reducing the transmission rate. The design of a good error control scheme requires knowledge of the channel conditions, typical error distribution and error rate, the reduction in error rate required, and the ratio between channel transmission and minimum tolerable source symbol speed (code rate).

One error control scheme commonly used on the telephone network, is *fallback switching*. In general, the error rate obtained on a communications system is a function of the transmission speed. If the bit rate is reduced (typically halved), the error rate is markedly reduced. This method has several disadvantages, firstly that the transmission speed is not optimized for the required error rate, and secondly, that it is not readily apparent when the channel conditions have improved, hence the system will operate at low speed unnecessarily.

Another method that is widely used, is forward error correction. Parity bits are added to the transmitted data in order that a limited number of errors may be detected and corrected. Usually the proportion of parity bits is fixed, and hence some loss of channel capacity occurs regardless of the channel condition. The

function used to generate the parity bits must be designed with knowledge of the channel error distribution. Often codes are designated *random* (i.e. suitable for the binary symmetric channel) or *burst* correcting (i.e. suited to a burst channel such as that approximated by the Gilbert model).

The third method of error control, which is growing in importance, is feedback error correction, or *automatic retransmission request* (ARQ). Data is formed into frames to which are appended parity words for error detection. A frame is transmitted and, if corrupted, the receiver can detect the presence of errors and request the retransmission of the erroneous frame. This has two advantages, firstly that the transmission rate depends on channel conditions, and secondly that the residual error rate depends on the number of errors that can be *detected* with the additional parity bits, which is far greater than the number of errors that could be *corrected* with the same number of parity bits.

4.5.1 Linear block codes.

In an (n,k) block code, the source output is divided into message blocks of length k bits. The encoder maps each of the 2^k possible k bit blocks onto a unique n bit codeword. The decoder performs the reverse mapping from an n bit received binary vector to a k bit message. A t error correcting code defines the message-codeword-message mapping so that the k bit message will be correctly recovered by the decoder if the codeword and received vector differ in no more than t bit positions.

Several important classes of code exist, of which a large number belong to the general family of linear block codes. The cyclic codes are a subclass of linear codes, and are widely used as, firstly they are readily implemented in hardware using shift registers, and secondly, the underlying algebraic structure provides the means for developing decoding techniques. Examples of cyclic codes are the BCH codes, defined in the binary field, and Reed-Solomon codes defined for non-binary code symbols.

The following definitions are needed for the ensuing discussion, and are widely known (e.g. Lin and Costello 1983).

(i) A binary code is *linear* iff the modulo 2 sum of two codewords is also a codeword.

(ii) A linear code is *cyclic* iff the cyclic shift of any codeword is also a codeword.

(iii) A code is *systematic* if the codeword consists of the k bit message with (n-k) parity bits appended.

(iv) The *rate* of an (n,k) block code is k/n.

(v) The *Hamming distance* between two binary codewords is the number of bit positions in which the codewords differ.

(vi) The *minimum distance* of a code, d_{min}, is the minimum of the distance between any two codewords, where the distance may be the Hamming distance.

(vii) A block code can correct all errors of weight t or less if the minimum distance of the code is at least 2t+1.

(viii) A block code can correct $2^{(n-k)}$ error patterns.

The encoding of k message bits to the n codeword bits of a linear block code may be represented as the multiplication of the k-tuple binary message vector by a (k,n) *generator matrix* $\underline{G}$. The multiplication and addition are carried out modulo 2, i.e. addition is an exclusive-OR, and multiplication an AND.

$$\underline{C} = \underline{M} \cdot \underline{G}$$

where $\underline{C}$ is an n-tuple, $\underline{M}$ is a k-tuple, and $\underline{G}$ is (k,n).

In transmission, the codeword may be corrupted by an *error vector* $\underline{\mathbf{E}}$ (also an n-tuple), giving the received vector $\underline{\mathbf{V}}$.

$$\underline{\mathbf{V}} = \underline{\mathbf{C}} + \underline{\mathbf{E}}$$

Decoding may be represented similarly as the matrix multiplication of the n-tuple received vector $\underline{\mathbf{V}}$, by the (k,n) *parity check matrix* $\underline{\mathbf{H}}$. The *syndrome* $\underline{\mathbf{S}}$ of the received vector is an (n-k)-tuple vector, defined as:-.

$$\begin{aligned}\underline{\mathbf{S}} &= \underline{\mathbf{V}} \cdot \underline{\mathbf{H}}^T \\ &= (\underline{\mathbf{C}} + \underline{\mathbf{E}}) \cdot \underline{\mathbf{H}}^T \\ &= \underline{\mathbf{M}} \cdot \underline{\mathbf{G}} \cdot \underline{\mathbf{H}}^T + \underline{\mathbf{E}} \cdot \underline{\mathbf{H}}^T \\ &= \underline{\mathbf{E}} \cdot \underline{\mathbf{H}}^T\end{aligned}$$

as $\underline{\mathbf{G}} \cdot \underline{\mathbf{H}}^T = [\underline{\mathbf{0}}]$

The syndrome is capable of representing $2^{(n-k)}$ error patterns, including the zero error vector. As there are in fact 2^n possible error patterns, it is apparent that each syndrome vector corresponds to 2^k error patterns which are indistinguishable to the decoder. For error detection this coincidence only matters when the error patterns correspond to the zero error vector.

Error correction codes are usually constructed so that most probable error patterns each have a one to one correspondence with a unique syndrome vector; this obviously requires knowledge of the error pattern distribution, generally assumed to be random (BSC), burst (any pattern of errors with burst length less than some maximum, as defined in Section 4.3.1), or mixed burst and random.

Cyclic codes may be described in the terms used above, as

linear block codes, however they are more usually represented in terms of polynomial operations within a finite field. The *generator polynomial G(X)* of a cyclic code, is a polynomial in the Galois Field *GF(X)* of degree (n-k).

Encoding may be represented in several ways, the more widely used being the systematic encoder, in which the parity bits are appended to the message *m(X)* . The encoder multiplies the message by $X^{(n-k)}$, then divides by *G(X)*, giving a remainder *b(X)*. When the remainder is added to the term $m(X).X^{(n-k)}$, the resulting codeword is divisible by *G(X)*. Hence, if the receiver also divides by *G(X)*, and no errors are present, the remainder (the syndrome) should be zero.

The transmitted codeword is *c(X)*, where:-

$$c(X) = b(X) + m(X)\ X^{(n-k)}$$

and $c(X) = a(X)\ G(X)$

During transmission, the codeword is corrupted by error vector *e(X)*, giving the received code vector *r(X)*:-

$$r(X) = c(X) + e(X).$$

The decoder divides the received vector by the generator polynomial *G(X)* to obtain the syndrome *s(X)* :-

$$\frac{r(X)}{G(X)} = \frac{c(X)}{G(X)} + \frac{e(X)}{G(X)}$$

$$= a(X) + e'(X) + \frac{s(X)}{G(X)}$$

If $s(X)=0$ the receiver assumes that no errors are present, however $e'(X)$, which represents any component of $G(X)$ in the error polynomial, will be non-zero. This corresponds to the error vector being a codeword, which according to condition (i) above will simply convert the transmitted codeword into another valid codeword.

4.5.2 Convolutional codes.

Convolutional codes differ from block codes in that the encoder contains memory, and the n bit encoder output depends not only on the k input bits but on m previous input blocks. Decoding algorithms tend to be complex, but with the aid of soft decision information (the quantized output from the demodulator), considerable coding gains can be made.

4.5.3 Random error correcting codes.

The decoder of an error correcting code has to deduce from the syndrome of the received vector the error pattern that has modified the transmitted codeword. The design of a decoder is generally complex, hence codes are often selected on the basis of the simplicity of the decoder rather than on absolute performance.

A general bound on the performance of a t random error correcting code is the Hamming (1950) bound:-

$$2^{(n-k)} \geq \sum_{i=0}^{t} {}_nC_i$$

This is readily apparent from the observation that the number of correctable error patterns cannot be greater than the number of syndromes.

The effectiveness of a random error correcting code on the binary symmetric channel is measured by the probability of uncorrected error. For an (n,k,t) code:-

$$\text{Probability of uncorrected error} = 1 - \sum_{i=0}^{t} {}_nC_i \cdot p_t^i \cdot (1-p_t)^{(n-i)}$$

The Hamming (ibid) codes are the earliest class of linear error correcting code. For any integer q>2, there exists a Hamming code such that:-

codeword length	$n = 2^q - 1$
number of message symbols	$k = 2^q - q - 1$
number of parity check symbols	$q = n - k$
minimum distance	$d_{min} = 3$
number of correctable errors	$t = 1$

These codes are extremely simple to decode, and may be put into cyclic form.

A class of cyclic codes capable of correcting multiple random errors are the BCH codes (Hocquenghem 1959, Bose 1960). For any integers q>2, and t there exists a BCH code such that:-

codeword length	$n = 2^q - 1$
number of parity check bits	$n - k \leq q.t$
minimum distance	$d_{min} \geq 2t + 1$
number of correctable errors	$t < 2^{(q-1)}$

BCH codes are more complex to decode than Hamming codes. Sinha (1983) describes a decoder for a (31,21,2) code, which uses a table look-up approach to determine error positions from the syndrome; this method is only suitable for small values of n and t. Blahut (1984) describes a more complex approach based on a universal decoder, capable of decoding BCH codes and the more general Reed-Solomon codes.

There are many more random error correcting codes, both linear block code, cyclic code and important classes of non-block code such as the convolutional code.

4.5.4 Burst error correcting codes.

The Reiger (1960) bound provides an upper bound on the burst error correcting capability of an (n,k) linear code. The maximum correctable burst length b_{max} is bounded by:-

$$b_{max} \leq \frac{(n - k)}{2}$$

Although a number of burst error correcting codes have been found, two important types will be discussed, Fire codes and interleaved codes.

Fire codes (Lin 1983) were the first class of cyclic burst error correcting code. The number of parity bits for a code capable of correcting all bursts of length b_{max} or less is given by:-

$$n - k = q + 2\, b_{max} - 1$$

thus

$$b_{max} = \frac{(n - k - q + 1)}{2}$$

If a number of consecutive codewords of some encoded message are stored and, rather than transmitting the codewords in sequence, one symbol from each codeword is transmitted in turn until every symbol from every codeword is sent, the code is said to be *interleaved* (Kitces 1963). If the basic code used to encode each word is a t error correcting code, and J words are interleaved,

then the interleaved code can correct any burst of length *J*.t or less, any random error pattern of weight t or less, and many random error patterns of weight *J*.t or less.

thus $b_{max} = J \,.\, t$

4.5.5 Burst and random error correcting codes.

If both random and burst errors are present on a channel, neither a random or burst error correcting code will perform effectively. Codes may however be designed to correct both types of error pattern. The interleaved coding technique described above is well suited to channels of this type. Lin (ibid) describes a number of other methods, such as product codes, concatenated codes and the Reed-Solomon codes.

4.5.6 Error detection.

The codes most commonly used for error detection in communication systems are the simple (k+1,k) parity check, and various binary cyclic codes with 8, 16 or 32 parity bits but variable k. The (k+1,k) code can detect only single errors, and is not suited for use with large k. For this purpose, cyclic codes are almost invariably used, although termed cyclic redundancy check (CRC), rather than

cyclic error detecting codes. The following discussion will relate to binary cyclic block codes of the form (k+p,k) where the number of message bits k, is not fixed. One widely used cyclic error detection code is that used in CCITT Recommendations V41 and X25:-

$$G(X) = X^{16} + X^{12} + X^5 + 1$$

For an (n,k) cyclic code, the probability of undetected error may be calculated. As shown in Section 4.4.1 the undetectable error patterns of a cyclic code correspond exactly to the set of 2^k codewords (including the 0..00 codeword). There will therefore be 2^k-1 undetectable error patterns out of 2^n possible patterns. Under the assumption that all error patterns are equiprobable:-

$$\text{Probability of undetected error} = \frac{2^k-1}{2^n} \approx \frac{1}{2^{(n-k)}}$$

If the weight distribution of the channel error patterns is known, the probability of undetected error can be more accurately calculated. On the binary symmetric channel, the probability of undetected error for an (n,k) code with minimum distance d_{min} is:-

$$\text{Probability of undetected error} = 1 - \sum_{i=0}^{d_{min}-1} {}_nC_i \cdot p_t^i \cdot (1-p_t)^{(n-i)}$$

The error pattern may be subject to extension by, for

example, the descrambling process discussed in Section 4.2. This may be represented by the multiplication of the channel error polynomial $e(X)$ by a polynomial $u(X)$. For a transmitted codeword $c(X)$, the received polynomial will be:-

$$r(X) = c(X) + e(X).u(X)$$

decoding the received polynomial,

$$\frac{r(X)}{G(X)} = \frac{c(X)}{G(X)} + \frac{e(x).u(X)}{G(X)}$$

$$= a(X) + e'(X) + \frac{s(X)}{G(X)}$$

The syndrome $s(X)$ will be zero if either $e(X)$ or $u(X)$ are zero, or if either $e(X)$ or $u(X)$ are multiples of $G(X)$ (i.e. codewords). Thus it is essential that the scrambler polynomial and generator polynomial are mutually prime.

4.5.7 Error correction using retransmission.

Automatic repeat request (ARQ) error control systems offer an efficient and reliable alternative to forward error correction; the technique is generally attributed to Van Duuren (1951).

A feedback path is provided between the decoder and encoder (either full or half duplex), and is used for the communication of control information. The message is segmented into blocks to which are appended parity bits for error detection. A

block is transmitted, and possibly corrupted by transmission errors. The decoder checks for the presence of errors in a received frame and sends an acknowledgement if no errors were detected, or a retransmission request if errors were present. The transmitter either transmits the next frame in sequence or retransmits an earlier frame.

The effective code rate of the system depends on the prevailing channel conditions. Under poor conditions, the number of retransmissions will increase, lowering the effective rate. For a good channel however, no retransmission will be needed, and the transmission rate may be very high. In addition, the uncorrected error rate depends on the error detection rather than the error correction properties of the code used.

The implementation complexity is fairly low in comparison with that of forward error correction, and hence the technique is extremely widely used. ARQ is ideally suited to software implementation, as the logic is extremely simple but the memory requirement significant. There is considerable support available for microprocessor implementations, in the form of interface integrated circuits which perform error detection code encoding and decoding as well as providing the serial interface.

4.6 Summary

The nature and source of transmission errors on the telephone channel have been discussed. The reduction of error rate may be achieved through the use of error control techniques such as forward error correction, and automatic repeat request. The design of these error control systems requires some knowledge of the error distribution, which is generally represented in the form of a mathematical model of the channel error process. The difficulty with the telephone channel is the wide variety of conditions that may exist, rendering the design of efficient error control schemes problematic.

The advantages of automatic repeat request over forward error correction have been discussed briefly. In the following chapter ARQ will be examined more fully, and a number of hybrid schemes discussed.

5 ARQ ERROR CONTROL

5.1 Introduction

The most widely used form of error correction is *automatic repeat request* (ARQ). The message is formed into blocks or frames, to which are appended parity words for error detection. A frame is transmitted, the receiver checks the received frame for errors and either requests retransmission or confirms correct reception via a return path. Communication is therefore bidirectional, but may be half or full duplex. A typical frame is shown in Figure 5.a, which shows a control field, a data field and a set of parity bits for error detection.

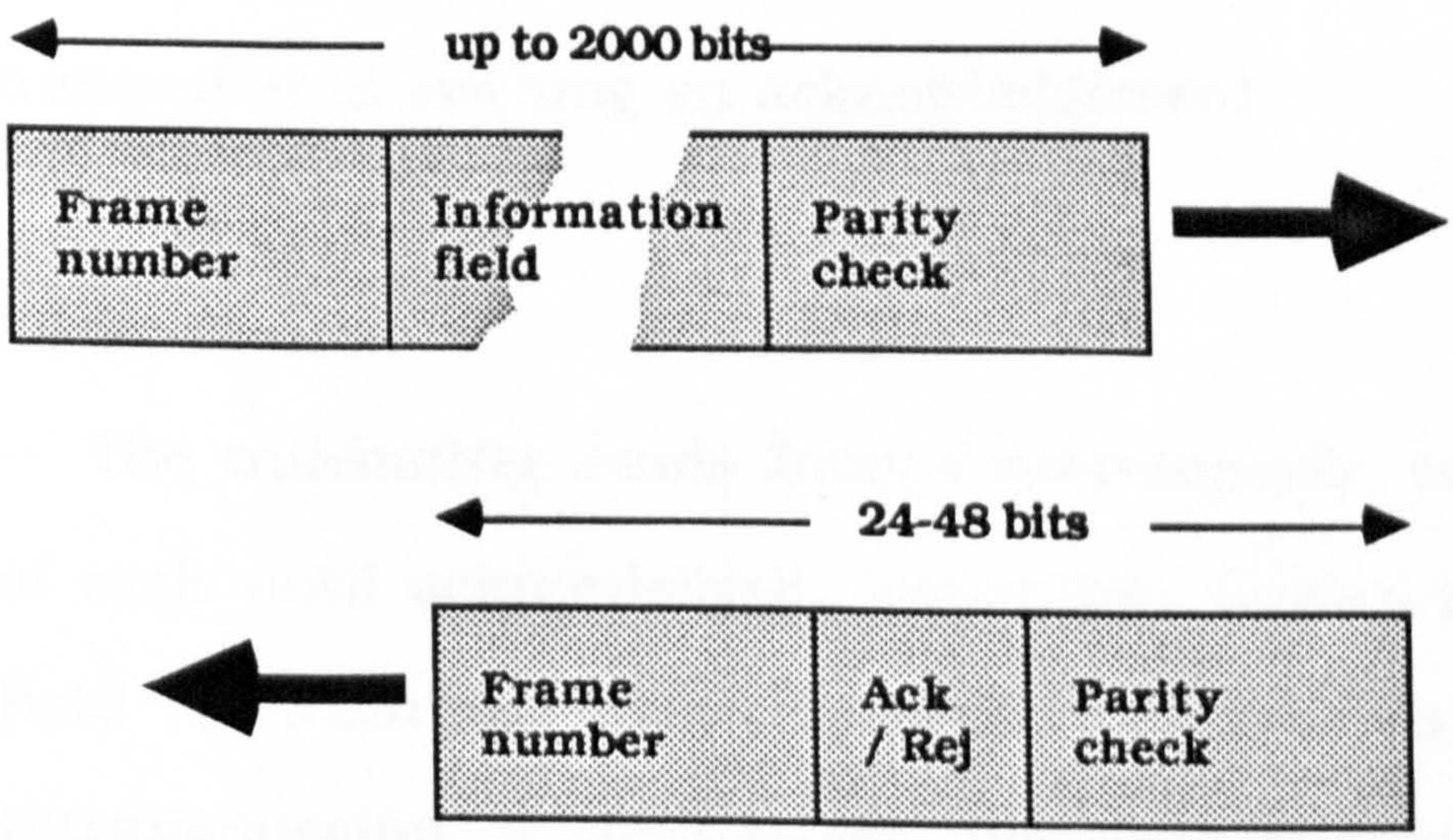

Figure 5.a General frame format for an ARQ information frame and acknowledgement frame.

Since the earliest application of the automatic retransmission principle (Van Duuren 1951), a large number of improvements have been made. There are now three basic types of ARQ, Stop and Wait (SW) termed Idle RQ by Benice and Frey (1964), Go Back N (GBN), and Selective Repeat (SR).

(i) Stop and Wait.

The transmitter sends one frame, retaining a copy in case of a retransmission request, and then waits for an acknowledgement. If a positive acknowledgement is received, the transmitter sends the next frame. If a negative acknowledgement is received (indicating that the frame was received in error), or no response observed within some predetermined interval (due to the transmitted frame or acknowledgement being heavily corrupted or lost) the transmitter resends the frame. This technique is simple to implement, but inefficient for channels with appreciable delay, as the forward channel is idle whilst the transmitter is awaiting an acknowledgement.

(ii) Go Back N.

The transmitter sends frames continuously, retaining copies of each until acknowledged, hence the frames need a control field containing the transmitted frame number. If a retransmission is requested, the transmitter resends the corrupted block, and all subsequent blocks. The transmitter thus needs storage sufficient to hold copies of frames until acknowledged. The number of outstanding frames will depend on

current channel conditions and round trip delay, and will be limited by the available storage capacity and the range of the frame numbering system (typically modulo 8,16 or 128). On channels with considerable delay, such as satellite channels, the efficiency of GBN ARQ drops due to the large number of frames that must be retransmitted each time a frame is rejected.

(iii) Selective Repeat.

The transmitter sends numbered frames continuously, as with Go Back N, but only retransmits those blocks which are negatively acknowledged. The receiver must therefore be able to store correct frames received after erroneous ones, in order that the correct data sequence is maintained. This approach is more efficient than either Stop and Wait or Go Back N ARQ, and has particular advantages on channels with long delay. The main drawback is the added complexity of the receiver due to the increased buffer requirement.

Figure 5.b provides a comparison of the relative transmission efficiency (defined in Section 5.2 below) of the three ARQ types, for a binary symmetric channel with delay from 0 to 5000 bit intervals. The poor performance of Stop and Wait ARQ and the excellent performance of Selective Repeat ARQ, with increasing delay can be clearly observed. It should however be noted that the limited receiver storage capacity and the range of the frame numbering system would in practice reduce the performance of Selective Repeat ARQ for channels with extended delay.

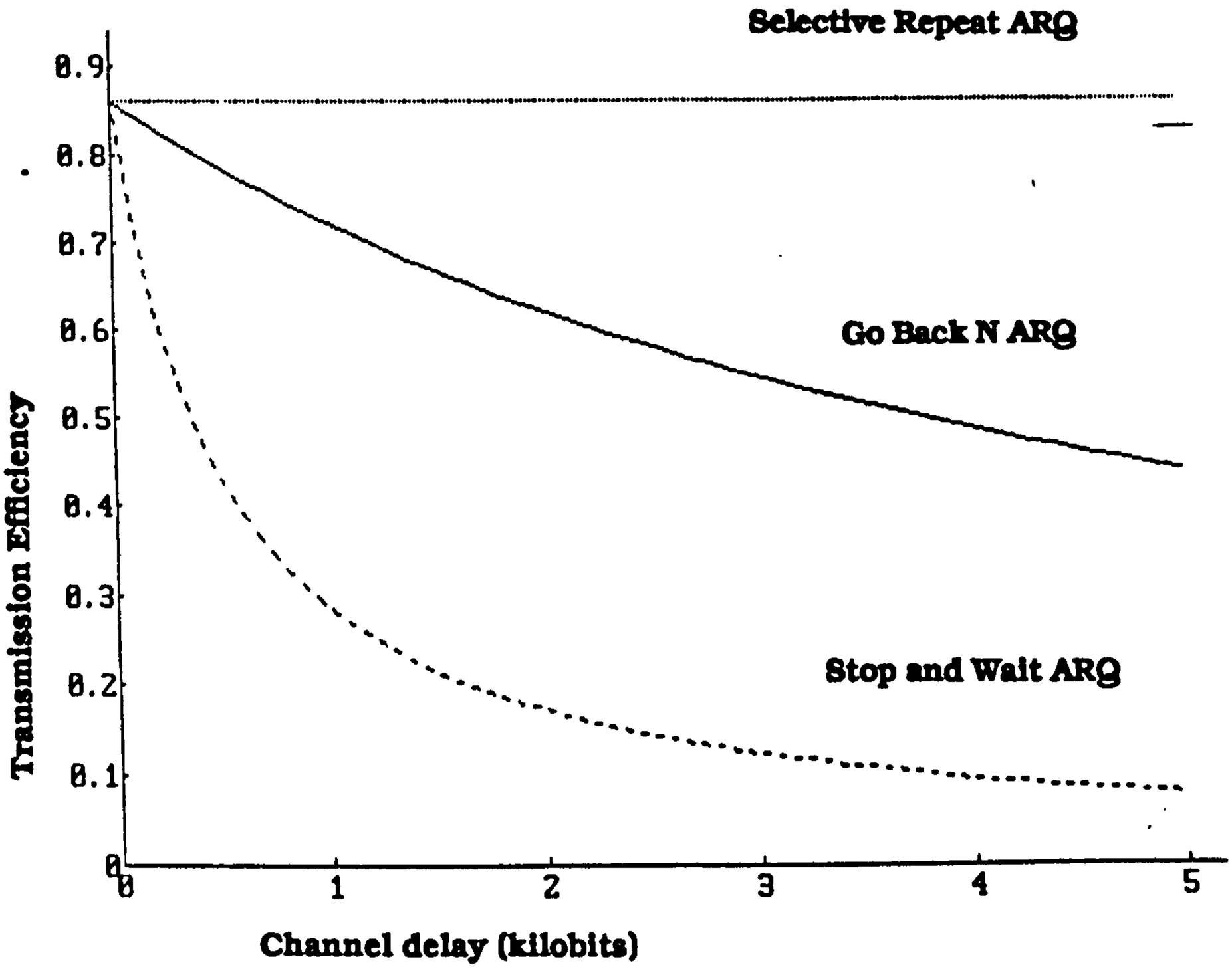

Figure 5.b Transmission Efficiency of Stop and Wait, Go Back N and Selective Repeat ARQ on the binary symmetric channel (with BER of 0.0001) for a range of channel propagation delays.

On high error rate channels ARQ becomes inefficient as the proportion of frames needing retransmission increases, and an additional stage of error control is often added. Between the ARQ transmitter and channel, a forward error correction (FEC) encoder is inserted, and a decoder placed before the corresponding receiver.

The use of forward error correction effectively reduces the channel error rate, and hence the number of retransmissions requested by the ARQ receiver. However, the effective bandwidth of the channel is reduced by the rate of the code, and hence the use of hybrid ARQ/FEC systems must be carefully considered.

This chapter discusses the choice of ARQ technique. Initially a number of hybrid ARQ schemes are described, and their performance compared under a range of channel conditions. The criteria for selecting frame length are discussed, and a method given for selecting the optimum length.

There are certain practical difficulties in the design of hybrid ARQ which are described, and an adaptive ARQ scheme proposed which overcomes these.

The uncorrected error rate (*residual error rate*) of an ARQ scheme is generally low, as it depends on the error detecting rather than error correcting properties of a code. This aspect of ARQ system performance is discussed, and the effects of errors on ARQ frame synchronization considered.

The chapter concludes with the selection of several ARQ schemes appropriate to use on the telephone channel.

5.2 Performance analysis of ARQ

The main performance parameters used to compare ARQ systems are throughput efficiency and uncorrected error probability. These are both dependent on the frame length and channel error conditions (i.e. error rate and distribution, and delay).

For a frame of length k, containing (k-h) data bits, and h header bits (including error detection bits) the efficiency is given by:-

$$\text{Efficiency} = \frac{(k-h)}{k} \frac{1}{T}$$

where T, the average number of transmissions required to successfully send one frame, is dependent on the protocol used, and the channel delay. k is used for the frame length rather than n to ensure consistency with the equations that will be given for hybrid ARQ below.

For the three basic types of ARQ, Stop and Wait (SW), Go Back N (GBN), and Selective Repeat (SR), the value of T is given below (these expressions are derived in Appendix C).

$$T_{SW} = \frac{N}{(1-P_E)}$$

where P_E is the probability of errors being detected in the received

frame P(>0,k), and N is the round trip acknowledgement delay (in frame intervals).

$$T_{GBN} = \frac{N P_E}{(1-P_E)} + 1$$

where N is, in this instance, equivalent to the number of frames retransmitted as a result of a negative acknowledgement. It is sometimes assumed that N is integer, however this is more applicable to a half duplex system. In a full duplex system, the transmitter may, on receiving a retransmission request, immediately discontinue sending the current frame, which could lead to non-integer values of N.

$$T_{SR} = \frac{1}{(1-P_E)}$$

The use of Go Back N ARQ will be assumed from this point onwards, as this is the most widely used method, being fairly simple to implement, and fairly efficient for channels with moderate delay. Selective Repeat does provide better performance with increasing delay, but at the cost of an extensive amount of buffer storage at the receiver.

The assumptions made in the performance comparison are:-

(i) The time taken to acknowledge a frame is twice the channel

delay plus a delay equivalent to one frame length (the receiver must completely receive a frame before it can detect errors). In the event of a retransmission request being received, the transmitter immediately abandons the frame currently being transmitted and starts retransmission of the rejected frame. This has the effect of making the N (i.e. the number of frames to go back) equal to 1+2D/n, rather.than the integer part thereof as often assumed (correctly for a half duplex channel).

(ii) The return channel is assumed error free. This is often justified by the assertion that the acknowledgement frames are short, and hence have a low probability of error. Often however, *piggybacking* is used (Lai 1982), in which the acknowledgements are carried within the header field of a data frame passing in the return direction, i.e. data is flowing both ways. In most cases the information flow, although bidirectional, is not symmetric (Fuchs 1970).

(iii) The error detection code is assumed to detect all errors. Hence the retransmission request probability for a frame is P(>0,k); the validity of this assumption is subject to the discussion in Sections 4.5.6, and 5.8. The effects of undetected errors and of errors in the return channel have been explored by a number of researchers, for example Benice and Frey (1964), Rocher and Pickholtz (1970), Field (1976), and Fujiwara et al (1978).

(iv) The frame sequence number range and transmitter memory capacity are assumed infinite. In practice the frame sequence numbering would be limited, and efficiency slightly reduced as a result. This effect will be further discussed in Chapter 6.

(v) Error correction codes are assumed to be (n,k,t) linear block codes, capable of correcting all error patterns of weight equal to or less than t. It is further assumed that for some rate R and block length n, a code may be constructed that satisfies k = R.n, by for example shortening a BCH code.

(vi) It is assumed that codes exist which satisfy the expression

$$t = \frac{\text{integer}(\ (n-k)\ )}{\text{integer}(\ \log_2(\ n+1\ )\ )}$$

This expression reasonably approximates the error correcting capability of a BCH code.

(vii) A header (frame number and error detection parity bits) size of 48 bits is assumed; this is a fairly typical size for a protocol such as the IBM Synchronous Data Link Control protocol (Donnan 1974).

(viii) Data from the source and source encoder is always available, which is in practice only likely during bulk data transfers. The transmission efficiency is of interest primarily under these conditions, and hence the assumption is not unreasonable. The source would also need to be able to respond to flow control

(Section 6.2.2), due to the inherently variable transmission rate of ARQ systems.

5.3 Hybrid ARQ

A large number of modified ARQ schemes have been developed to suit particular types of channel. Generally, ARQ provides higher efficiency on burst than on random error channels, for a given bit error rate. Forward error correction may be applied within the ARQ to correct most of the random errors, leaving the ARQ system to cope with the remaining errors. Figure 5.c shows a simple hybrid system of this type.

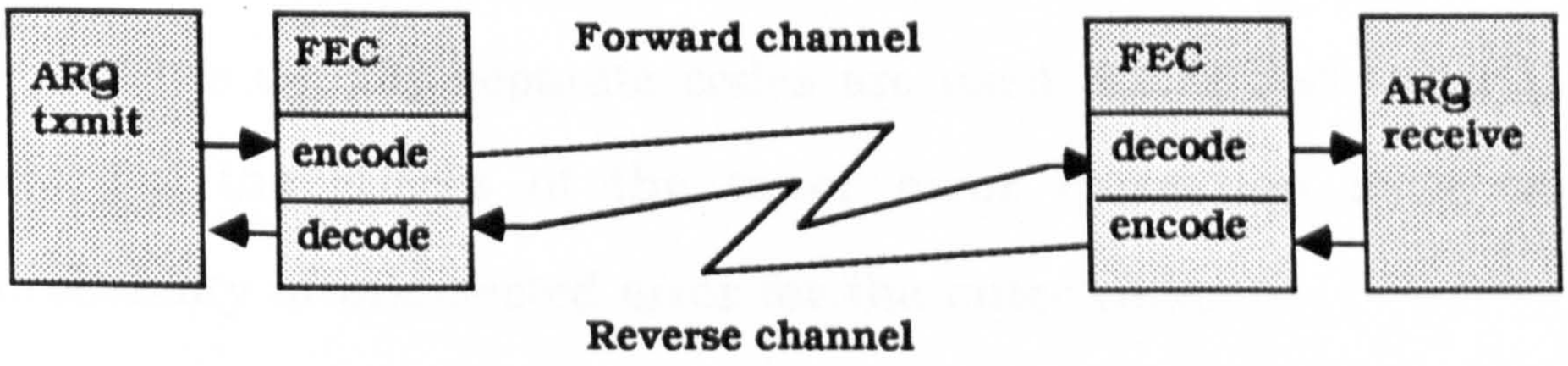

Figure 5.c A simple hybrid ARQ system

The efficiency of ARQ is improved by the effective reduction in channel error rate and hence number of frame retransmissions, however the forward error correction code reduces the effective channel transmission rate by the rate of the code used. Thus the overall efficiency is :-

$$\text{Efficiency} = \frac{k}{n} \frac{(k-h)}{k} \frac{1}{T'}$$

where T' is the mean number of transmissions required per frame, which is reduced by the use of the (n,k) error correcting code.

The error correcting code used within the hybrid ARQ system may also be used for error detection, obviating the need for separate codes (and hence two decoders). Several examples of this type are described below. The minimum distance of a block code used for the correction of up to t errors and the detection of up to d errors (where d>t) (Lin 1983), is:-

$$d_{min} > t + d$$

More usually separate codes are used. Klove and Miller (1984) discuss the effects of the inner error correction code on the probability of undetected error for the outer code.

Two basic types of hybrid ARQ system will be discussed, the Type I, as described above, and the Type II or parity retransmission scheme.

5.3.1 Type I Hybrid ARQ Schemes.

Various authors have described hybrid FEC/ARQ protocols

similar to that described above, Lin (1983) defines these as Type I hybrid ARQ schemes. Figure 5.d(i) shows the general frame format of a Type I scheme. The principal characteristic of this class of hybrid ARQ scheme is that the additional parity bits needed for error correction are transmitted with the frame to which they relate.

Brayer (1968) discusses the use of hybrid ARQ on HF channels, using a non-binary block code for error correction. The channel differed considerably from the telephone channel however, and the results are not comparable with those below.

Rocher and Pickholtz (1970) described several hybrid ARQ schemes, and evaluated their performance on the binary symmetric channel.

They relate their results to performance on a high data rate modem on a voice grade line, and comment that the forward error correcting code rate may be very high, as "*only a few errors need to be corrected to reach a very high reliability while still retaining a reasonable throughput*".

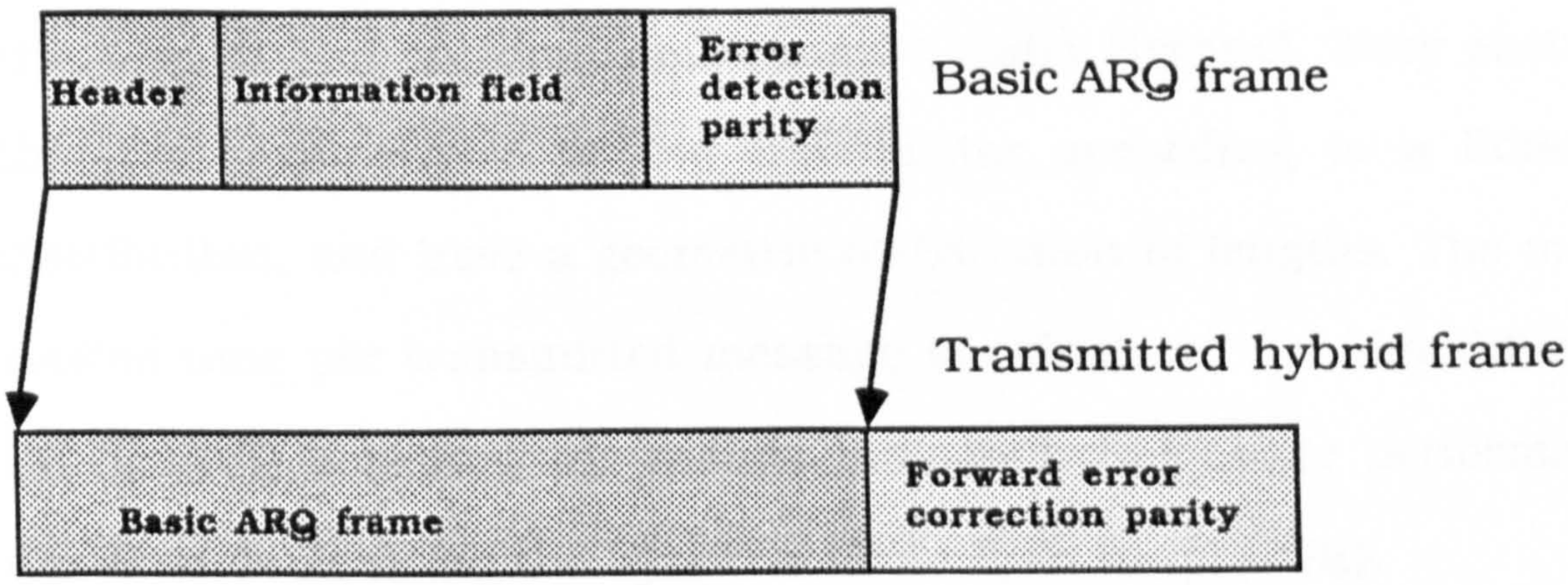

Figure 5.d(i) Type I hybrid ARQ frame structure

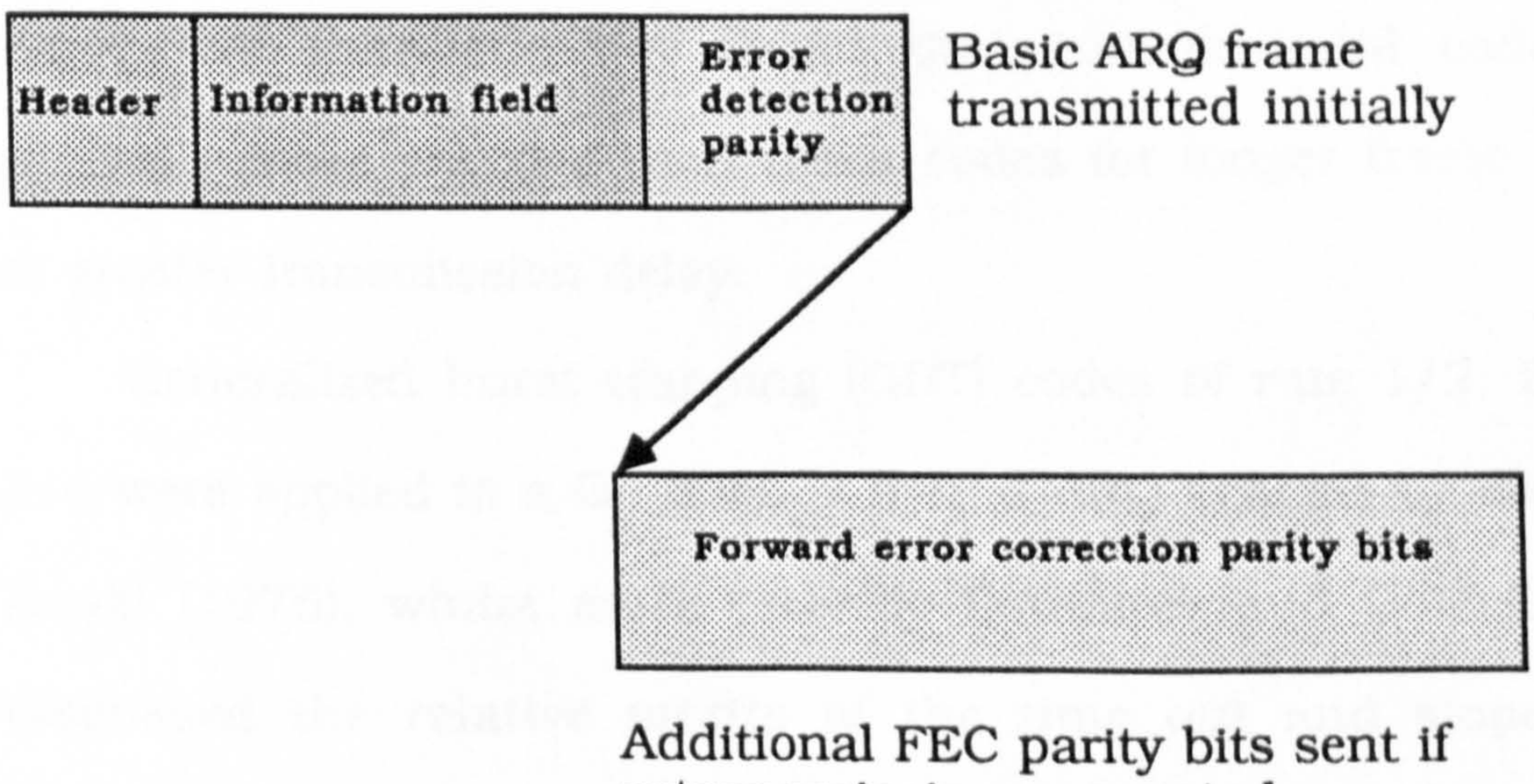

Figure 5.d(ii) Type II hybrid ARQ frame structure

Leung and Lam (1981) describe a hybrid Stop and Wait protocol, using rate 0.84 BCH codes with block lengths from 255 to 2047 bits. They examined the performance of the scheme on both a random error channel (BSC) and a Rayleigh fading channel model characteristic of multipath effects on a radio channel. They assume that messages arrive at the transmitter according to a Poisson distribution, and have a geometric distribution of lengths. The mean wasted time per transmitted message is calculated for four different hybrid ARQ schemes, the results indicated that better performance was obtainable using the hybrid scheme than simple ARQ.

Convolutional codes have been used in a number of hybrid ARQ schemes. The relative merits of block and convolutional codes are discussed by Drukarev and Costello (1982), who compare the efficiency of hybrid Stop and Wait, Go Back N and Selective Repeat ARQ using BCH and rate 1/2 convolutional codes for the binary symmetric channel. They found that convolutional codes were generally more effective than block codes for longer frame lengths or greater transmission delay.

Generalized burst trapping (GBT) codes of rate 1/2, 2/3, and 3/4 were applied to a Go Back N hybrid ARQ scheme by Sastry and Kanal (1976), whilst more recently Drukarev and Costello (1983) discussed the relative merits of the time out and slope control convolutional code decoding algorithms within a hybrid ARQ system.

The rate of the code used for error correction is fairly critical. If a low rate code is used, most errors will be corrected without

using retransmission, but the transmission efficiency is reduced considerably by the code rate. If however a high rate code is used, a smaller proportion of errors will be corrected and thus retransmissions will be more frequent. The code should ideally be selected to maximize the efficiency of the system.

5.3.2 Type II Hybrid ARQ Schemes - Parity Retransmission.

A Type II (Lin 1983) hybrid ARQ protocol employs some form of parity retransmission. A frame is sent with only error detection parity bits added, as with normal ARQ. If however retransmission is requested, a set of additional parity bits are sent rather than a copy of the frame (Figure 5.d(ii)). The receiver is now able to correct some errors, but if there are still uncorrected errors a further retransmission is requested. This has the advantage over Type I hybrid ARQ, that the efficiency is never less than that of simple ARQ.

One of the earliest schemes of this type was described by Mandelbaum (1974), using incremental code redundancy. The message is encoded using an error correcting code, then the code is punctured by having some parity bits removed. The message is sent using the punctured code, and if a retransmission is requested, some of the removed parity bits are sent. The receiver is then able to correct some errors, but if necessary the process is continued until all the parity bits have been sent, or all the errors corrected.

Metzner (1979) suggested splitting the k message bits into small sub-blocks (say 4 bits), and using a half rate code to generate additional parity bits using for example an (8,4) Reed-Muller code. Initially the k message bits are sent, but in the event of a retransmission, the k parity bits are sent. The receiver may then attempt correction of the transmission errors using the (2k,k) code.

A Type II hybrid ARQ scheme using a half rate code was also used by Lin and Yu (1981), who suggested encoding using a long block length BCH code, (1023,523) shortened to (1000,500), which can correct 5 errors and simultaneously detect 105 errors. In the event of a retransmission, the receiver has two chances of retrieving the message, firstly, the message is recoverable from the k parity bits if no errors occurred on the second transmission, and secondly by using the error correcting capabilities of the code. Wang and Lin (1983) also used a shortened (1000,500) BCH code but with a separate error detection code; the (1000,500) code can correct 55 errors.

The advantage of Type II ARQ over Type I is that the additional forward error correcting code parity bits are sent only when a retransmission is requested, hence at low error rates there is little overhead due to the code rate. At high error rates, Type II ARQ should again perform well, as the error correcting capability is very high.

There will be a range of error rates for which Type I will perform well, as the disadvantage of Type II is that there has to be at least one retransmission before the receiver is able to perform any

error correction. A combination of Types I and II should prove effective under both high and low error conditions, as a high rate forward error correcting code can be used to correct a small number of errors and hence avoid the need for retransmission, but if channel conditions deteriorate parity retransmission may be used.

5.3.3 Transmission efficiency of some hybrid ARQ schemes.

Three hybrid ARQ schemes will be compared with the basic Go Back N ARQ protocol. These are:-

(i) A simple Type I hybrid ARQ, employing a fixed rate linear block code.

(ii) A simple Type II hybrid ARQ, using parity retransmission based on a half rate linear block code.

(iii) A combination of Types I and II, using a high rate forward error correcting code in Type I mode, and a half rate code in Type II mode.

The transmission efficiencies of the three schemes are given below, the equations are derived in Appendix C:-

(i) Type I hybrid, using a fixed rate (n,k,t) linear block code, as

discussed above:-

$$\text{Efficiency} = \frac{k}{n}\ \frac{(k-h)}{k}\ \frac{1}{T'}$$

T' depends on the uncorrected error probability for the (n,k,t) forward error correcting code, i.e. the probability of more than t errors occurring within a block of length n bits .. P(m>t,n) (Section 4.3.2).

hence $$T' = 1 + \frac{N.\ P(m>t,n)}{(1 - P(m>t,n)}$$

(ii) Type II, with parity retransmission:-

$$\text{Efficiency} = \frac{(k-h)}{k}\ \frac{1}{T'}$$

T', the expected number of transmissions required to send a frame successfully, depends on two error probabilities. P_E is the probability that the first transmission will be corrupted P(m>0,k), whilst $P(m>t_2,2k)$, the probability that subsequent transmissions will fail, is lower due to the error correcting capability of the $(2k,k,t_2)$ code.

$$T' = 1 + \frac{N\ P_E}{(1 - P(m>t_2,2k)}$$

(iii) Combined Type I and II, with a high rate forward error correcting code, and parity retransmission:-

$$\text{Efficiency} = \frac{k}{n} \quad \frac{(k-h)}{k} \quad \frac{1}{T'}$$

The expression for T is as given in (ii) above, however the initial probability of error P_E will be replaced by $P(m>t_1,n)$ due to the effects of the (n,k,t_1) forward error correcting code element.

$$T' = 1 + \frac{N.\ P(m>t_1,n)}{(1 - P(m>t_2,2n))}$$

The choice of code rate should depend on the prevailing channel conditions. In practice however, knowledge of the bit error rate in the forward channel can only be determined by observing the number of retransmission requests, and hence acquired only after some significant delay.

Figure 5.e shows the relationship between efficiency and bit error rate for Type I hybrid ARQ, for code rates of 0.98, 0.9, 0.8, and 0.6. For comparison the theoretical channel capacity (from Section 4.4) is also shown. Clearly for low error rates the code rate should be as high as possible, however at high error rates some intermediate code rate will give optimum performance.

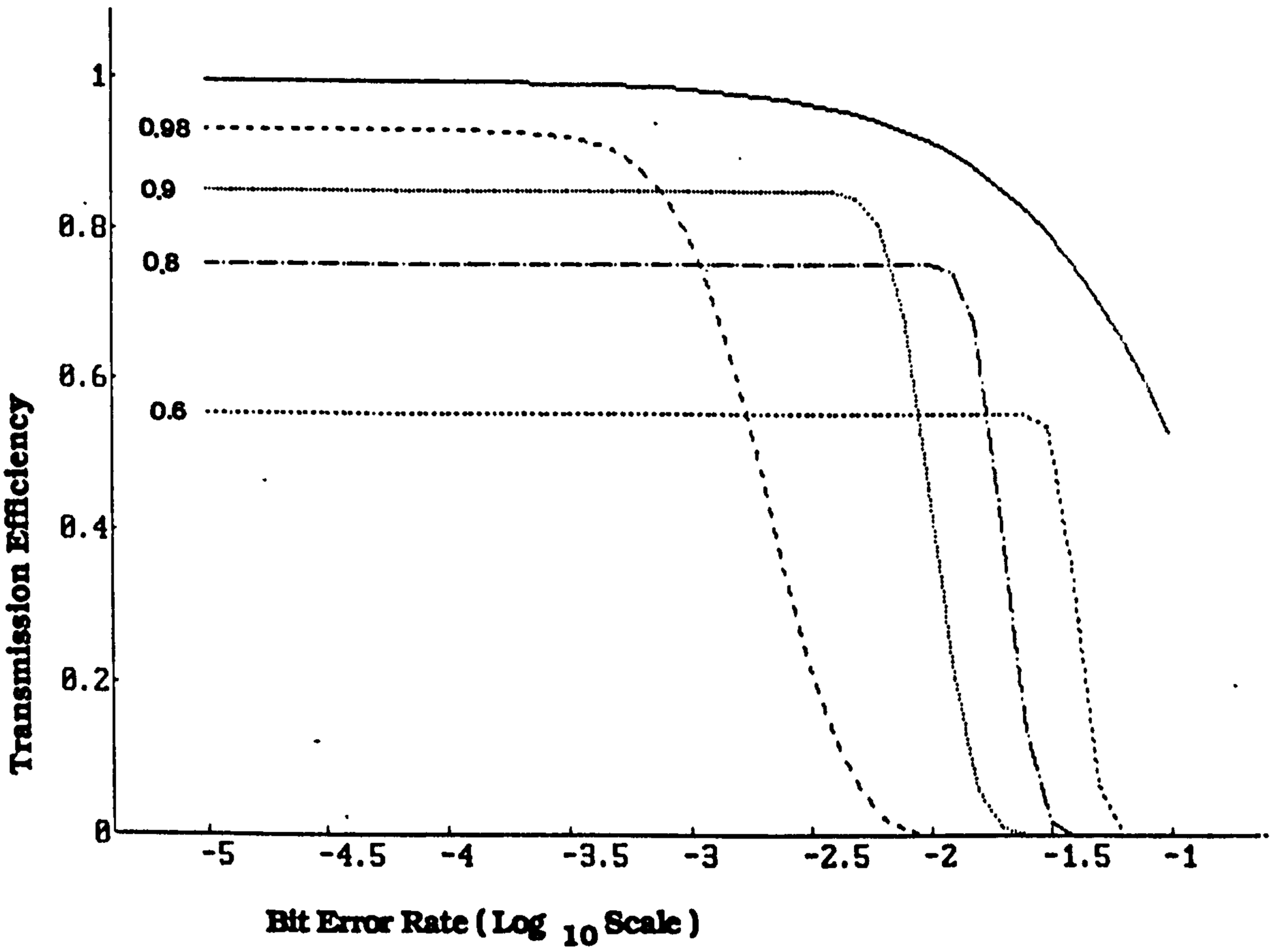

Figure 5.e Transmission Efficiency of Type I Hybrid ARQ for the binary symmetric channel, with a range of FEC code rates.

A rate 0.85 BCH code will be used for scheme (i), this corresponds for example, to the (1023,873,15) and (511,439,8) BCH codes. The performance of Type (i) hybrid ARQ for other code rates will be explored more fully in Sections 5.6 and 5.7. A higher code rate, 0.95, will be used for scheme (iii).

5.4 The Effects of Channel Error Distribution and Delay on Transmission Efficiency.

In Section 4.3, three different channel models were proposed for the evaluation of error control schemes. These were the binary symmetric channel, the run length encoded data published in Lewis and Cox (1966) and a model for the block error rate derived from data from the Bell telephone network survey of 1969-70 given in Balcovic (1971). The models will be referred to as the BSC, Lewis and Cox, and Bell models.

The performance of four ARQ schemes will be compared, Go Back N ARQ, and the three hybrid GBN ARQ schemes described in Section 5.3.3. The channel models used for comparison of the ARQ schemes outlined above are (from Section 4.3.2) :-

(i) The binary symmetric channel.

- BSC model

$$P(>m,n) = 1 - \sum_{i=0}^{m} {}_nC_m \; p_t^i \; (1-p_t)^{(n-i)}$$

(ii) Data given by Lewis and Cox (1966). (Bit error rate 0.0007)

This recorded gap length data will be used to calculate P(>m,n) directly for each result required.

(iii) Model based on A.T.&.T. results from 1969/70 survey, reported by Balcovic et al (1971). (bit error rate ≈ 10^{-6})

- Bell model:-

$$P(\geq m,n) = n^a.m^b.c$$

with calculated values of a=0.87, b=-1.66, c=1/98700

Figures 5.f(i) and 5.f(ii) show the efficiency of the four ARQ schemes when applied to a binary symmetric channel, with bit error rate ranging from 0.00001 to 0.1. The frame length is 1000 bits, and the transmission delay zero in Figure 5.f(i), and 1000 bit intervals in Figure 5.f(ii).

At low error rates, ARQ performs well, but efficiency falls rapidly with increasing error rate. Hybrid scheme (i) has an efficiency slightly less than the code rate, which remains constant to a high error rate, indicating that the error correcting code is correcting almost all errors at low to medium error rates, and hence avoiding the need for retransmission.

Hybrid scheme (ii), parity retransmission, has a similar performance to ARQ at low error rates, but the half rate (2n,n) error correcting code provides reasonable throughput at very high error rates (up to a BER of 0.05).

The combined Type I/II hybrid provides good performance at all error rates. At error rates up to 0.001, the throughput is approximately 90 percent, whilst the parity retransmission provides reasonable efficiency up to an error rate of 0.03.

There is a larger difference in efficiency between the Type II and combined Type I/II scheme than might be expected. At error rates higher than 0.001, the Type II scheme has an efficiency of approximately 0.47, whilst the Type I/II scheme has an efficiency of approximately 0.3, about 46 percent less than the Type II scheme. One would expect the difference to be only five percent, due to the rate of the additional code, however the effect is due to the additional encoding/decoding delay allowance of one block length.

The effects of increasing the delay to 1000 bit intervals (shown in Figure 5.f(ii)) is to reduce the efficiency of all schemes. At low error rates the simple ARQ and the Type II hybrid ARQ show greatest signs of degradation, due to the increased number of blocks that must be retransmitted. The Type I and combined Type I/II are less affected, as the forward error correction reduces the incidence of retransmission requests. At higher error rates, the Type I/II scheme shows reduced efficiency, however the Type I scheme is only slightly affected.

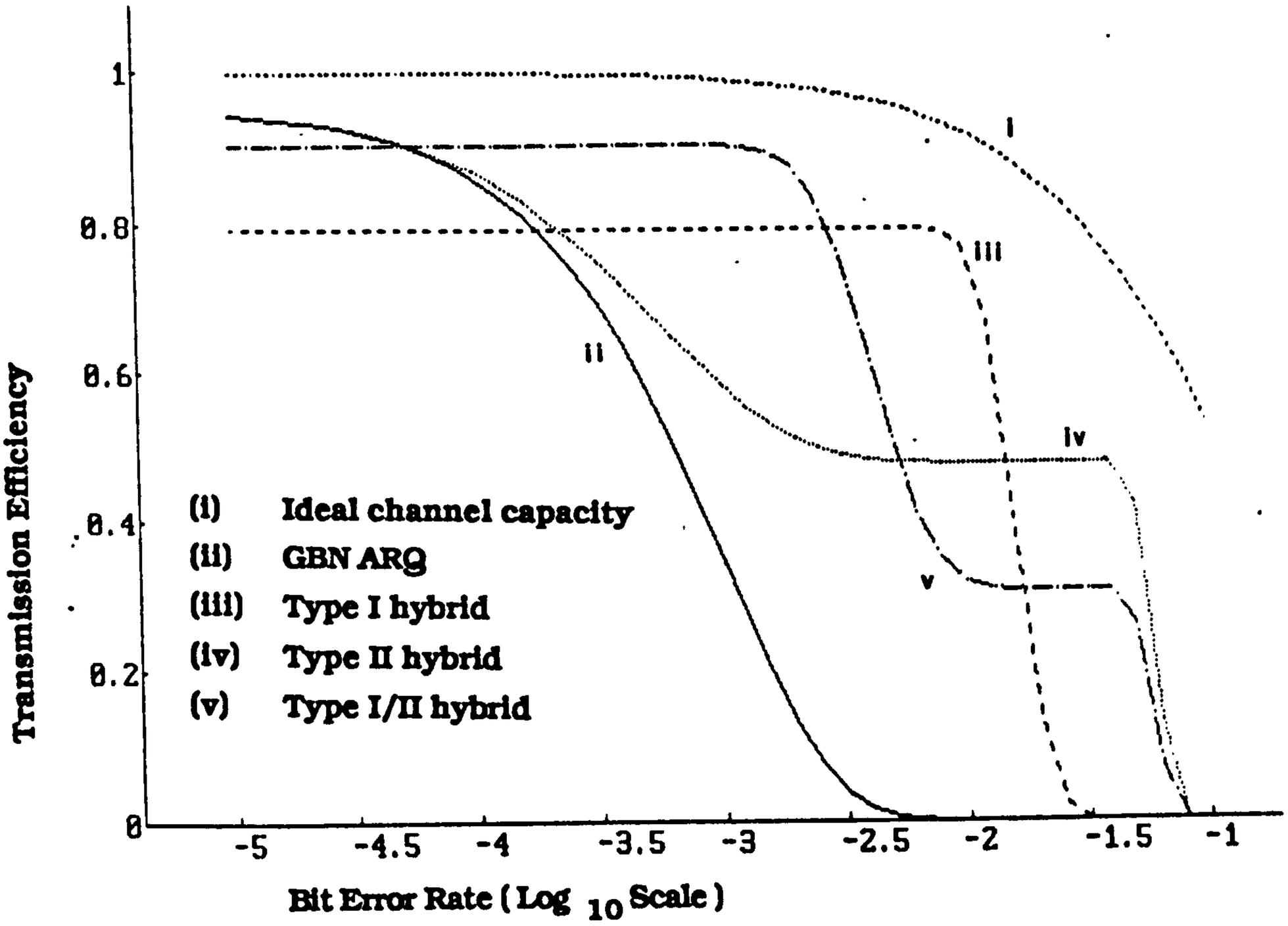

Figure 5.f(i) Transmission Efficiency of the four ARQ schemes, for the binary symmetric channel, with zero delay.

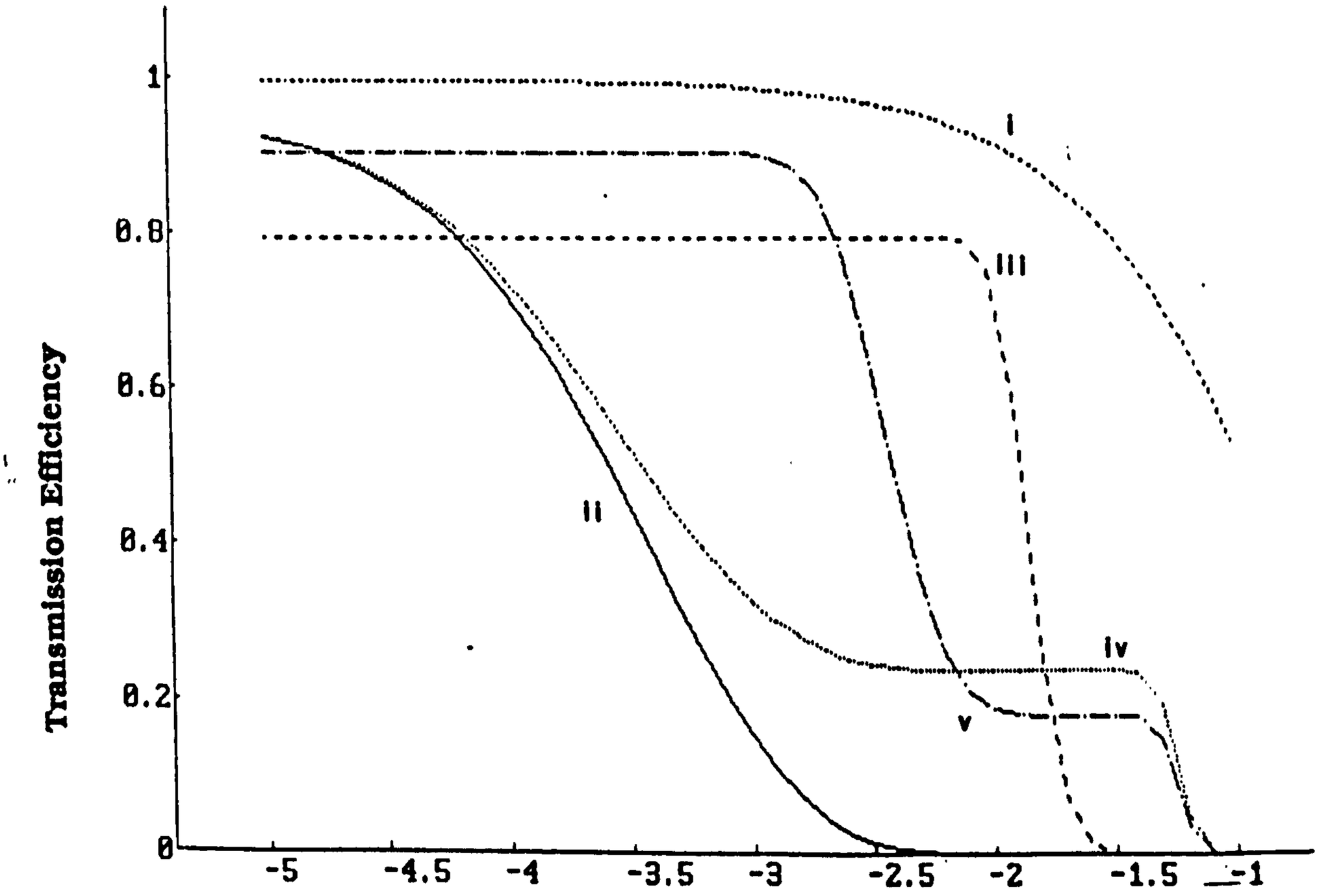

Figure 5.f(ii) Transmission Efficiency of the four ARQ schemes, for the binary symmetric channel, with a 1000 bit channel delay.

Figures 5.g(i)-(iv) and 5.h(i)-(iv) illustrate the effectiveness of the four ARQ schemes on four different channel models. The first set of graphs show the relationship between efficiency and channel delay for each ARQ scheme, whilst the second set of graphs compare the performance of the different ARQ schemes for each channel model.

In Section 4.2.1, channel delay was briefly discussed, and typical values given. A PSTN channel typically has a delay of 5 to 30 milliseconds, whilst a satellite channel may have a delay of 300 milliseconds or more. In terms of bits, the expected end to end delays are:-

(i) At 1200 bits per second, from 6 to 36 bits on the PSTN, and approximately 360 bits for a satellite channel.

(ii) At 19200 bits per second, from 90 to 580 bits on the PSTN, and approximately 5800 bits for a satellite channel.

A range of end to end delay of 0 to 5000 bits is used in Figures 5.g and 5.h., with an assumed frame length of 1000 bits.

Go Back N ARQ performs fairly poorly on channels with extensive delay, as shown if Figure 5.g(i). The worst performance occurs on the high error rate binary symmetric channel. On the low error rate BSC, and Lewis and Cox channel, ARQ performs tolerably at delays of less than 2000 bits, whilst on the low error rate Bell model, very little degradation is noticeable. There is a significant

difference in performance between the high error rate BSC and the Lewis and Cox channel results, despite the similarity in bit error rate, 0.001 for the BSC and 0.0007 for the Lewis and Cox data. This is due to the clustering of errors on the recorded channel data, discussed in Section 4.3.2.

Figure 5.g(ii) shows the relationship between efficiency and delay for the Type I hybrid scheme. This is relatively insensitive to delay; a slight degradation may be observed on the result for the Lewis and Cox channel data. This is again due to the clustering of errors, which results in the forward error correcting code failing to correct a small proportion of errored blocks.

The Type II hybrid scheme does not perform markedly better than simple ARQ, with the exception of slightly improved performance on the high error rate BSC, as shown in Figure 5.g(iii).

Figure 5.g(iv) shows the result for the combined Type I/II scheme, which shows some interesting features. The efficiency of the scheme is relatively insensitive to delay, with the exception of the result obtained for the Lewis and Cox data. This illustrates that the high rate error correcting code is able to cope well on the binary symmetric channels, but is unable to correct a fair proportion of errors on the recorded channel data.

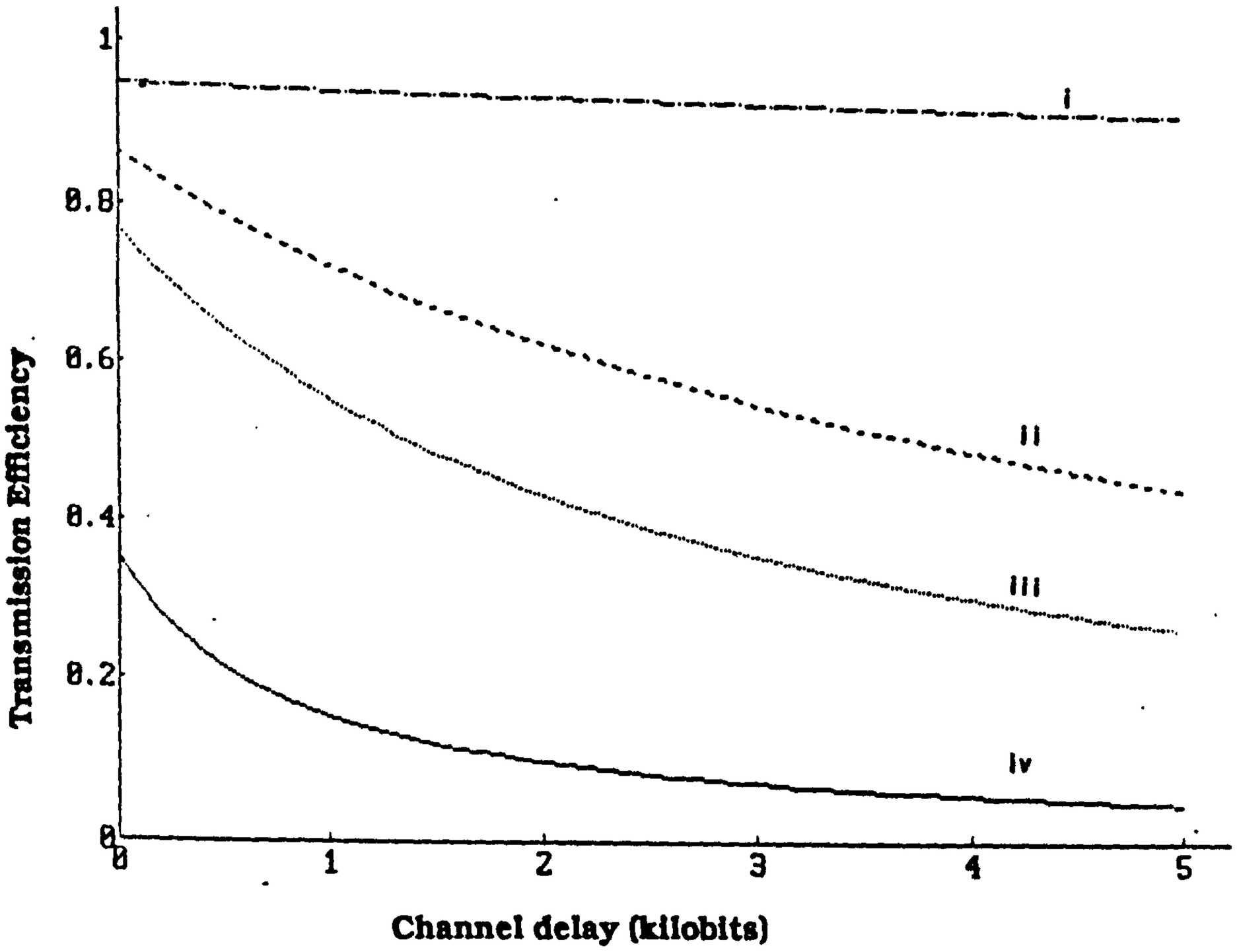

Figure 5.g(i) The Relationship between Transmission Efficiency and Channel Delay for Go Back N ARQ, with four different channels.

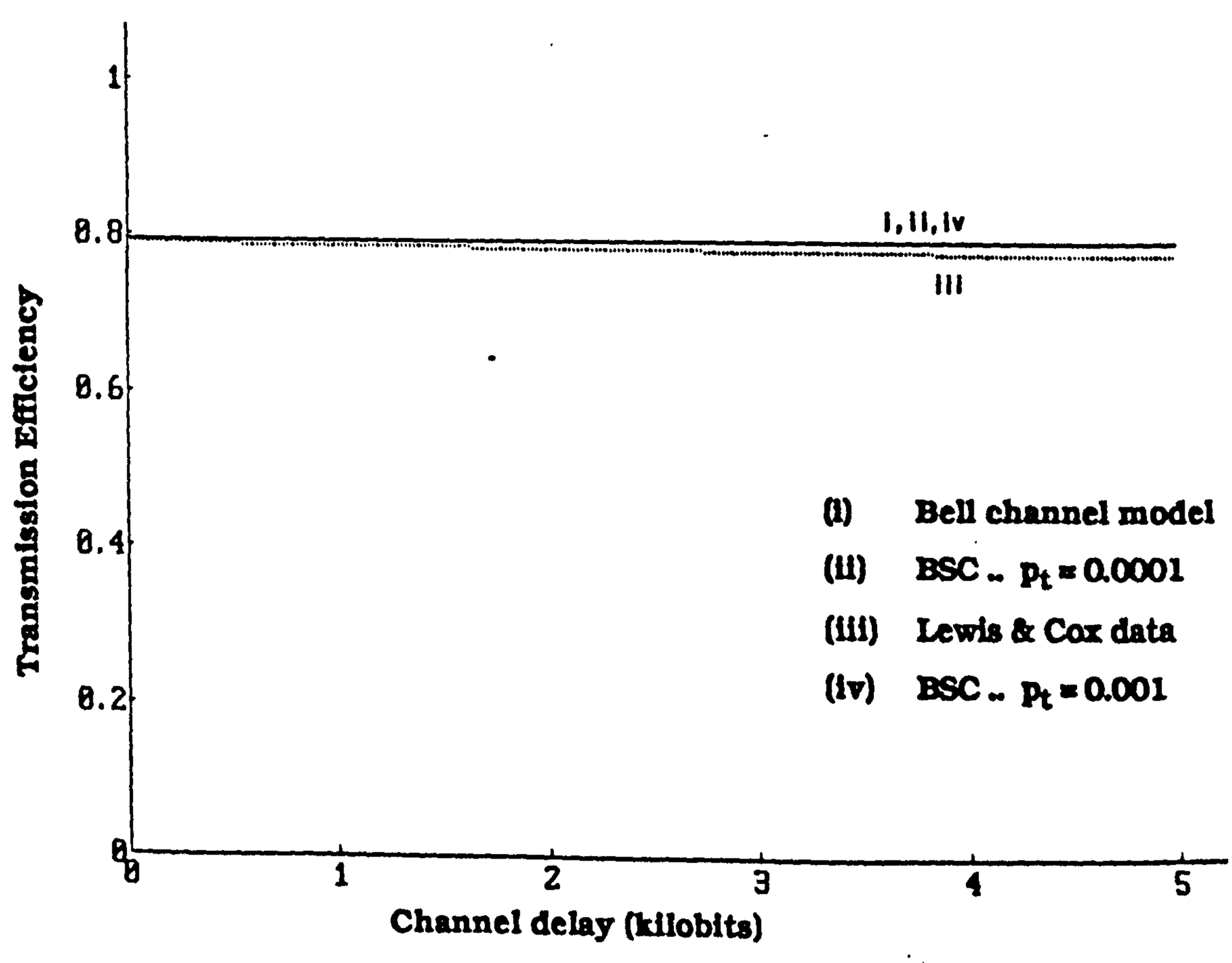

Figure 5.g(ii) The Relationship between Transmission Efficiency and Channel Delay for Type I Hybrid ARQ, with four different channels

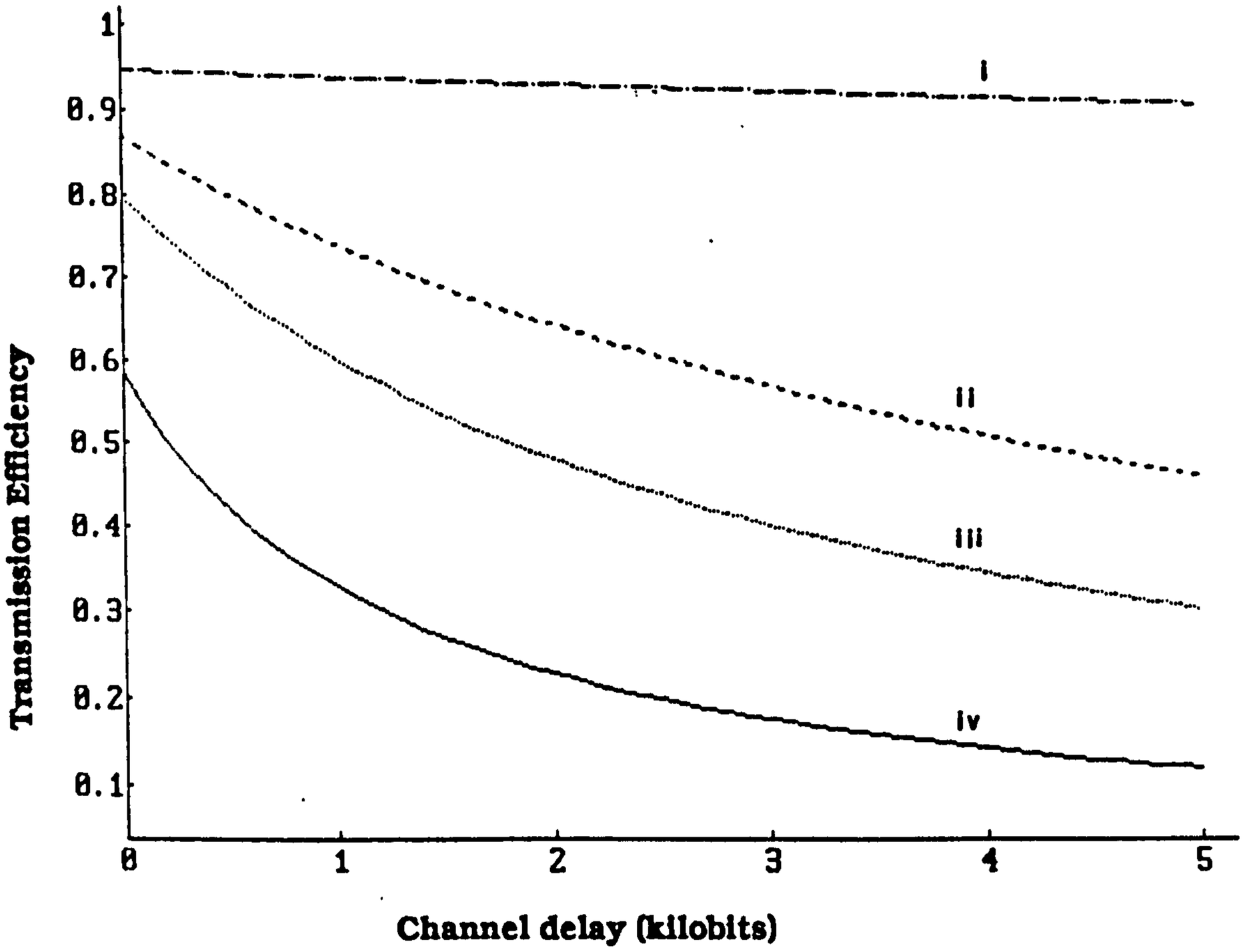

Figure 5.g(iii) The Relationship between Transmission Efficiency and Channel Delay for Type II Hybrid ARQ, with four different channels.

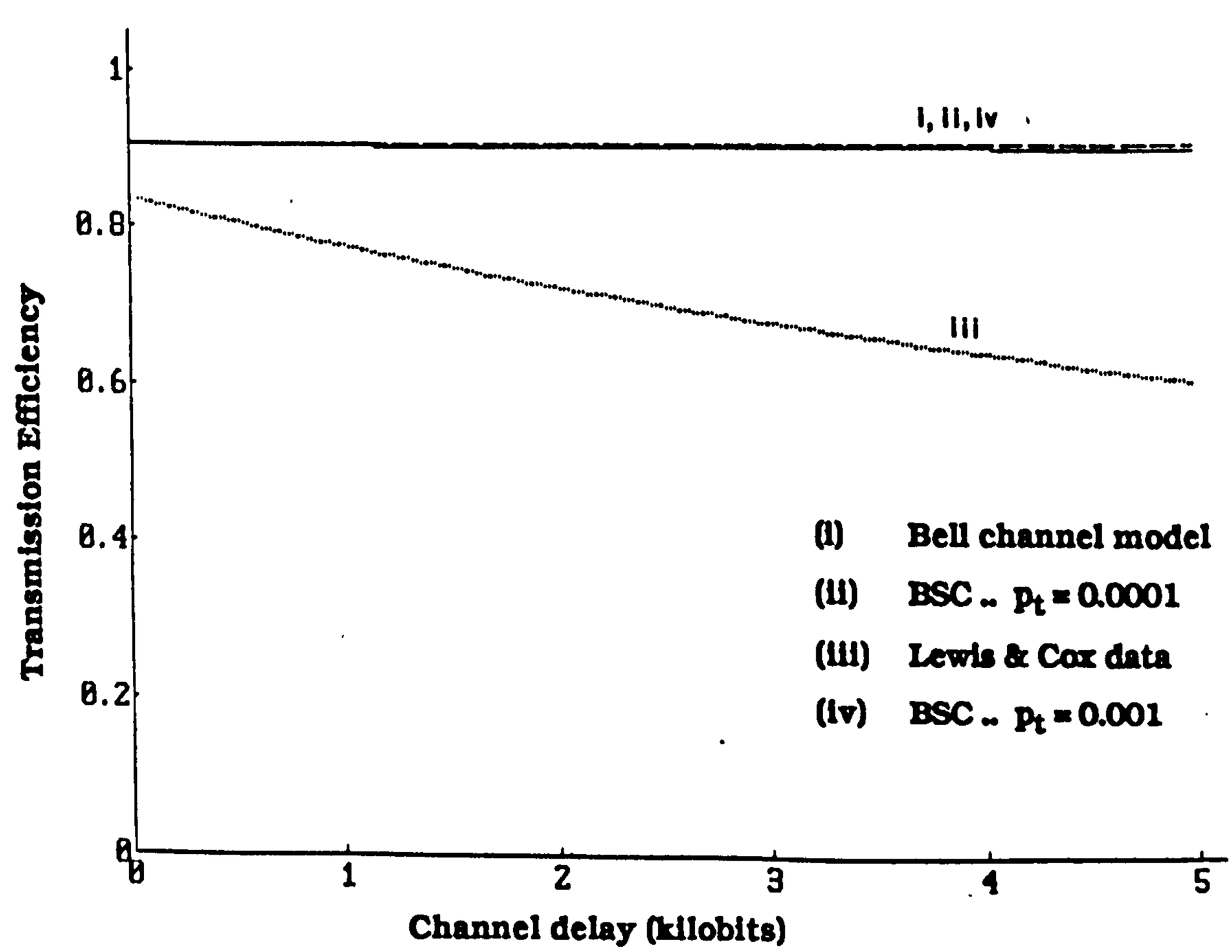

Figure 5.g(iv) The Relationship between Transmission Efficiency and Channel Delay for the Mixed Type I/II Hybrid ARQ Scheme

On the high error rate BSC, the two schemes employing forward error correction are considerably more effective than those relying solely on retransmission, as shown in Figure 5.h(i). This is also apparent on the low error rate BSC, although only for large delays. Figure 5.h(ii) shows that under these conditions, the ARQ and Type II hybrid perform well for delays less than 1000 bits, but the Type I/II scheme performs best for all delays.

A slightly different picture emerges when the results for the Lewis and Cox channel data are examined (Figure 5.h(iii)). The performance of the Type I/II scheme drops markedly, but is still the most efficient for delays of less than 1000 bits. The Type I scheme is fairly efficient for the whole range of channel delay.

The low error rate Bell model provides a good environment for simple ARQ and the Type II hybrid, whilst the loss in efficiency due to the overhead of the forward error correction code limits the performance attained by the Type I and Type I/II schemes.

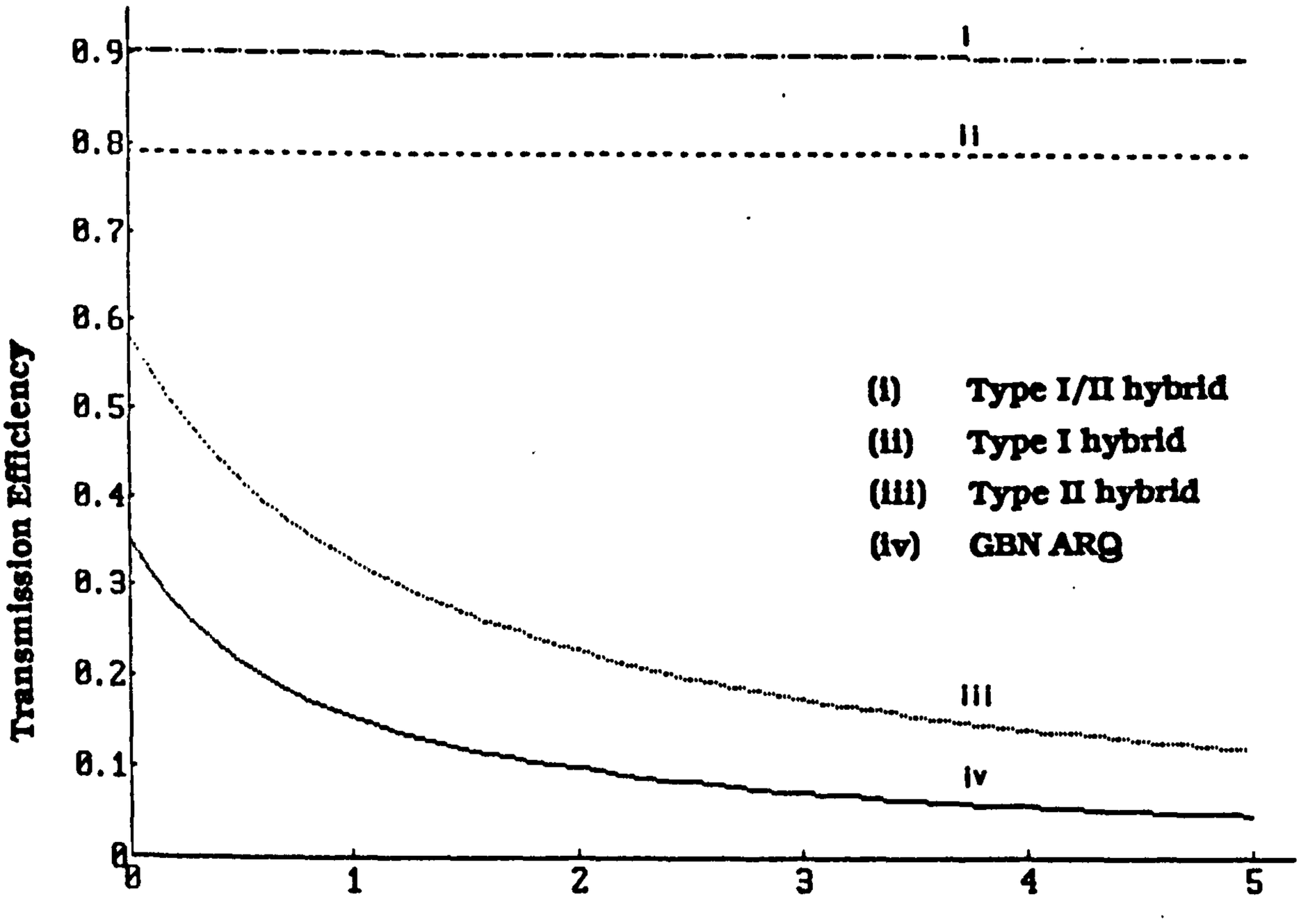

Figure 5.h(i) The Relationship between Transmission Efficiency and Channel Delay for the four ARQ schemes on the binary symmetric channel with BER 0.001.

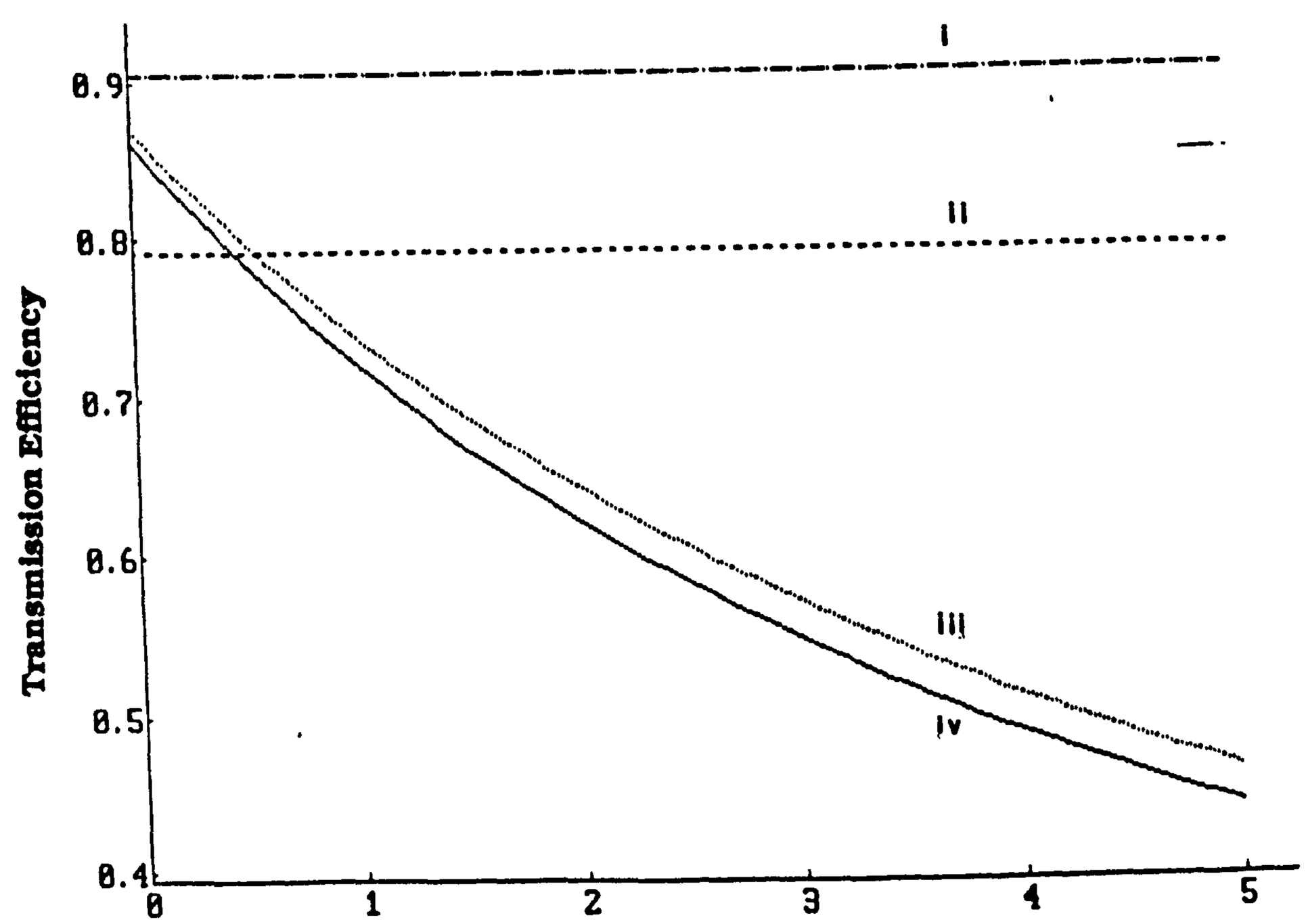

Figure 5.h(ii) The Relationship between Transmission Efficiency and Channel Delay for the four ARQ schemes on the binary symmetric channel with BER 0.0001.

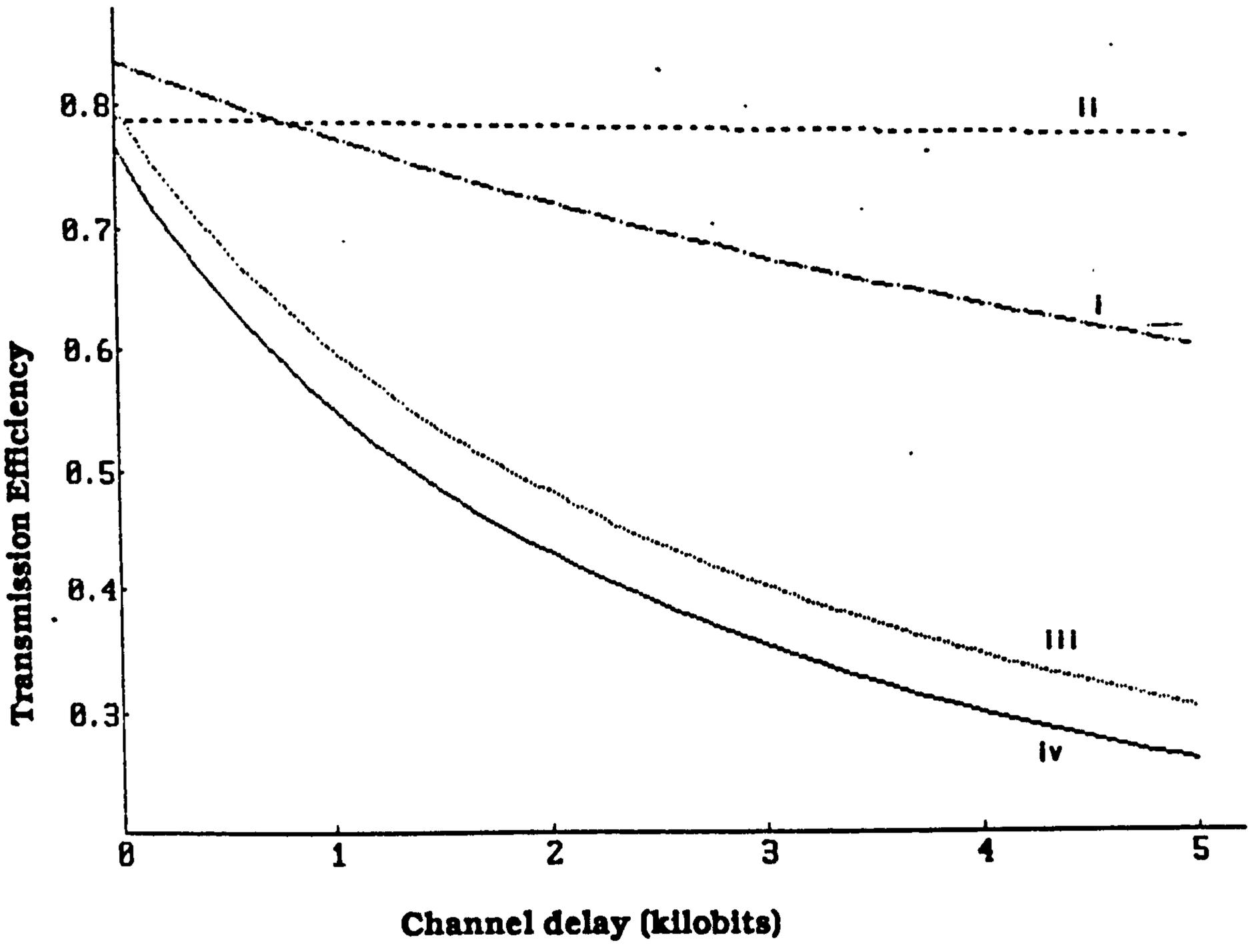

Figure 5.h(iii) The Relationship between Transmission Efficiency and Channel Delay for the four ARQ schemes on the Lewis and Cox error data.

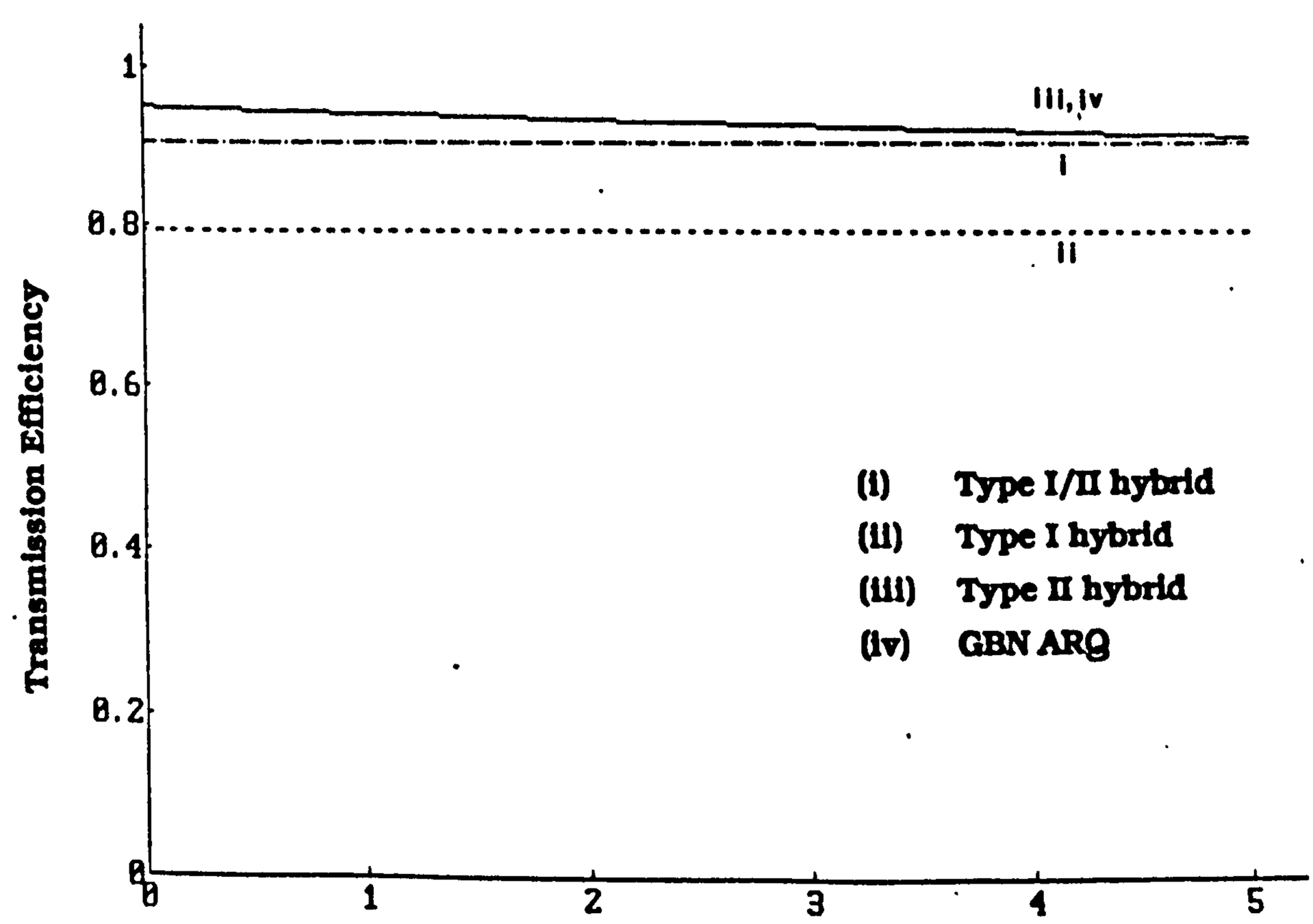

Figure 5.h(iv) The Relationship between Transmission Efficiency and Channel Delay for the four ARQ schemes on the Bell channel model.

5.5 The Relationship between Frame Length and Efficiency.

The efficiency of an ARQ system depends heavily on the choice of frame length. Long frames are susceptible to errors, whilst short frames are inefficient due to the proportionally large header. For some given channel conditions, there exists an optimum frame length. It will be assumed initially that channel conditions are stable, and that the use of a fixed frame length is practical.

The use of forward error correction should substantially alter the frame length/ efficiency relationship, as longer frames will be less affected by channel errors. In this section, the performance of the Go Back N, and Type I hybrid ARQ schemes is considered for a range of frame lengths of 100 to 2000 bits, on the four channel models used in the preceding section. This is followed by some discussion of the choice of optimum or maximum frame length.

Figures 5.j and 5.k show the variation of efficiency with frame length for the two ARQ schemes and four channel models, for transmission delays of 0 and 1000 bits.

Go Back N ARQ is fairly efficient for short block lengths, as shown on Figure 5.j(i). Distinct maxima are present for all channel types except the low error rate Bell model. For the high error rate BSC the maximum efficiency is 0.62, at a block length of approximately 250 bits. The Lewis and Cox model provides higher efficiency, 0.78 at a block length of 600 bits.

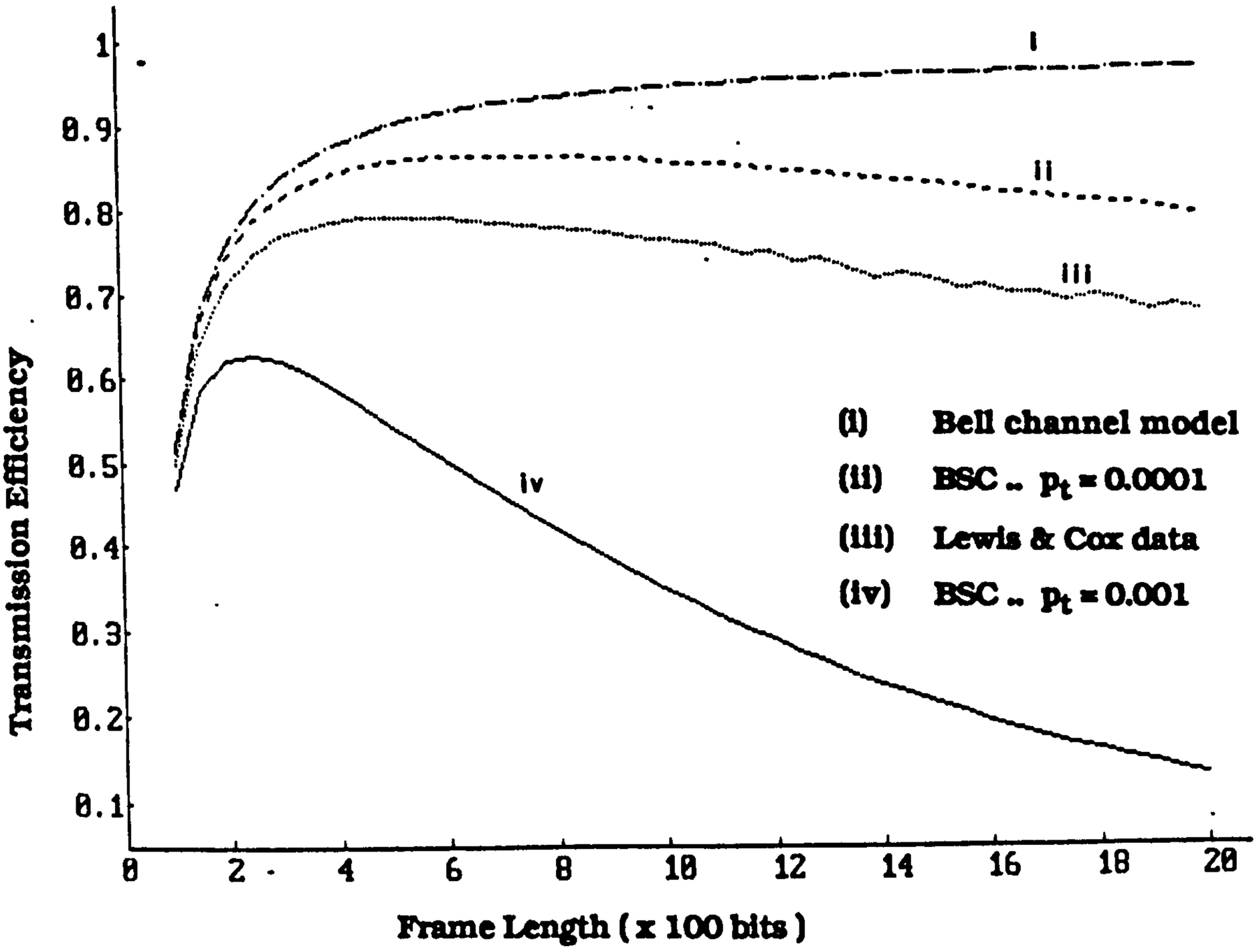

Figure 5.j(i) The Relationship between Transmission Efficiency and Frame Length for Go Back N ARQ , for zero end to end delay.

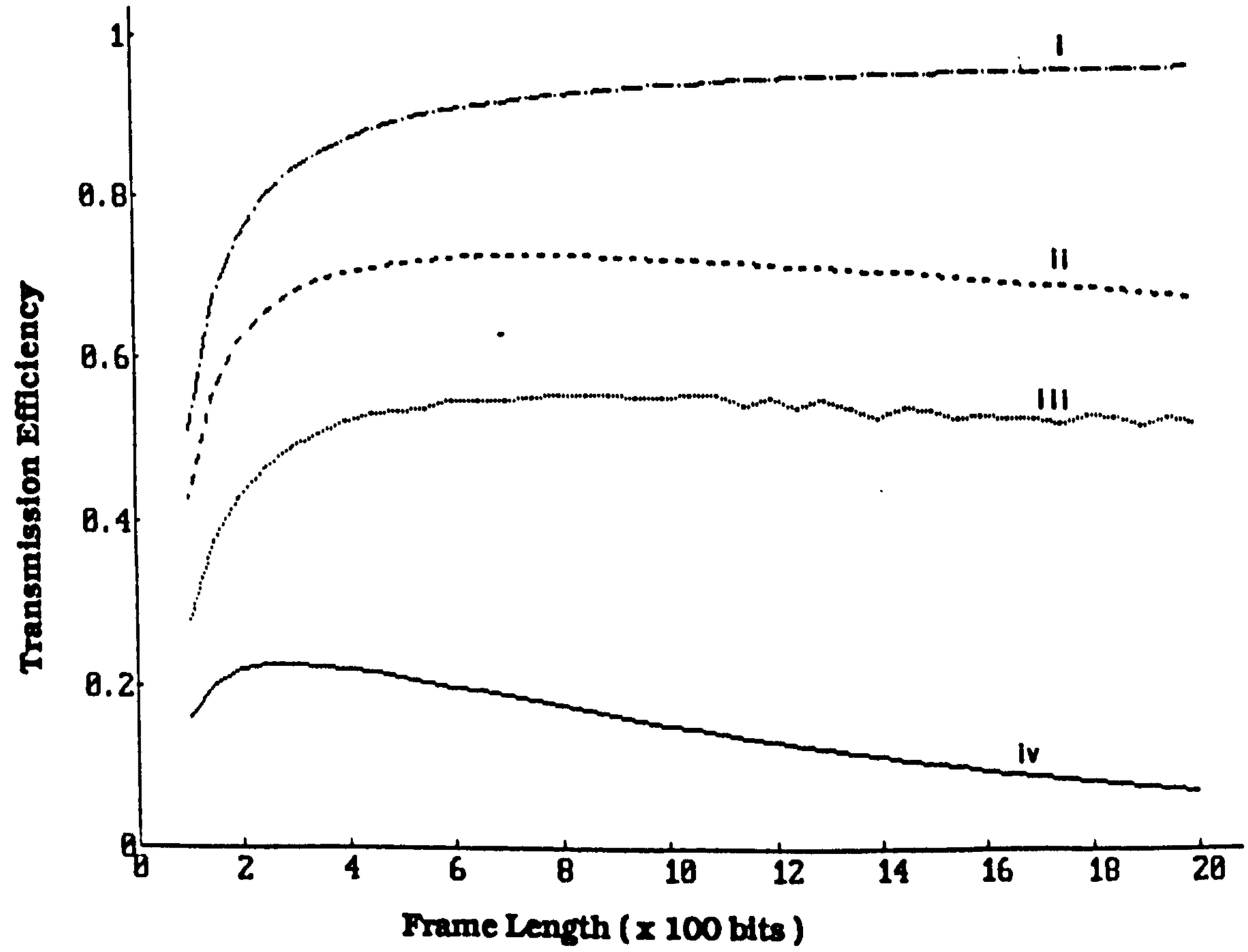

Figure 5.j(ii) The Relationship between Transmission Efficiency and Frame Length for Go Back N ARQ , with an end to end delay of 1000 bits.

The effect of increasing the channel delay is shown in Figure 5.j(ii). The performance of ARQ on the Lewis and Cox, and low error rate BSC is degraded by 10-20 percent, and the optimum block length increased noticeably. The efficiency of ARQ on the high error rate BSC has dropped considerably, to just over 0.2, whilst the optimum block length has increased slightly to 300 bits.

Some slight irregularity may be observed on the curves for the Lewis and Cox model, as the model is based on recorded channel data rather than a mathematical model.

In Figure 5.k(i) the efficiency of the Type I hybrid ARQ may be seen to increase monotonically for the range of block lengths considered. There is little difference in performance for any of the channel models.

The effect of increasing the channel delay to 1000 bits is shown in Figure 5.k(ii). The results are very similar to Figure 5.k(i), which illustrates the high tolerance of the scheme to delay. Some slight degradation is noticeable on the curve for the Lewis and Cox model, the explanation for this was given in the preceding section.

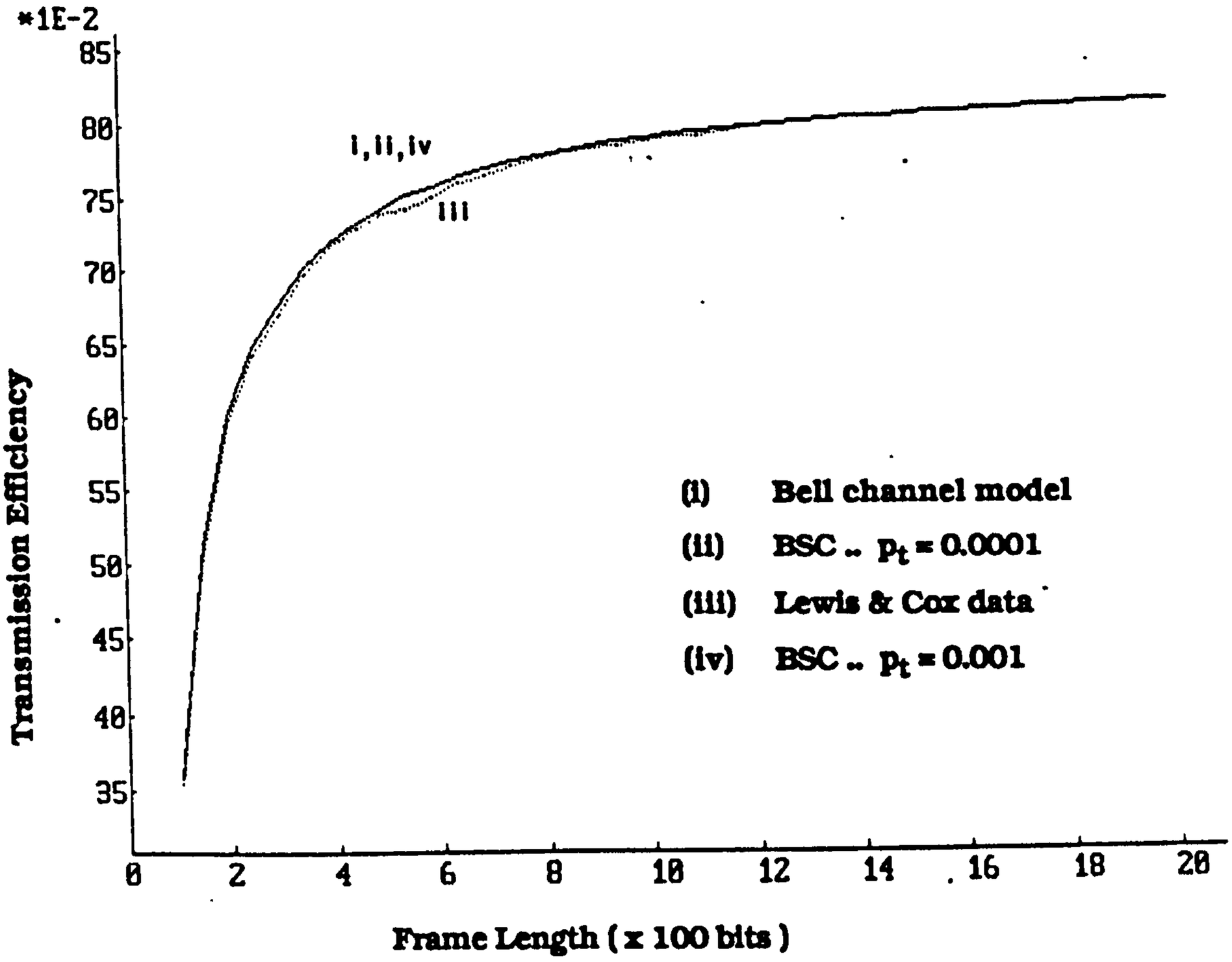

Figure 5.k(i) The Relationship between Transmission Efficiency and Frame Length for Type I hybrid ARQ, with zero channel delay .

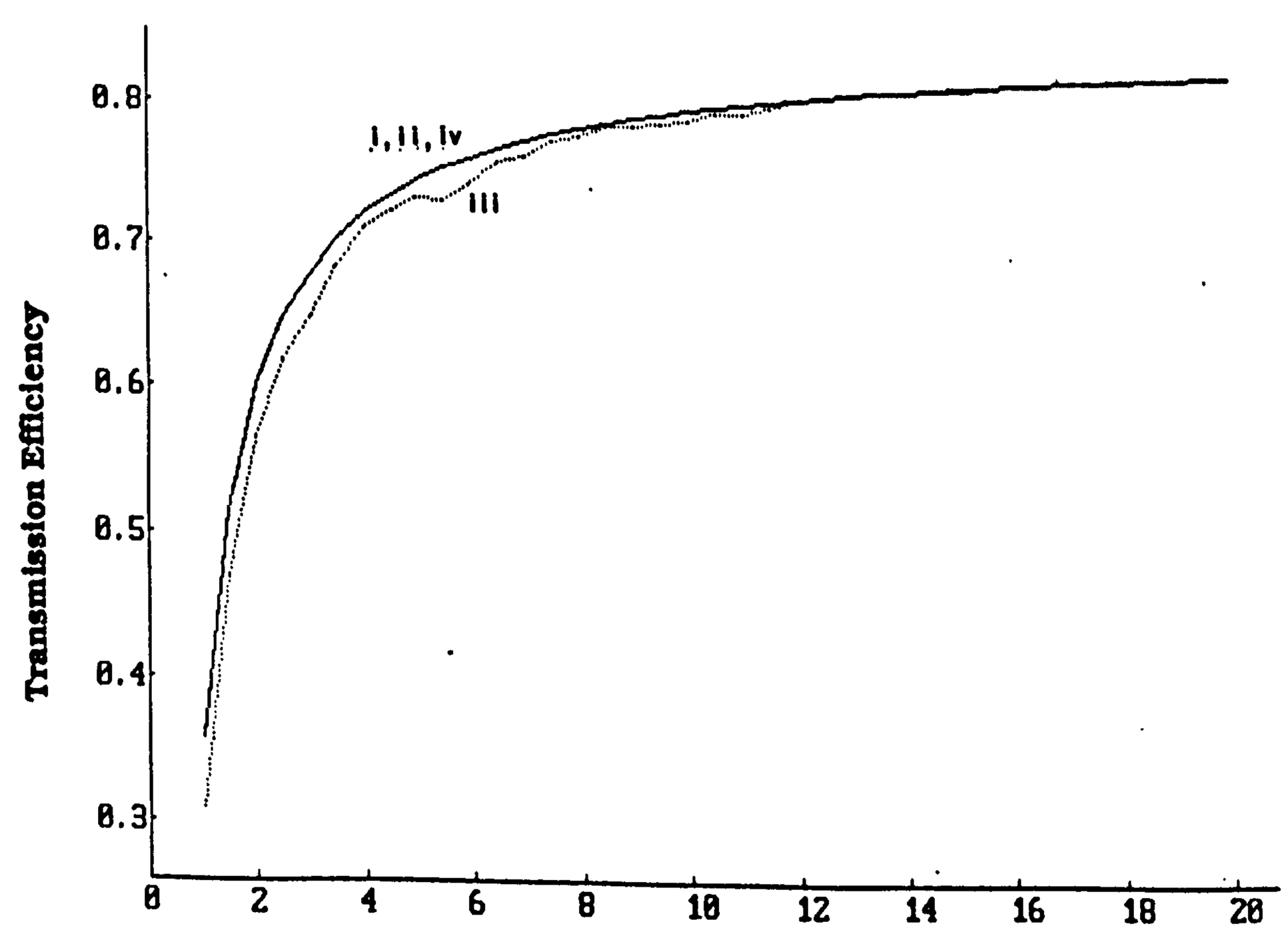

Figure 5.k(ii) The Relationship between Transmission Efficiency and Frame Length for Type I hybrid ARQ, with a 1000 bit end to end delay .

Expressions for optimum frame length are derived in Appendix D for Stop and Wait, Selective Repeat, and Go Back N ARQ. The equation for the efficiency is differentiated, and the zero crossing point of the derivative (corresponding to the maxima) found. The conditions under which this approach is valid are discussed by Chu (1974) and Morris (1979).

For Go Back N, the optimum frame length k satisfies:-

$$(1-P_E)(h + k.P_E.(N-1)) - (k-h).k.N.P_E'=0$$

where P_E is the probability P(m>0,k), and P_E' is the derivative of P_E with respect to k. Note particularly that P_E,P_E' and N are functions of k.

In practice, the choice of frame length does not depend solely upon channel conditions. For example in a data network, packets of data maintain their identity, and are effectively encapsulated within an ARQ frame. Alternatively, a communications system carrying data between a VDU operator and computer should not have to accept sufficient characters typed from the keyboard to fill an optimum frame before sending it. The optimum frame length should therefore be regarded as the maximum size for the given channel conditions, rather than a fixed value.

Many studies of data communications traffic have been

published. For example Fuchs and Jackson (1970) discuss the terminal-computer data traffic characteristics of three time shared computer systems. They found that the traffic occurred in bursts, generally conforming to a Gamma distribution, with mean burst length of 11 characters for user terminal to computer traffic, 41 characters for computer to terminal traffic.

The channel conditions are not in practice known precisely, thus the optimum frame size can only be roughly estimated. An adaptive scheme could be used, for example, to reduce the frame length when a retransmission is requested and increase it when no frames have been rejected for a set period of time.

This highlights one unfortunate aspect of ARQ, namely that once a frame has been transmitted its size is fixed. If during good conditions the frame size has increased to say 2000 bits, then the stored copies of the most recently transmitted frames are all 2000 bits in length. If the channel then becomes noisier, with an error rate of say 0.005, the transmitter will then retransmit 2000 bit frames, which have a very low probability of successful transmission (0.000044 for a binary symmetric channel). The system will effectively be locked up until the error rate improves.

This may be prevented by using, for example, the Type II hybrid ARQ scheme discussed above. Another possible solution, which does not involve the use of forward error correction involves a slight modification to the definition of ARQ. Instead of retaining the unacknowledged frames, the unframed data can be stored. If a retransmission is requested, the frames are reconstructed and may

be longer or shorter than the original ones. Thus the lockup situation described above would result in successively shorter frames being transmitted, the frame length would reduce until some throughput was obtained. This approach would be more suited to slowly time varying channels.

5.6 Selection of Code Rate for Type I Hybrid ARQ.

The rate of the code used for forward error correction in a hybrid ARQ system has a considerable influence on performance. In Figure 5.e, the efficiency of a Type I hybrid ARQ scheme was shown to depend heavily on code rate, for the binary symmetric channel. This section considers the choice of optimum code rate for the four different channel models used in Sections 5.4 and 5.5.

In Figure 5.m, the efficiency of a Type I hybrid ARQ scheme is shown for a range of code rates from 3/4 to 1. The codes are all known (1023,k,t) BCH codes, otherwise the assumptions are as defined in Sections 5.1-5.4, with zero delay. For the low error rate Bell model, the optimum rate is 1, i.e. no forward error correction; whilst for the worst case, the Lewis and Cox data, the optimum rate is 0.92.

The comment of Rocher and Pickholtz mentioned in Section 5.3.1 above, may be called into question as a result of the observations made on Figure 5.m. They state that, in the context of data transmission over voice grade telephone lines, only a few errors need to be corrected by the forward error correcting code within a hybrid ARQ system. This may well appear to be the case if the binary symmetric channel is used a basis for comparing codes, however the bursty nature of telephone channel errors leads to an increased proportion of error free frames, and an increased error density within errored frames.

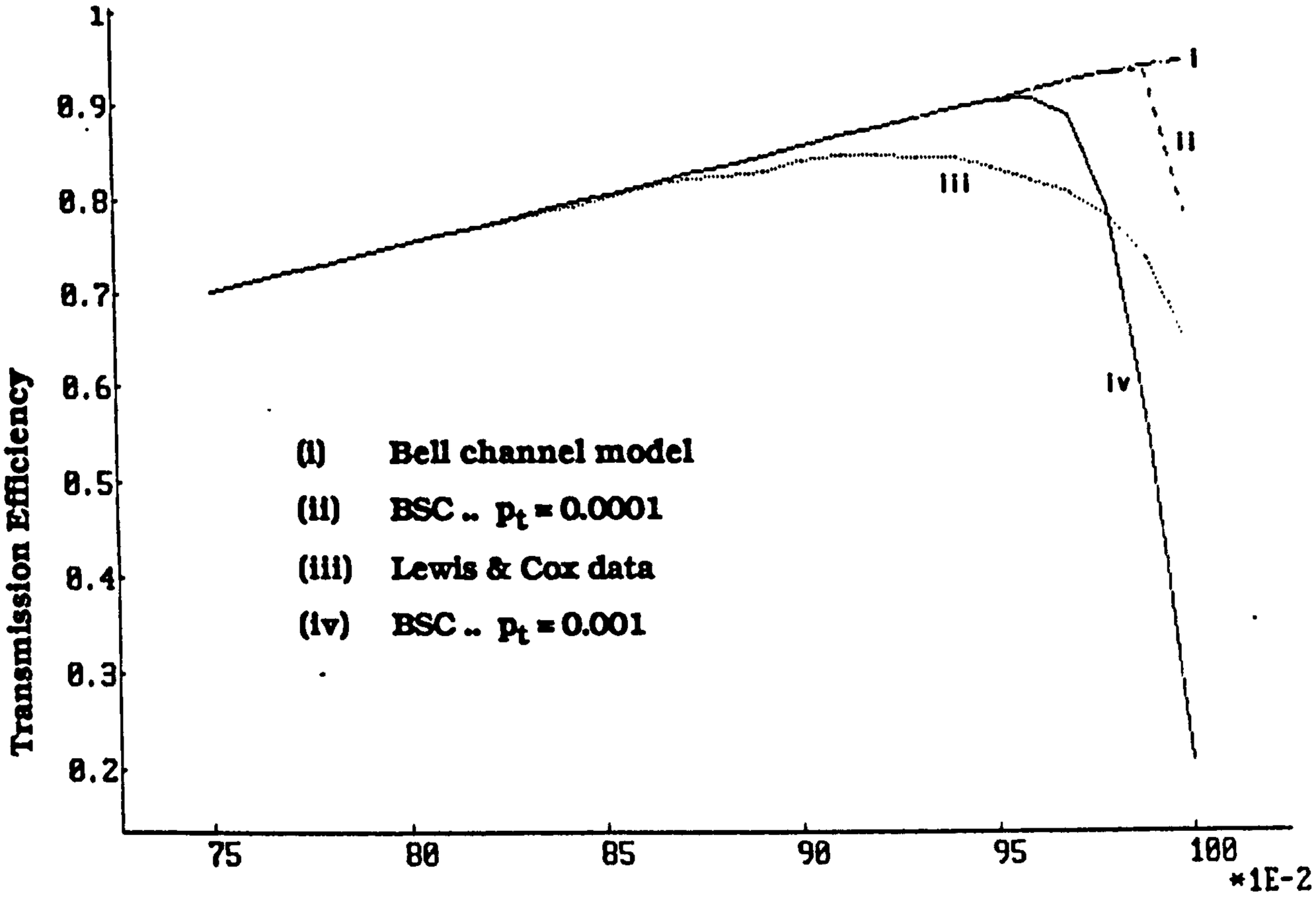

Figure 5.m The Transmission Efficiency of Type I Hybrid ARQ, for FEC code rates from 0.75 to 1.00, for the four different channel models .

In practice the channel error distribution is usually not known in advance, and may well be time varying. The code rate may be fixed, or could be made adaptive. If an adaptive scheme is to be effective however, the rate of change of error rate would need to be slow, due to the delay before the transmitter detects a change in channel conditions (from an increase in the number of retransmission requests).

5.7 Adaptive Selection of Code Rate.

The distribution of errors with time for the Lewis and Cox data is shown in Figure 5.n. The graph shows the time sequence of the number of errors per 1000 bit block, and clearly shows the clustering effects discussed above. An adaptive code redundancy selection scheme seems likely to achieve some success, at least for low values of channel delay.

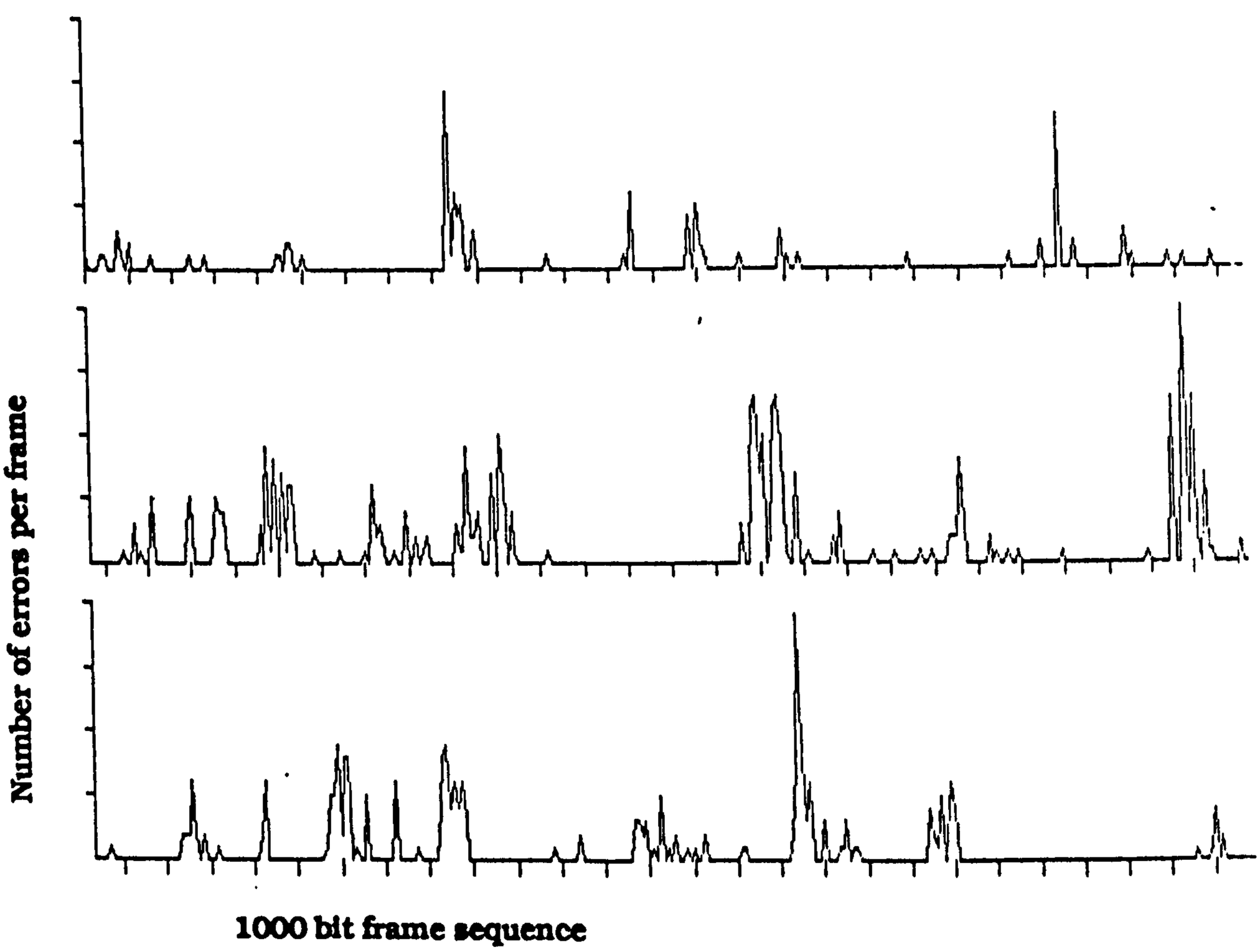

Figure 5.n Number of errors per 1000 bit block , as a time series, from the Lewis and Cox channel error data.

Variable redundancy codes have been investigated by Weng and Sollman (1967), and others. Their application to ARQ systems

was discussed by Farrell (1969) and Goodman and Farrell (1975). Farrell suggested that sets of optimum (i.e. maximum code rate for a given d_{min}) short block codes should be used, and the code selected from amongst these. The performance of this and a number of other error control coding schemes was given for a white noise and optical channel.

Goodman and Farrell studied the performance (in terms of undetected error rate and efficiency) of variable redundancy ARQ schemes over HF radio channels. A number of linear block codes with the same codeword length were used, the choice of code rate being made on the basis of the block error rate measured at the receiver. A PSTN line was used to provide information feedback to the transmitter.

They conducted a number of tests, involving the use of sets of up to eight different code rates, and found that it was possible in many cases to achieve better performance with only two code rates. The use of a simple channel condition monitor is suggested, of the form "if two successive blocks with errors occur, change to the more powerful code".

5.7.1 An adaptive hybrid ARQ scheme.

To investigate the effectiveness of adaptive schemes it is necessary to perform simulation rather than analysis. A simple Go Back N model may be constructed, and the Lewis and Cox data used to provide error information directly. The GBN model shown below includes two code rates (high and low rate); the transmitter switches to the low rate code whenever a *reject* or *errored* acknowledgement is received, and stays in that state for a period of time. The *errored* acknowledgement is needed to indicate that, although a retransmission is not being requested, errors are still present on the channel. The delay before switching back to the high rate code, the *sustain factor*, is considered below.

The potential advantage of the technique is not that it improves the probability of the retransmitted block being accepted as with the scheme suggested by Mandelbaum (ibid), although it does this. Rather the advantage is that the retransmitted block and a number of following blocks are encoded using the lower rate code, hence the error correcting code is still in use after retransmission has been completed. Thus if the errors are grouped such that a number of successive blocks contain errors, the more powerful low rate code is in use for the duration of poor transmission conditions.

Transmitter:

```
Begin
    If block(i) rejected
        then   last block no. = next block no.
               next block no. = i
               retransmitting = true
    If retransmitting and next block no. = last block
        then retransmitting = false
    If block(i) rejected or errored
        then   count = c  {sustain factor}
               code state = low rate
    If count > 0 then decrement count
    If count = 0 then code state = high rate
    Encode block( next block no. )
    Send block
    Increment next block no
End.
```

Receiver:

```
{ Note that it is assumed that the receiver is able to deduce the code
rate used to encode the received block }
Begin
    Decode the received block(j)
    If j = next rcv block no then
        if block(j) contains uncorrectable errors
            then return reject block(j)
        if block(j) contains a correctable errors
            then return errored(j)
                 increment next rcv block no
    if block(j) contains no errors
        then   return acknowledged(j)
               increment next rcv block no
End.
```

5.7.2 Performance of the adaptive ARQ scheme on a burst error channel.

The codes selected for comparison in the simulation were the BCH codes (1023,923,10), (963,923,4), (933,923,1) and (923,923,0), with a frame header size of 48 bits (i.e. a data segment of 875 bits), and a sustain factor of 16 frames. The value of k, the message length was kept constant, but the block length n allowed to vary. Thus a block may initially be transmitted using a (933,923,1) code with 875 data bits, 48 header and error detection bits and 10 error correction parity bits, but retransmitted using a (1023,923,10) code. The schemes compared were selected to be of roughly comparable complexity:-

(i) Go Back N ARQ

(ii) Selective Repeat ARQ

(iii) Hybrid GBN ARQ with adaptive selection of code rate

(iv) Hybrid GBN ARQ with fixed rate code

Table 5.p shows the performance estimates obtained from simulation using the GBN ARQ model with channel errors obtained from the Lewis and Cox data. In addition, the channel error information was passed through a 23 bit descrambler, to examine the effects of error extension (cf Section 4.3.2) on the error control

schemes. For comparison the efficiency of the various techniques on an error free channel is shown. Ideally a hybrid scheme should have the same performance as ARQ on the error free channel, but substantially better performance on the errored channels.

The results show clearly that the adaptive scheme is able to provide good performance on the sample of channel error data, and would be expected to perform well on channels with persistent bursts. The technique would not perform as well on a random error channel, or channel with short bursts, due to the delay before switching code rate.

The best adaptive scheme is that using the (1023,923) and (963,923) codes, as the high rate code is near optimum for the channel, which may be verified by examining the performance of the fixed code rate hybrid schemes.

To obtain some insight into the operation of the adaptive hybrid scheme, the number of correctable and uncorrectable blocks were counted for the (1023,923)/ (923,923) scheme for different channel delays. These are shown in Table 5.q.

The effects of increasing delay are apparent in the increasing proportion of low rate blocks, and the increasing proportion of these low rate blocks that contain no errors. This is due to the slower response of the transmitter to changes in channel conditions.

ARQ Scheme	Efficiency for :- No errors	Lewis & Cox Raw	Lewis & Cox Scrambled.
GBN ARQ	**0.95**	0.64	0.64
GBN -Type II	**0.95**	0.74	0.74
SR ARQ	**0.95**	0.77	0.77
Adaptive			
(963/923,923)	**0.95**	0.79	0.70
(1023/923,923)	**0.95**	0.80	0.74
(1023/933,923)	0.94	0.82	0.74
(1023/963,923)	0.91	**0.84**	**0.75**
Fixed rate			
(933,923)	0.94	0.74	0.63
(963,923)	0.91	0.81	0.73
(1023,923)	0.86	0.80	**0.75**

Table 5.p Efficiency of ARQ schemes, obtained by simulation based on both scrambled and unscrambled channel error data (Lewis and Cox data). Channel delay is 1000 bits.

Value of N (acknowledgement delay - frames)	**1**	**3**	**5**
Efficiency	0.92	0.82	0.77
Proportion of blocks sent using:-			
High rate code	92%	85%	81%
Low rate code	8%	15%	19%
Proportion of blocks encoded using the high rate code with:-			
no errors	85%	85%	84%
correctable errors	6%	6%	6%
uncorrectable	9%	8%	10%
Proportion of blocks encoded using the low rate code with:-			
no errors	46%	62%	72%
correctable errors	49%	33%	26%
correctable by high rate code	13%	8%	9%
uncorrectable	4%	4%	2%

Table 5.q Some statistics obtained from the simulation of the (1023,923)/(923,923) adaptive ARQ scheme.

It may be possible to improve the efficiency of the adaptive schemes still further if the transmitter had earlier warning of degrading channel conditions. If forward and backward channel data streams share the same path, the transmitter could estimate the error rate in the forward direction by observing the number of errored blocks or acknowledgements received on the backward channel.

5.7.3 Performance of the adaptive hybrid ARQ scheme on the binary symmetric channel.

The efficiency of the adaptive hybrid ARQ scheme on the binary symmetric channel may be calculated using the expression (derived in Appendix C):-

$$\text{efficiency} = \frac{(k - h)}{(P_f.n + (1-P_f).k)} \cdot \frac{1}{T}$$

where P_f is the probability of a frame being transmitted with forward error correction, and p the number of additional parity bits used.

As the channel is an independent error channel, the probability P_f is simply the probability that the c previous frames were not errored (where c is the number of frames for which the FEC is maintained):-

$$P_f = 1 - (1 - P(m>0,k))^c$$

The average number of retransmissions T is basically the same as that for Go Back N ARQ, although the frame error probability will depend on P_f, giving:-

$$T = 1 + \frac{N.(1 - P_{nr})}{(1 - P(m>t,n))}$$

where P_{nr} is the probability of the first transmission being successful:-

$$P_{nr} = P_f.(1-P(m>t,n)) + (1-P_f) \; . \; (1-P(m>0,k))$$

Figures 5.r(i) and (ii) show the results for channel delays of 1000 and 5000 bits. For comparison, Go Back N, Selective Repeat, and Types I and II hybrid ARQ schemes are also shown. The binary symmetric channel with large delay represents a fairly severe test of the various schemes.

The adaptive hybrid ARQ scheme shows some interesting features. At low error rates, the efficiency is close to that of both Go Back N and Selective Repeat ARQ. As the error rate increases, efficiency drops, but not as much as that of GBN ARQ, due to the effect of the forward error correction, which is being invoked more often. At an error rate of approximately 0.0001, the efficiency rises, until it coincides with that of the Type I hybrid scheme. This is

because the high error rate results in the forward error correction element being continually invoked, and hence the adaptive scheme acts exactly as a Type I hybrid.

The effect of increasing the delay to 5000 bits is shown in Figure 5.r(ii). The dip in the efficiency curve has deepened, however this is dependent on the sustain factor, the delay built into the adaptive algorithm, and increasing this value would increase efficiency in this area, at the cost of a slight reduction in efficiency under low error rate conditions.

The performance of the adaptive scheme in comparison to the others shown, is generally good. The scheme has the benefits of the Type I hybrid at high error rates, and of simple ARQ at lower error rates. At high error rates the method outperforms Selective Repeat ARQ, as the forward error correction element reduces the number of retransmission requests.

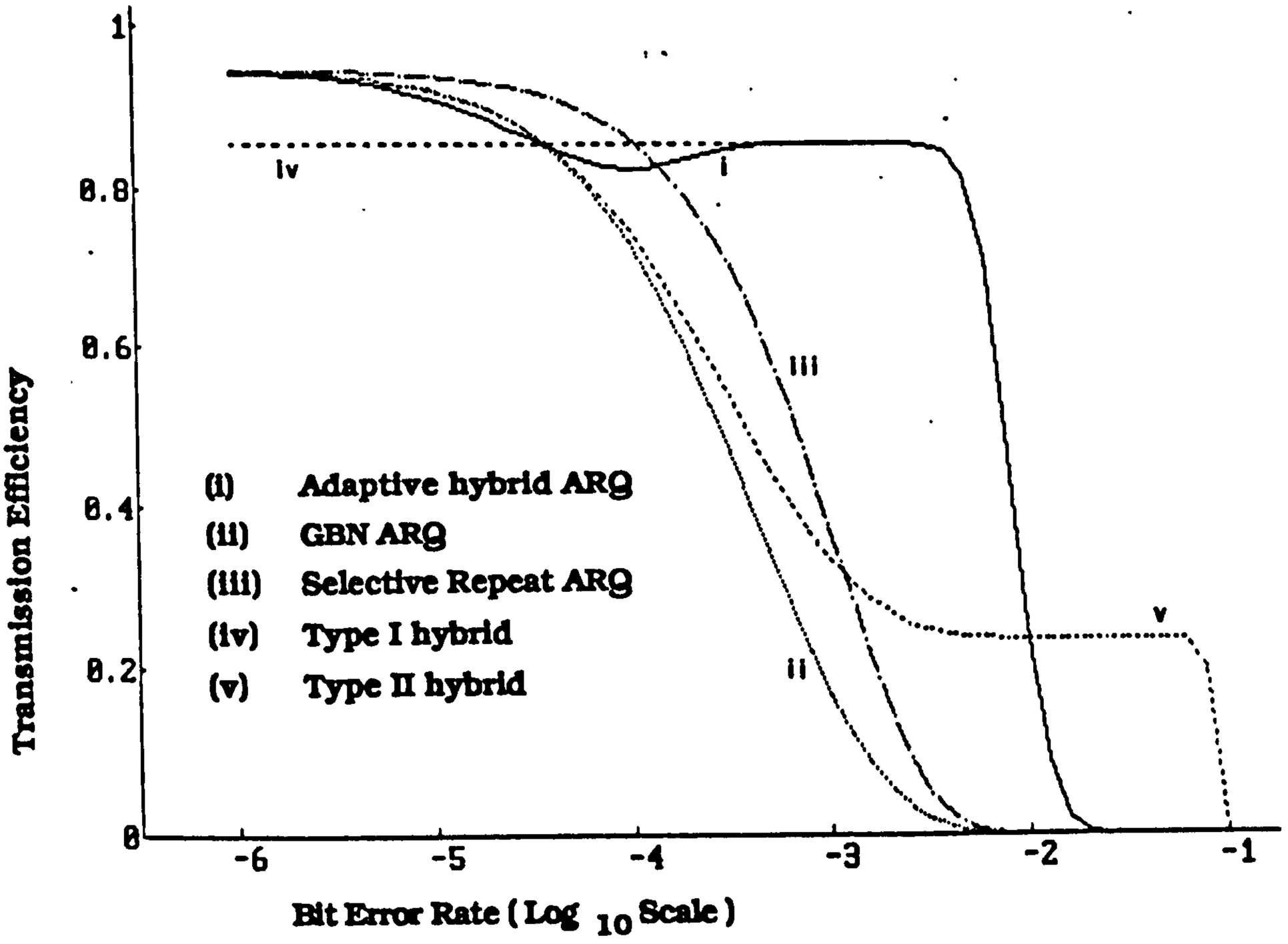

Figure 5.r(i) Transmission Efficiency of five ARQ schemes, for the binary symmetric channel, with zero delay.

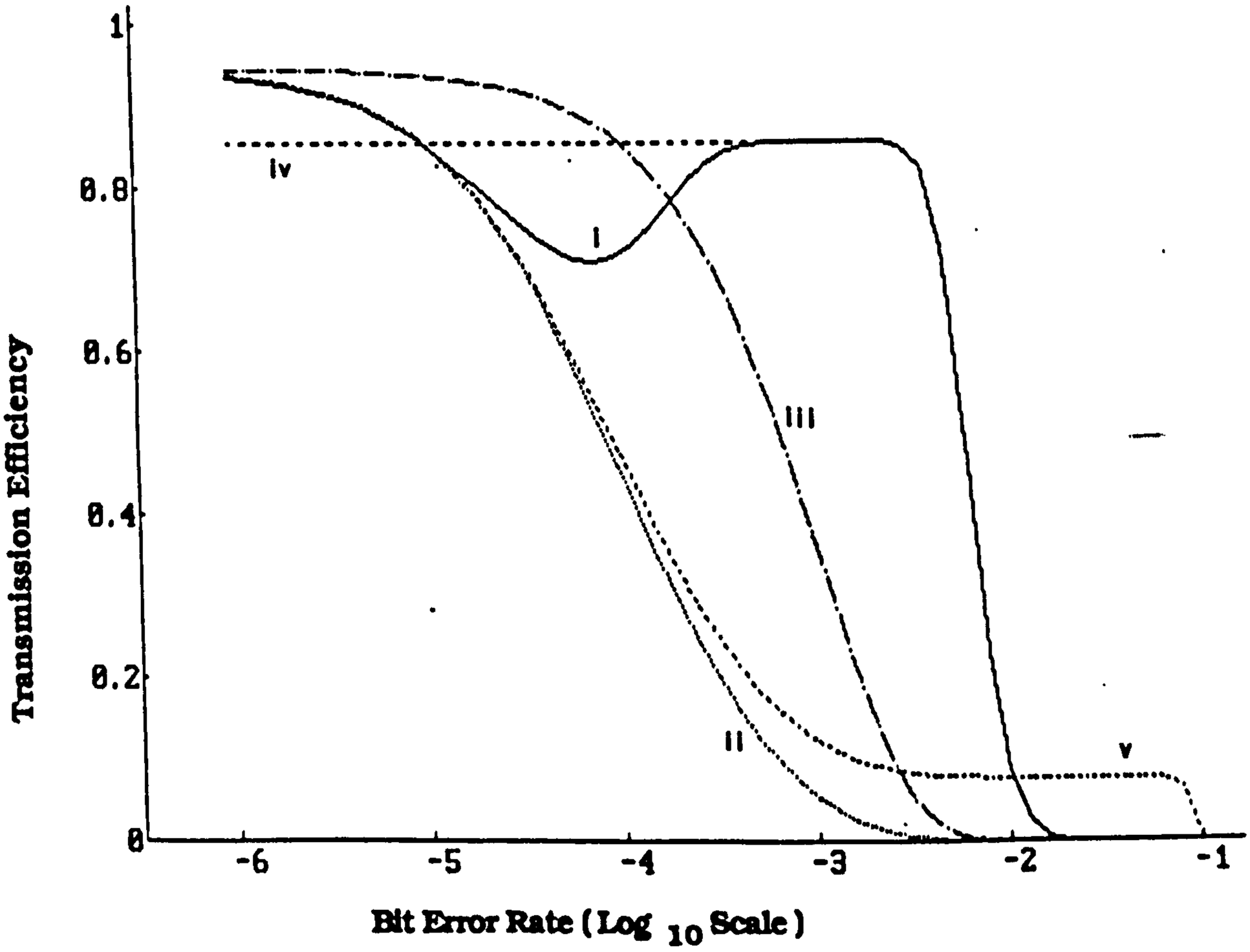

Figure 5.r(ii) Transmission Efficiency of five ARQ schemes, for the binary symmetric channel, with a 1000 bit channel delay.

5.7.4 Selection of code rate and sustain factor.

The preceding sections have shown that the adaptive hybrid ARQ scheme is effective for certain values of the sustain factor and code rate. The aim of this section is to discuss the choice of these parameters.

On the binary symmetric channel, the adaptive scheme behaves as simple ARQ at low error rates, and as type I hybrid ARQ at high error rates. Ideally the sustain factor would be zero at low error rates to avoid any loss in performance, and infinitely large at high error rates to maintain the system in type I mode. The crossover occurs where the efficiency of the two ARQ schemes is the same:-

$$\frac{(k-h)}{k}\,\frac{1}{\left(1+\frac{N\,P_E}{(1-P_E)}\right)} \;=\; \frac{(k-h)}{n}\,\frac{1}{\left(1+\frac{N\,P(m>t,n)}{(1-P(m>t,n)}\right)}$$

At low to medium error rates, P(m>t,n) is very small, and hence the crossover between ARQ and type I mode should occur at:-

$$P_E \approx \frac{(n-k)}{(n+2.D)}$$

Figures 5.s(i) and (ii) show the efficiency of the adaptive scheme on the Lewis and Cox channel error data, for a range of values for the sustain factor of 1 to 16. Each graph gives the efficiency curve for a number of values of channel delay.

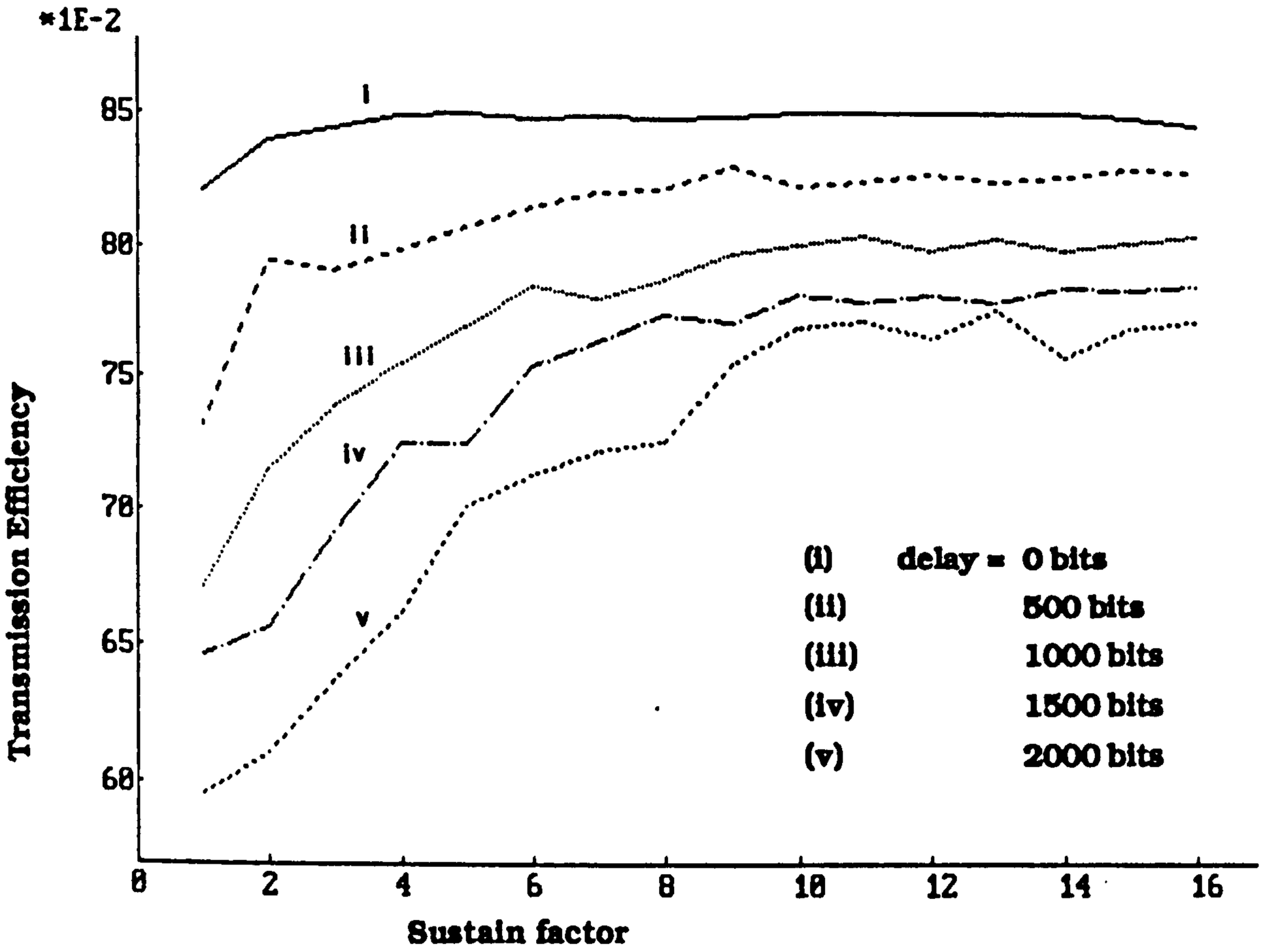

Figure 5.s(i) Transmission efficiency of the adaptive hybrid ARQ scheme on the Lewis and Cox channel data

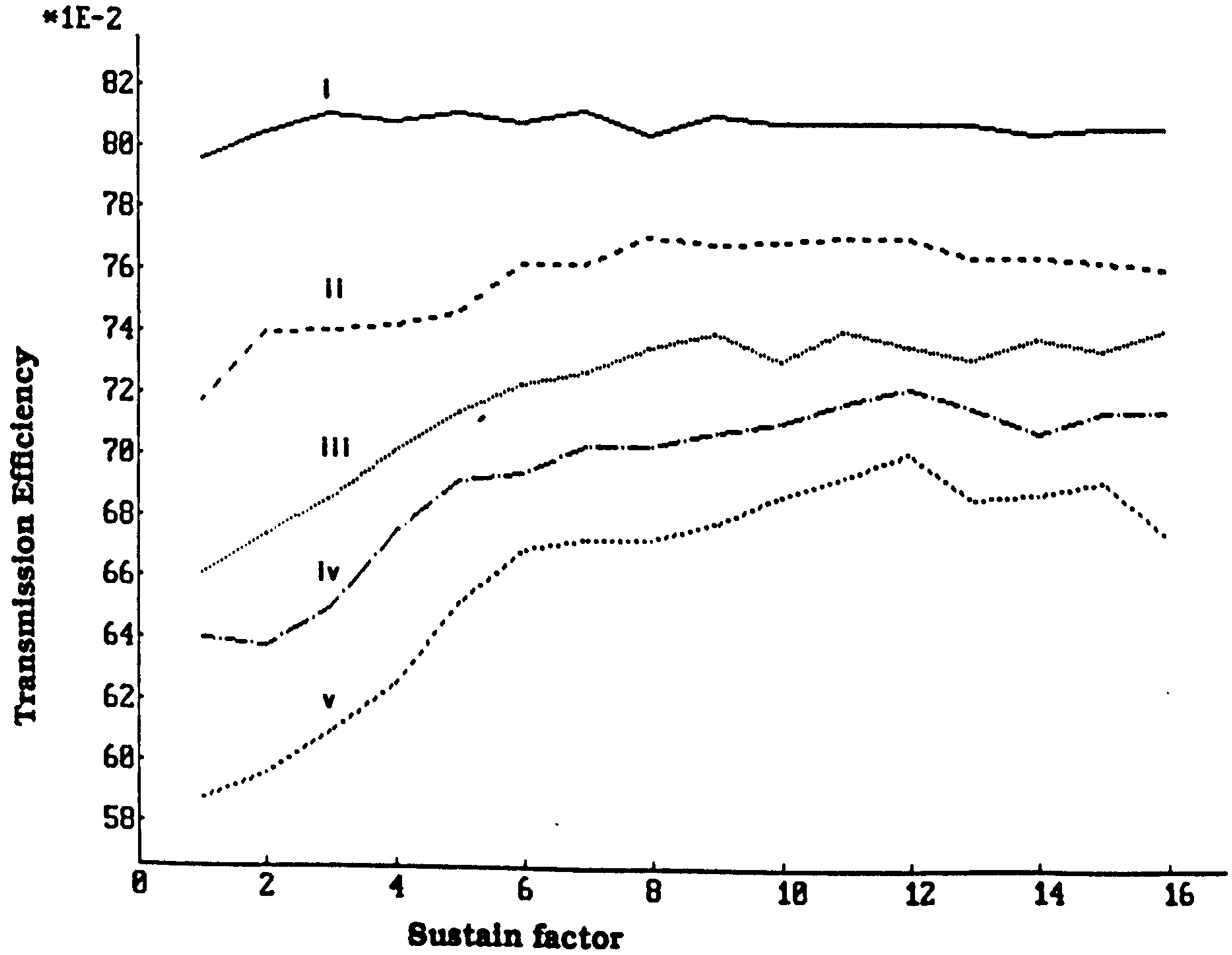

Figure 5.s(ii) Transmission efficiency of the adaptive hybrid ARQ scheme on the scrambled Lewis and Cox channel data

Figure 5.s(i) shows that the efficiency is relatively insensitive to the sustain factor for low channel delay, but the sustain factor needs to be substantial if the channel delay is increased. The value of 16 selected in the preceding sections would appear to be a reasonable choice for this set of data, although the curve shown for zero delay indicates that this value is just over the optimum region.

Figure 5.s(ii) shows the equivalent results for the scrambled Lewis and Cox data. The performance is markedly affected by the higher error rate, but the observations made above are still applicable.

In Section 5.7.2, the use of a (1023,923,10) low rate code, and (923,923,0) high rate code (i.e. no forward error correction) was proposed on the grounds that high efficiency was desirable when low error rate conditions exist. A small number of alternative code rates were tried, and the results compared.

Figure 5.t shows the efficiency of the adaptive scheme for a range of code rates. The optimum rate for the FEC element of the adaptive scheme for this channel is between 0.92 and 0.96, and depends on the channel delay.

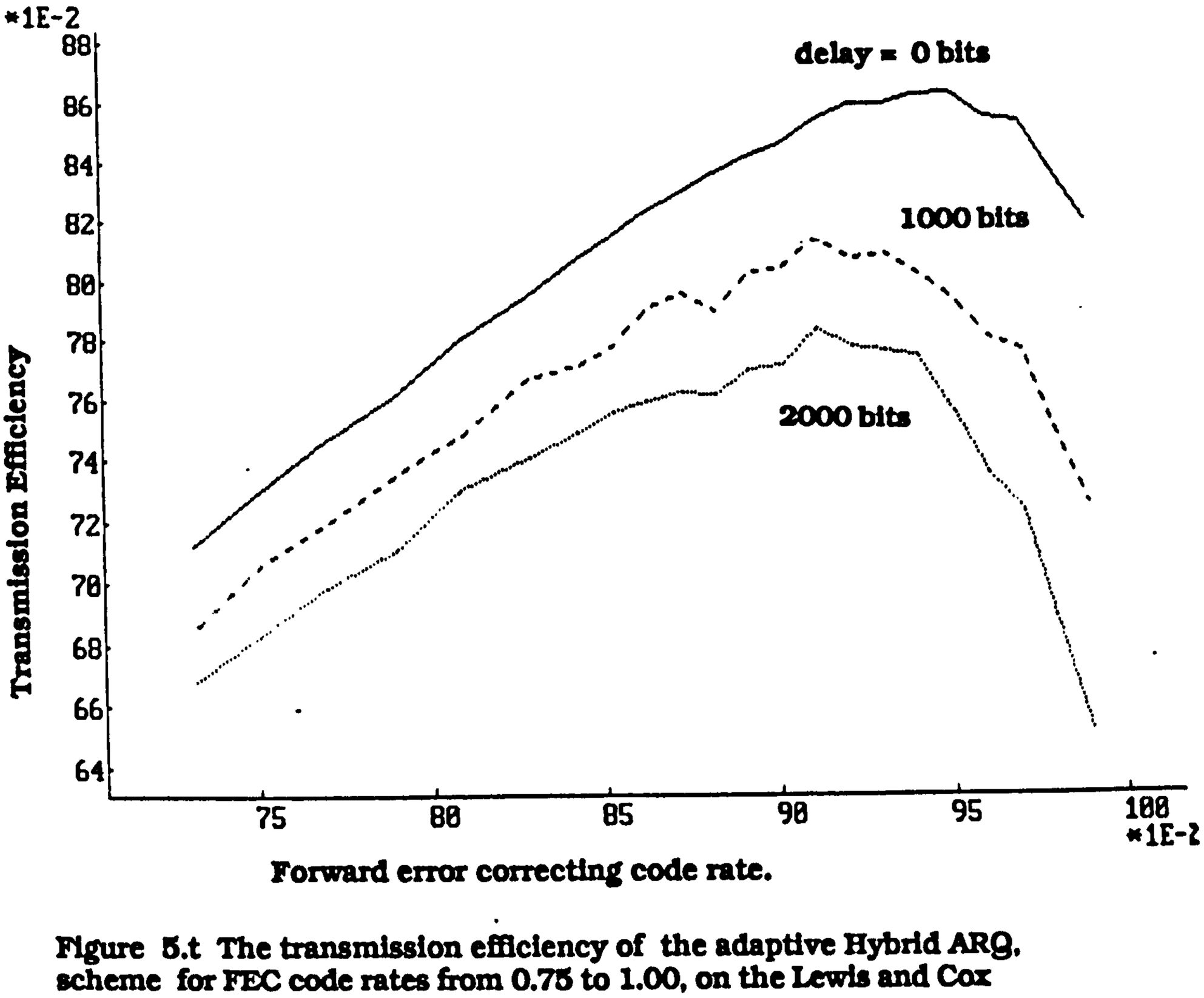

Figure 5.t The transmission efficiency of the adaptive Hybrid ARQ, scheme for FEC code rates from 0.75 to 1.00, on the Lewis and Cox channel error data.

To summarise, the choice of the sustain factor does not seem critical. On the binary symmetric channel an optimum only exists at some given error rate. From the results obtained on the Lewis and Cox channel error data, the earlier choice of 16 would seem reasonable, although a larger value would improve performance if the channel delay is substantial. The selection of code rate is still fairly important, which implies that the use of variable redundancy forward error correction may achieve more than Section 5.7.2 intimated.

5.8 Residual error rate

The uncorrected error rate, i.e. the probability of a bit error in the ARQ receiver output, depends on the error detecting capability of the error detection code used to construct the ARQ frame. For a hybrid ARQ scheme, there will be some interaction between the forward error correcting and the error detecting codes. Klove and Miller (ibid) found that, for certain linear block codes used on the binary symmetric channel, the reliability of the error detection code is increased if its minimum distance is less than half of the minimum distance of the inner error correcting code, i.e.:-

$$d_{min}\ (FEC) > 2\ \ d_{min}\ (\text{error detection code})$$

In order to make realistic estimates of the uncorrected error rate, the error pattern and code weight distributions must be known. If a sufficiently accurate model can be constructed, simulation may be used (Muntner and Wolf 1968).

In Section 4.2, the error pattern distribution of high speed voiceband data communications systems was discussed. The errors typically occur in bursts with length and weight dependent on the modem design. The use of long scramblers would result in bursts of length 23 bits or more, as found by Balcovic et al. Provided that the scrambler polynomial and generator polynomial are mutually prime, and the unscrambled error patterns are less than (n-k) bits in length, even long error patterns can be detected, as discussed in

Section 4.5.6.

Funk (1982) examined another source of error extension, synchronization failure. The SDLC protocol and its derivatives (HDLC, X25, ..) use a bit oriented frame synchronization mechanism, described in Section 6.7, with a 16 bit cyclic error detection code of minimum distance four. The receiver relies on the detection of a unique bit pattern (01111110) within the received data stream for location of the end of a frame.

In order to achieve data transparency (i.e. to allow the message to contain any bit pattern, including the synchronization sequence) a bit stuffing mechanism is used. After any sequence of five 1's (excluding the synchronization sequence) the transmitter inserts a zero; similarly, the receiver removes a zero if preceded by five 1's.

A cyclic code should be capable of detecting all errors of weight three or less, however a single bit error is sufficient to cause loss of synchronization, with consequent large error extension. Funk discusses the following possible causes of undetected error:-

(i) Spurious flags (synchronization characters) may be created within the frame. As the bit stuffing mechanism inserts a single zero after a series of five 1's (0111110XX), a single error is sufficient to corrupt the inserted 0, and hence create a false synchronization character. The effect of this is the division of the frame into two parts.

(ii) The flag character (01111110) may be corrupted by a single error, causing the frame to merge with the following frame.

(iii) An abort sequence is a series of eight consecutive 1's, and may be caused by failure of the bit stuffing mechanism, as in (i).

(iv) A bit may be lost if, in a sequence of the form 0111010XX a 0 is corrupted, causing an incorrect bit removal operation.

(v) A bit may be gained if, in a sequence of the form 0111110XX a 1 is corrupted, resulting in the bit removal not being carried out.

The undetected error probability for case (i), is given as:-

$$P(\text{uncorrected error}) \approx (n-32) \cdot p_t \cdot 4.8\times10^{-7}$$

due to false flags, where n is the block length, and p_t is the transition probability for the assumed binary symmetric channel. For a bit error rate of 0.001, and a block length of 1023, the predicted residual error probability due to false flags would be 2×10^{-8}.

Reliability may be improved by the use of lower rate error detection codes; (n,n-32) is a common alternative to the (n,n-16) code used in most ARQ systems. The lower rate will of course result in a slight reduction in efficiency. Goodman and Farrell (ibid) describe an ARQ system for a high frequency radio channel, which

employed adaptive selection of an error detecting code to suit the prevailing channel conditions. The range of frame lengths and the type of channel used preclude comparison of their results, however it is interesting to note that they report that frame synchronization problems led to bursts of uncorrected errors.

5.9 Discussion

This chapter discussed a number of ARQ and hybrid ARQ techniques, and considered the performance of four schemes under a range of channel conditions. The results showed consistent differences in performance between the error control schemes, providing some basis for the selection of an appropriate method for the test channels. In addition, an adaptive hybrid ARQ scheme was proposed, that showed good performance under simulated channel conditions.

Ideally an error control scheme should provide high efficiency under zero error conditions, and low uncorrected error rate under poor channel conditions. The importance of the first criteria may be justified by Balcovic et al (1971) who found that between 20 and 50 percent of calls in the A.T. & T. 1969-1970 network survey were error free.

Two effects due to the channel error distribution were observed when comparing the performance of the ARQ schemes on the various channel models. The clustering of errors that occurred in the recorded data given by Lewis and Cox tended to give a larger number of error free blocks, and a higher density of errors within the errored frames than for the binary symmetric channel. This improved the performance of ARQ, and reduced the effectiveness of the forward error correction component of hybrid ARQ.

The choice of code rate for a hybrid ARQ scheme was

discussed in Section 5.6. For some known channel conditions it is possible to select an optimum code rate, however the channel conditions are generally not stable. The disadvantage of hybrid ARQ schemes employing fixed rate codes is the loss in efficiency during error free periods.

An adaptive hybrid ARQ scheme was proposed in Section 5.7, which behaved well under a range of simulated test conditions. The scheme was compared to Go Back N ARQ, hybrid ARQ, ARQ with parity retransmission, and to Selective Repeat ARQ, and was found to give reasonable performance under both errored and error free conditions. On the random error (BSC) channel, the scheme performed very well, with an efficiency close to that of GBN ARQ at low error rates, and equal to that of an equivalent Type I hybrid scheme at high error rates.

The selection of code rate for the adaptive scheme still proved to be important, and should be investigated further.

The choice of scheme depends on the degree to which the channel conditions are known, and on the allowable complexity. Simple Go Back N ARQ provides reasonable performance on burst error channels such as the telephone channel, as Burton and Sullivan (ibid) concluded, however substantial performance improvements may be made if an increase in complexity is acceptable.

Selective Repeat ARQ provides better performance than Go Back N ARQ for channels with delay, and may be an acceptable technique given that the error rate is low. If the channel error rate

is substantial, with a bit error rate greater than 0.001 for example, hybrid Go Back N ARQ offers better performance.

The adaptive hybrid ARQ scheme appears to give the promise of high efficiency under both low and high error conditions, and would be preferable to a scheme using a fixed rate forward error correcting code or parity retransmission.

To summarize, the ARQ schemes that seem appropriate to the telephone channel are:-

(i) Go Back N ARQ, when low complexity is a criteria.

(ii) Selective Repeat, when channel delay is high, and the error rate is low or the errors occur in short bursts.

(iii) Adaptive hybrid ARQ, when high efficiency is required under a range of error conditions.

6 SYSTEM DESIGN CONSIDERATIONS.

6.1 Introduction.

The objective of this study was to investigate the use of data compression and error control techniques in data communication systems for the telephone channel. The preceding chapters have examined these areas in some depth, and have suggested appropriate techniques. The aim of this chapter is to discuss some of the practical implementation considerations for a system employing both types of coding technique, and to provide some estimates of the system performance.

The discussion of data compression schemes highlighted several problem areas, including the difficulty of designing a compression algorithm suitable for a source with unknown or time varying characteristics. Several adaptive codes were proposed in Chapter 3, with differing degrees of complexity and performance.

In Chapters 4 and 5, error control techniques suitable for burst error channels such as the telephone channel were discussed. Three alternative automatic retransmission error control techniques were suggested, each more appropriate to some stated channel conditions.

Figure 6.a may help to place the data compression and error control subsystems in context. A terminal provides a source of asynchronous characters, which are read into the error control/data compression system (the ECU), and are stripped of their

asynchronous framing elements and stored in a buffer. Characters are read from the buffer, compressed and passed into the ARQ stage. Frames from the ARQ transmitter are sent through the forward error correction code encoder (if applicable) and then to the modem transmit channel. The receive path follows the reverse order to the transmit channel, however an additional path is provided from the ARQ receiver to the ARQ transmitter for acknowledgements.

More formally, the logical and procedural interface between two communicating systems is generally defined by a class of protocols. The protocols may include a number of elements or functions, including ARQ and data compression. For correct operation of the communications interface, both systems must interpret the information flow identically. However one of the principal problems in telecommunications has been the low degree to which interworking between communications equipment is possible.

The International Standards Organisation (ISO) proposed a seven layer model, the Open Systems Interconnection (OSI) model, to introduce a framework for co-ordinating the development of data communications standards, and for placing existing standards in perspective,

The OSI model has achieved general acceptance, and has been adopted by the CCITT as the X200 series recommendations. In addition most major companies involved in the development of data communications equipment have supported the introduction of the model.

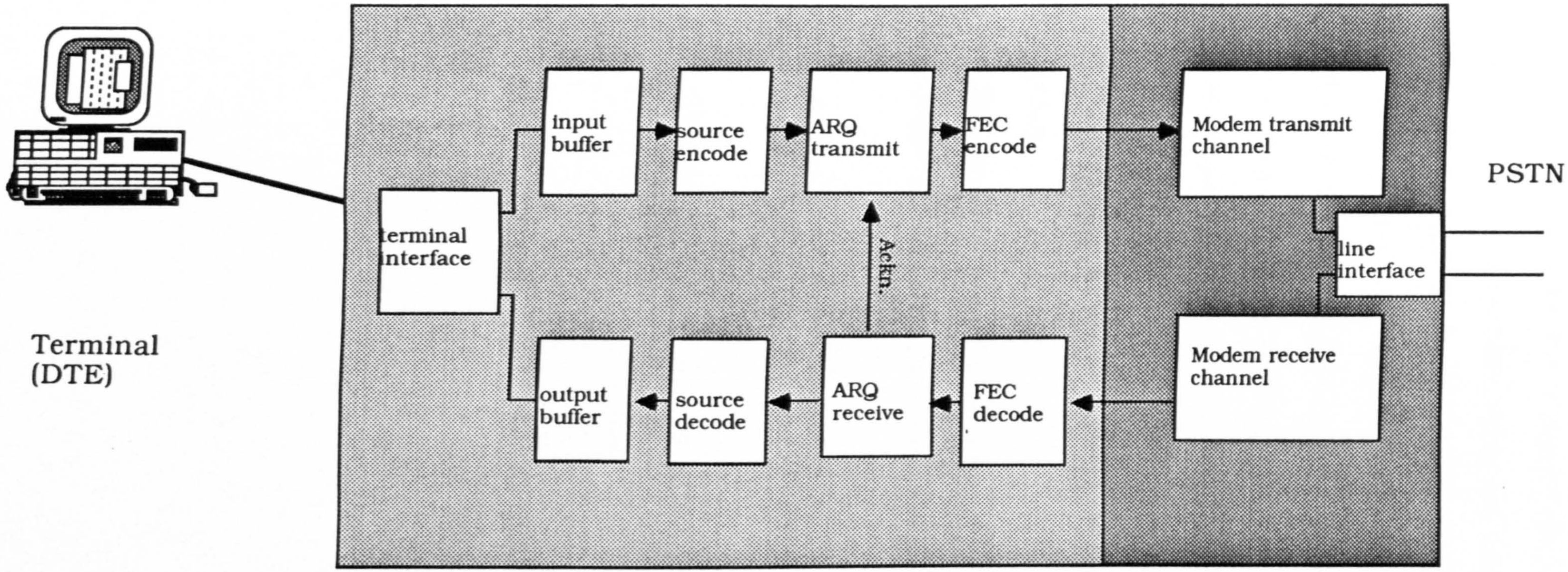

Figure 6.a General block diagram of the error control / data compression system

The seven layers of the model are shown in Figure 6.b. Those of immediate interest to this study are layers one, two and six. Layer one, the physical level, is concerned with the electrical and mechanical aspects of transmission, and provides the means to transmit bits of data across a continuous communications path. Layer two, the data link layer, provides an error controlled path across a physical communications link, and may for example incorporate ARQ. Layer six, the presentation layer, incorporates more general data transformation functions, such as data compression and encryption.

Figure 6.c illustrates how the error control and data compression system may be represented in terms of the OSI model.

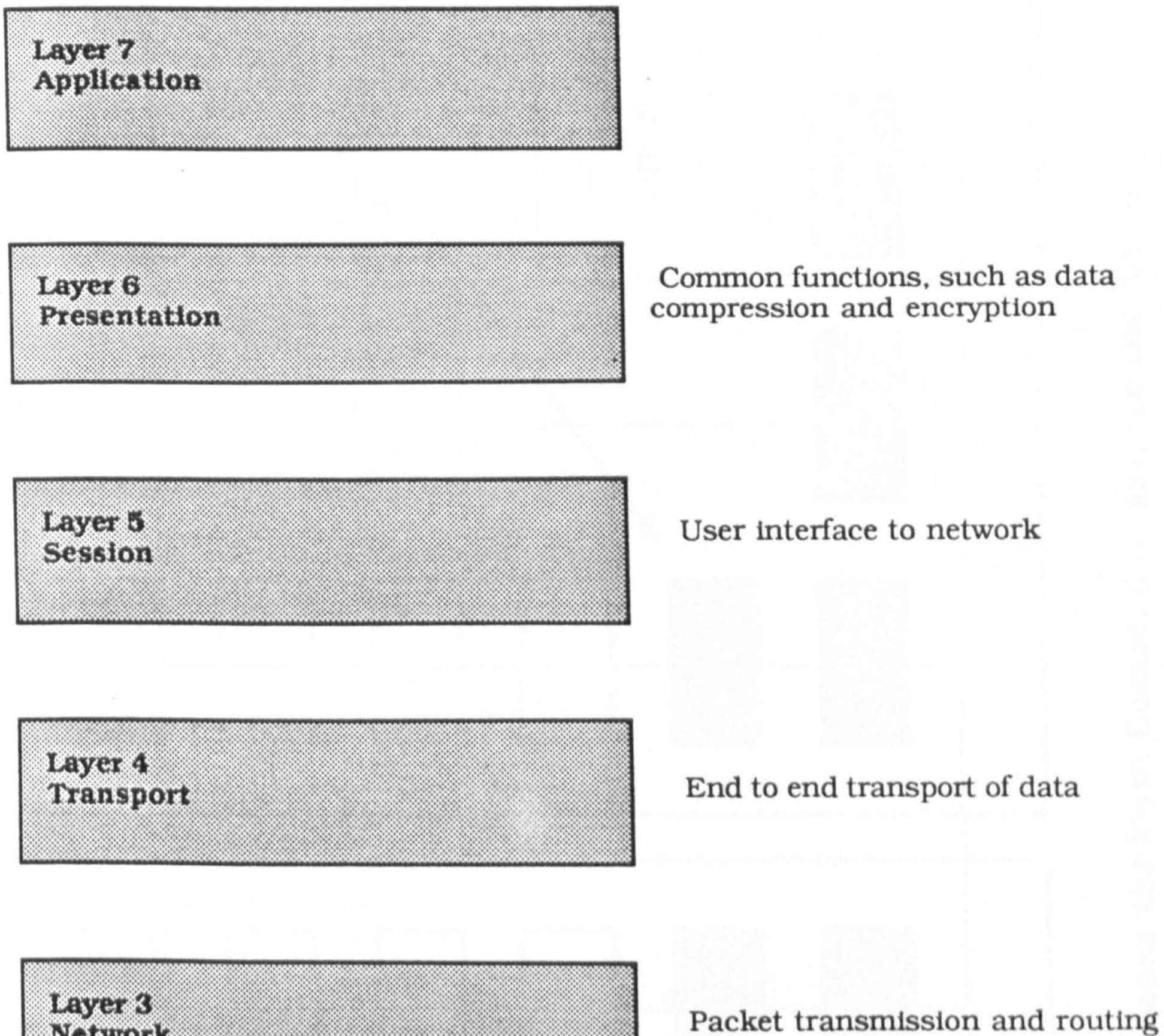

Figure 6.b The Open Systems Interconnection Model.

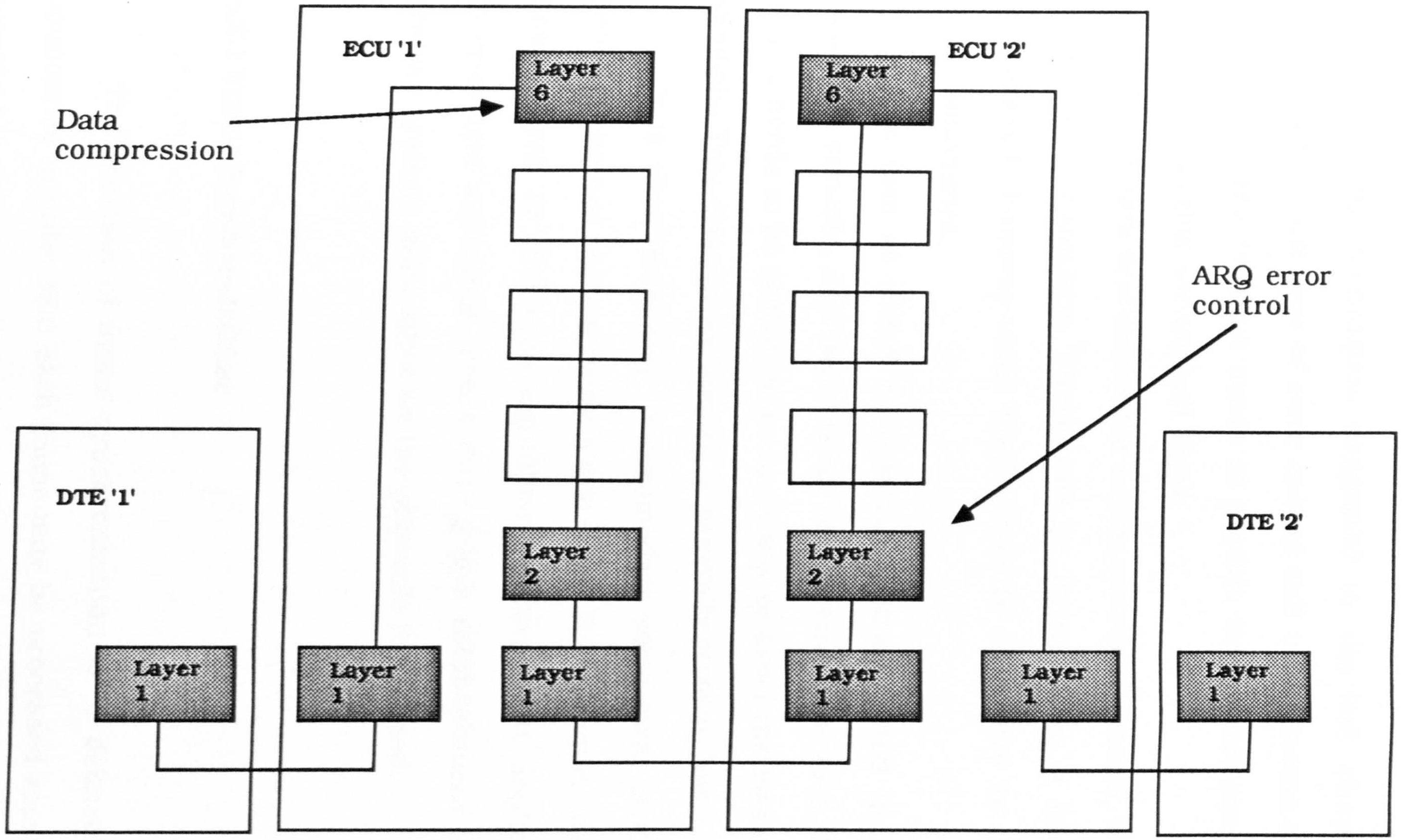

Figure 6.c Correspondence between the Error Control Unit and the OSI model

6.2 ARQ Protocol Design.

The ARQ techniques suggested in the last chapter were examined from the view of error control and transmission efficiency. In fact an ARQ protocol needs to provide more than simply error correction, as this section will show.

When data first enters the communications system, it consists of discrete characters. These will be formed into a frame and transmitted, however some mechanism is necessary to delineate frame boundaries.

The rate at which information is transmitted in an ARQ system is variable, and depends on the current error rate. Thus the source needs to be controlled in some way to avoid the loss of source symbols. This class of techniques is generally termed flow control.

It is often necessary to accomodate some form of signalling between the two communicating systems. For example, end to end flow control, or exception signalling. An important application of inter-system signalling occurs during link establishment in which the two systems must agree on the protocols to be used.

6.2.1 Frame Synchronization

The function of frame synchronization is to delineate frame boundaries in order that each frame may be processed seperately. A specific bit pattern is included within the transmitted frame, which may be detected by the receiver. The pattern may be a short

sequence of bits, or may be distributed through the frame.

One problem that occurs with this type of frame synchronization technique relates to the difficulty of preventing the synchronization pattern from occuring within the data field of the frame. This may of course be cured by not allowing the source to emit symbols which cause false synchronization, or by using a fixed length frame. It is however desirable to place few constraints on the manner in which the system is used.

A common approach is byte oriented frame synchronization, as typified by the IBM BISYNC protocol. The frame is composed of a series of bytes or octets, the first two of which are synchronization characters. If a synchronization character appears within the information field of the frame, a defined control character DLE is inserted before it. The receiver may thus discriminate between true and false synchronization characters.

The second common approach is bit oriented synchronization method used by IBM SDLC (Donnan ibid) and its derivatives, already mentioned in Section 5.8. This uses the bit sequence 01111110 as a synchronization pattern. If a sequence of five 1's is detected in the frame prior to transmission, a 0 is inserted. The receiver may detect the start of a frame by testing the input bit stream for the synchronization pattern, but also reverses the bit stuffing procedure by removing any 0 preceded by five 1's.

The frame synchronization techniques work well in practice, as shown by the popularity of the protocols, but are susceptible to errors. In Section 5.8 the effects of errors on the SDLC approach was discussed.

6.2.2 Flow Control

Flow control is necessary to provide a mechanism for regulating the transmission rate of various elements of a communications link to prevent congestion or loss of data (Ahuja 1985). On the simple point to point link under consideration three flow control procedures are needed for each direction of transmission.

(i) To control the flow of data from the terminal into the error control system, to prevent data from the terminal being lost whilst the error control system is retransmitting frames.

(ii) To control the flow of data from the error control system into the terminal. This would be applied by the terminal to prevent for example, loss of data whilst performing other tasks, or applied by the user of a terminal to prevent the screen contents from scrolling whilst it is read.

(iii) To control the flow of data across the channel, to prevent the receiver from running out of buffer capacity whilst the far end terminal is applying flow control as in (ii).

Flow control across the terminal / error control system interface is usually accomodated by one of two techniques RTS/CTS or X-ON/ X-OFF. The first technique makes use of the control lines

on the interface, RTS/CTS for the EIA RS232 definition of the interface, and circuits 105 and 106 within the CCITT V24 recommendation. The second technique is an in-band method, which employs two ASCII control characters to turn on and turn off the data flow.

Flow control across the channel is implemented in two ways, explicit control signals and the use of the frame number window.

The first of these is simple, a control message is sent to indicate that no further data frames may be accepted, and a further control frame used to indicate that data flow may be resumed. In the SDLC protocol the two frames are termed RNR (receive not ready) and RR.

Information frames are numbered to allow retransmission requests to be made. The number is a fixed width binary field, and hence is incremented modulo M, where M depends on the field width. Within this number range a window may be defined, and the transmitter programmed to send frames only whilst no unacknowledged frames exist with sequence numbers outside the window. For example modulo 8 numbering may be used, with a window of 4. The transmitter may send four frames, but cannot send any more until an acknowledgement is received for at least one of the transmitted frames.

6.2.3 In Band Signalling

Some examples of in-band signalling have already been given, the RR and RNR flow control frames of the preceding section. Other control frames may be needed for exception signalling. The most obvious example of exception signalling is the break signal, which is often used in terminal applications to escape from some catastrophic situation. The CCITT recommendation X.3 defines several alternative actions that could be taken as a result of receiving a break from a terminal, including sending an *interrupt* frame or a *indication of break* frame. The principal point is that the break key on a terminal does not produce an ASCII character, and hence would not normally result in an indication being transmitted across an ARQ link.

6.2.4 Link Establishment and Clearing

The link establishment phase of the communications cycle has two functions, firstly to allow both of the communicating systems to move from the disconnected to the information transfer state, and secondly to permit the negotiation of parameters such as window size, and maximum frame size. The clearing or disconnect phase permits both systems to establish that no further information is to be sent, and then to clear down the link.

Within the HDLC protocol (Davies 1979), the SABM (set asynchronous balanced mode) control frame may be used to initiate

link establishment, UA (unnumbered acknowledgement) is used to confirm. Parameter negotiation may be accomodated with the XID (exchange identification) frame.

6.2.5 Transmission Efficiency

The system transmission efficiency will depend on a number of factors, of which only some were considered in the preceding two chapters. The finite frame sequence numbering, and the use of windowing for flow control will reduce the efficiency of the ARQ system. The effects will depend on channel delay and on frame length. If short frames are sent on a channel with moderate delay the transmitter will spend a considerable proportion of the time waiting for acknowedgements.

The analysis of the transmission efficiency of ARQ protocols using queueing theory rather than the approach taken in Chapter 5, allows the effects of window size and random distributions of frame lengths to be accomodated. Reiser (1979), Konheim (1980), Bux et al (1980), Agnostou (1984) and Hayes (1985) have applied this approach to ARQ protocols.

The ARQ system may be modelled by a set of queues (Figure 6.d). The transmitter is represented as a queue with service time $1/\mu_1$, which feeds M branches, where M is the window width. Each branch may hold only one frame at a time, and is made up of three service times, the round trip delay T, and the delay until an acknowledgement is sent $1/\mu_2$. Labetoulle and Pujolle (1981)

compared analytic results obtained using this approach with a simulation of the HDLC protocol, and obtained reasonable correspondence.

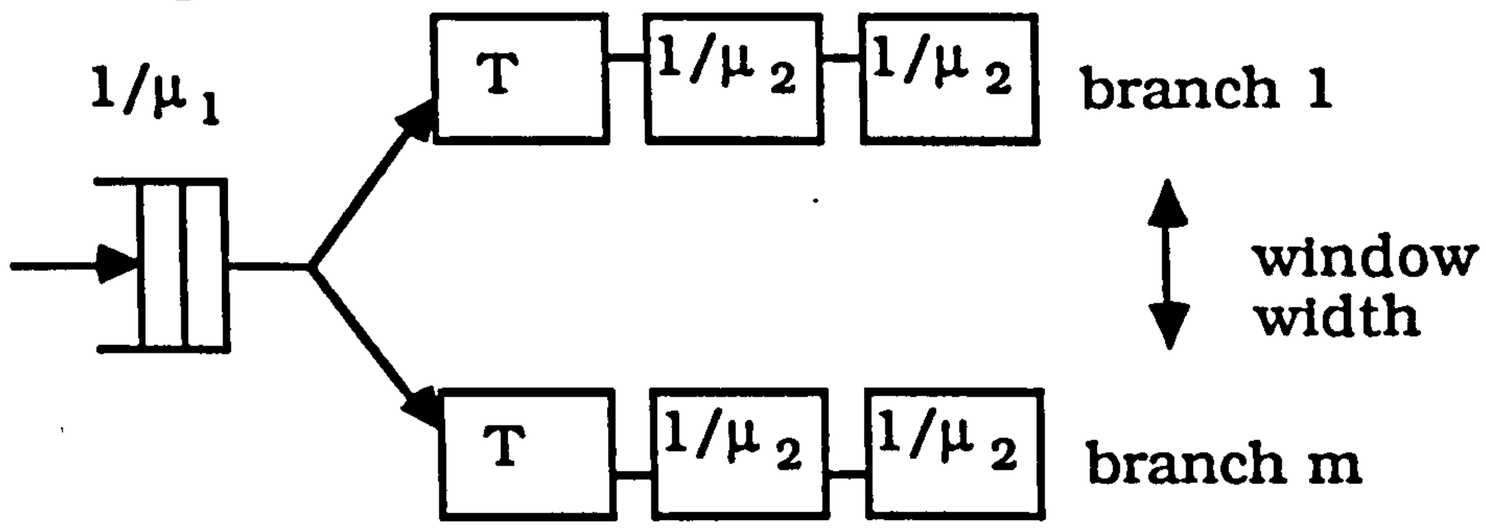

Figure 6.d Modelling an ARQ system with queues, to allow the effects of window size to be incorporated. (Hayes 1984).

The transmission efficiency of ARQ when used over a synchronous channel is enhanced by the gain made in reframing asynchronous characters. Typically an asynchronous character has seven data bits, one parity bit, one start bit and one or two stop bits. This format may be reconstructed at the receiver prior to onward transmission to the remote terminal. The effective gain is therefore forty to fifty percent, prior to transmission. The preceding chapter showed that transmission efficiencies of 0.6 to 0.95 were achievable under a wide variety of channel conditions, hence the likely overall transmission efficiency is between 0.8 and 1.4. The implication is that under good conditions the bit rate between terminal and error control system may be higher than the channel bit rate, even without data compression.

6.3 Integrating Data Compression into the Protocol

6.3.1 Adaptive Data Compression Algorithms.

Several alternative data compression schemes were proposed in Chapter 3, with varying degrees of complexity and coding gain. The choice of technique depends on the application, the memory and processing requirements, and the transmission speed.

The simplest scheme suggested used a simplified Huffman code with codeword lengths of 4 and 8 bits. The average encoded symbol length for the sample data employed was between 5 and 6 bits, although better performance may be obtained if the data is suitable for run length encoding. The encoder and decoder may be realised in software, without placing an undue burden on a standard microprocessor, and only require approximately two hundred bytes of memory each.

The more powerful compression algorithm discussed was based on the Ziv-Lempel technique. This provides substantially better performance, the average length found in tests was between 2.5 and 4 bits per encoded symbol. The additional performance was obtained at the expense of an increase in the memory capacity needed at both transmitter and receiver.

An implementation of the algorithm was proposed that was economical in both memory and processing requirements compared to other known implementations. The encoder and decoder may be

simply realised in software, but require up to 20 kilobytes of memory for the encoder and 28 kilobytes for the decoder. Less memory may be used at the expense of some slight degradation in performance.

6.3.2 Interaction with ARQ Protocol

Two significant problems arise when implementing a combined data compression and error control scheme, firstly the risk of uncorrected errors causing loss of synchronization of the source coder and decoder, and secondly the processing overhead introduced by the data compression sub-system.

The effects of uncorrected errors on adaptive data compression systems was discussed in Section 3.4. Errors will result in incorrect decoding of received codewords, and may possibly result in error extension. If the encoder and decoder are not in synchronism some mechanism is needed to prevent indefinite system misoperation.

Variable length codes may be designed such that codeword synchronization will be recovered, although adaptive variable length codes would benefit only if the error did not result in a modification to the symbol frequency table. The Ziv-Lempel decoder has some limited error detection capability. At some given instant there will generally be some unused codewords, which if received by the decoder, may be regarded as indicating loss of dictionary integrity.

For either code, the corrective action would be the same, to reset both encoder and decoder to some starting condition. This leads to an interesting point; it is tempting to apply the compression algorithm to data prior to buffering in the transmitter, to save storage. If the source encoder is reset, the data held in encoded form in the buffer will be effectively lost. Hence it is desirable to hold data in uncompressed form. On the other hand, if the data is held in uncompressed form, the adaptive source encoder will have adapted to the latest transmitted data. If the ARQ system requires any retransmissions, it would be necessary to reconstruct the ARQ frames, which would require the source encoder to be reset to the state that it was in immediately after the last acknowledged frame was encoded.

For example, an ARQ system sends frames 1...16. Frame 5 is received in error, and a retransmission requested. Frame 13 is also corrupted but the errors not detected until frame 14 is decoded, hence the source decoder is assumed to have lost synchronization with the adaptive source encoder.

(i) Data held in uncompressed form.

When the retransmission request is received, frame 5 must be retransmitted. This requires the uncompressed data that was originally in frame 5 and subsequent frames to be re-encoded. However the source decoder has successfully decoded frame 4, and has adapted to the source statistics at the end of that frame. The source encoder will need to be reset to the equivalent state. When the request to reset the encoder, due to loss of

synchronization in frame 13 is received, the code table is set to the initial or default state, and the data re-encoded and transmitted. This may result in a small amount of corrupted data being sent to the remote terminal, however the recovery process is straightforward.

(ii) Data held in compressed form.

When the retransmission request is received, the encoded data is retransmitted, this presents no difficulty. When however the request to reset the encoder is received, the compressed data is effectively lost.

One solution to the problem may be to store data in both compressed and uncompressed form, which would of course require more storage. The other alternative would be to assume that undetected errors are comparatively rare, and thus select option (ii).

The framing imposed by the ARQ stage can be of some help. If a frame always contains an integral number of source code words, the source decoder has two additional safeguards. The first bit in a received ARQ frame is the first bit of a source code word, hence the decoder is realigned each frame. The last bit in a received frame is the last bit of a source code word, hence the source decoder can detect a proportion of the errors missed by the ARQ stage. If the source decoder does detect any residual errors, a *reset* message should be sent to the source encoder, to ensure that the encoder and decoder code tables are resynchronized.

The data compression operations performed in the encoder and decoder fall into two classes, symbol related and housekeeping. The symbol related operations, such as encoding or decoding a symbol, occur regularly and may be designed to have little impact on the normal operation of the system.

The housekeeping functions of an adaptive data compression system include dictionary or frequency table updating, purging or scaling, and are likely to impose an occasional but heavy burden on the processor. It is quite possible that the system may lose transmission efficiency because the processor is performing dictionary maintenance and is temporarily unable to handle normal traffic.

6.4 Designing for Reliable Operation

The potential problems with adaptive data compression systems due to errors causing loss of synchronization are the tip of a rather large iceberg. The general problem is that of two finite state machines communicating over a noisy link, each can never know the precise state of the other. To design a reliable system it is necessary to construct the finite state machines in such a manner that no disastrous system state is ever reached. The difficulty of this task may be illustrated by reference to Zafiropulo (1980) in which specific examples are given of communications protocols that were found to contain design errors after acceptance.

Modelling or formal description of protocols has been an area of active research for a number of years. The well known techniques are Petri nets (Diaz 1982), and communicating finite state machines (Bochmann 1978, Milner 1980).

The reliability of the protocol is as important as the reliability of the error detection mechanism. There is little point in providing an error correcting system that may itself introduce errors. If the protocol is designed in accordance with the OSI model, and formal verification techniques are applied to the mathematical model and the software implementation of the protocol, the reliability of the system can be improved. The present difficulty is that many of the necessary tools are still in an early stage of development.

6.5 System Realization.

A number of options for the various system components have been given. This section will discuss the implementation of three systems 'A', 'B' and 'C', which represent realistic alternatives. The 'A' system is a low complexity moderate performance error control system, whilst the 'B' and 'C' systems offer higher performance at the expense of complexity.

The terminal interface of the three systems is similar, and is shown in Figure 6.e. Data is read from the DTE port into a buffer. The buffer level is monitored, and if it exceeds a threshold level flow control is applied at the DTE port, to inhibit the flow of incoming data. An output buffer holds data received from the ARQ system, until ready for transmission. The buffer level is similarly monitored, but flow control is applied by requesting the ARQ stage to send in-band flow control messages to the remote system.

The interface to the terminal must also handle break detection, although the ARQ stage would transmit an in-band control message to communicate the break to the remote terminal.

The interface lines (EIA RS232 or CCITT V24) are used to indicate terminal and modem status, provide timing for synchronous data, and allow out-of-band flow control.

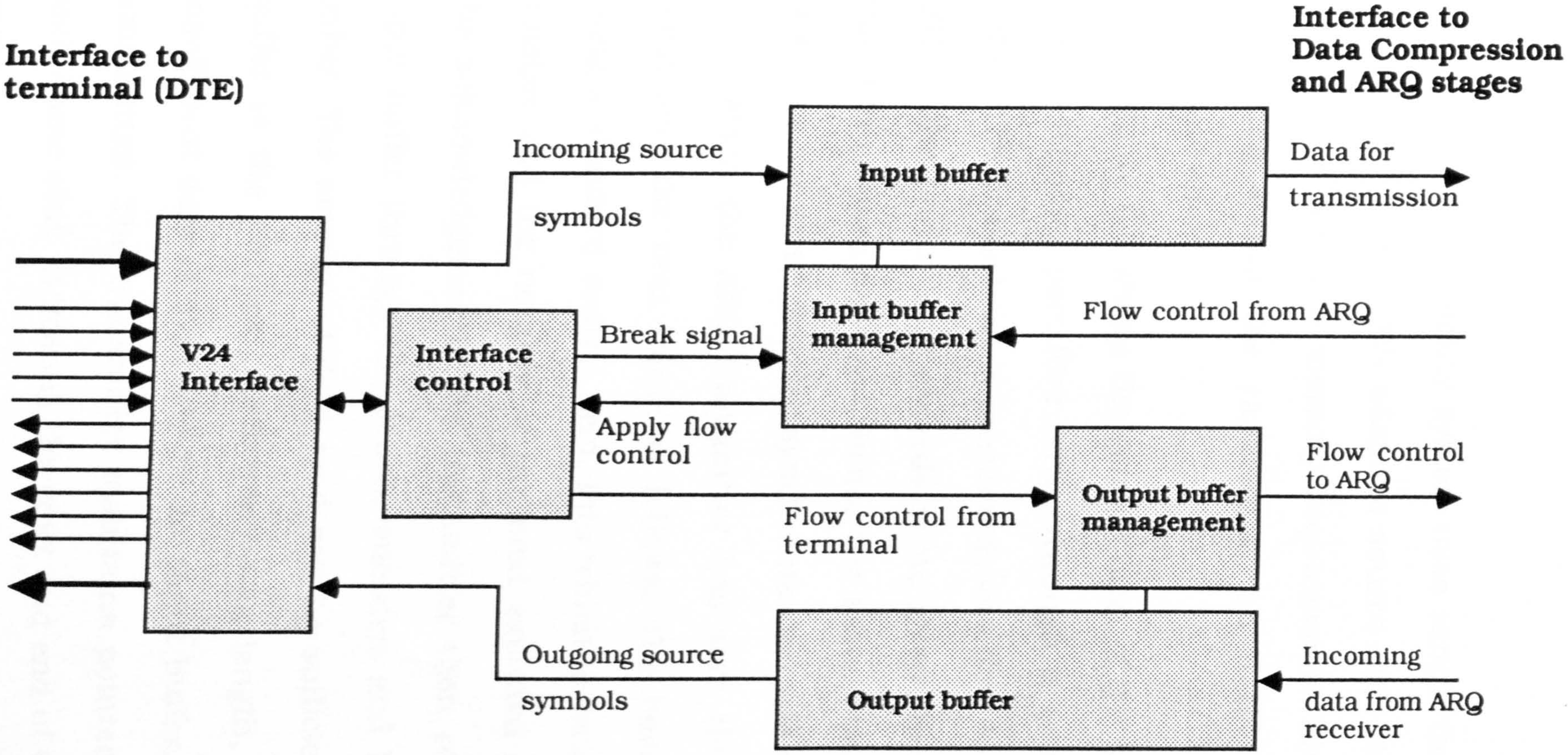

Figure 6.e. General block diagram of the Terminal Interface section of the Error Control Systems

6.5.1 System A.

The low complexity system uses simple Go Back N ARQ, with the 4/8 variable length adaptive source code described in Section 3.3.1. (type (i)). The memory requirements are modest, and the processing within the capacity of an 8 bit microprocessor or microcontroller.

Figure 6.f shows the general form of the system. Data is read from a terminal port (the DTE), through a source encoder into a buffer. The buffer level is monitored, and flow control applied to the DTE if the buffer level exceeds some given threshold. The data is stored in compressed form, hence the assumption is made that loss of source encoder/ decoder synchronization will not occur.

When the ARQ transmitter has sent the current frame, it constructs the next frame as follows. The header is constructed, using a standard control field, into which is entered the next frame number and the number of the next expected received frame (i.e. the acknowledgement). The transmitter then reads data from the input buffer through the source encoder, and hence into a frame buffer. The amount of data read will be sufficient to fill the frame buffer to the current maximum frame length, or may be less if insufficient data is available in the input buffer. The frame is then transmitted. The transmitter maintains pointers to the buffer for each frame sent, indicating the start and end of the data field of the frame.

The source code may be updated at the end of each frame, or after each character is encoded. The choice will depend on the

available processing time.

The receiver accepts frames from the modem, delimited by the synchronization patterns and checks them for transmission errors. If errors are not detected and the frame sequence number matches that expected, the receiver calculates the *next* expected received, and passes the frame contents through the source decoder to the output buffer.

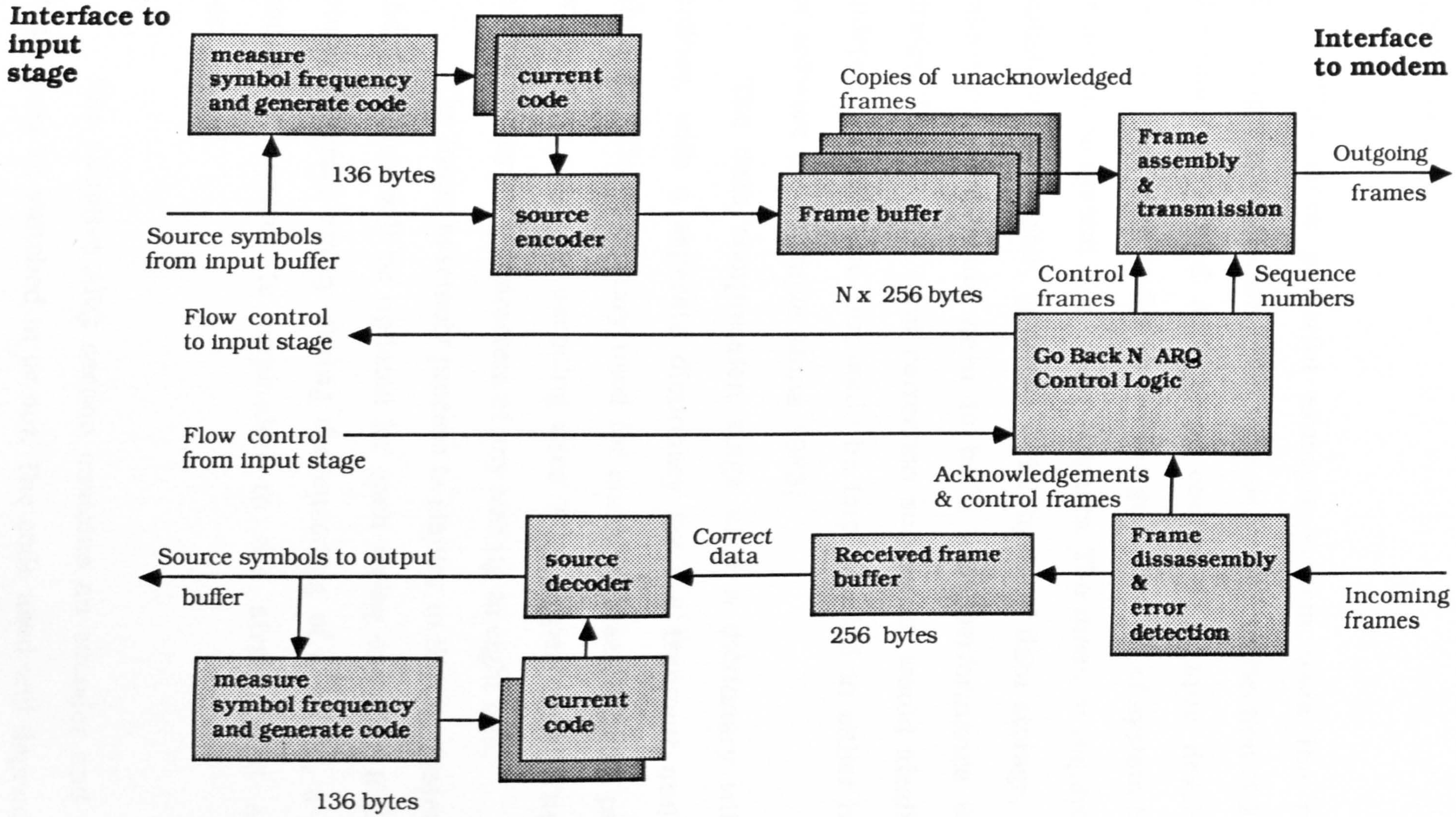

Figure 6.f Data Compression and ARQ section of Error Control System A

6.5.2 System B.

This more powerful example system uses the modified Ziv-Lempel compression technique described in Section 3.3.2., and the adaptive hybrid ARQ error control technique developed in Section 5.7.1. Figure 6.g shows the general form of system B, which is similar to system A in many respects. The memory requirement is substantially greater, typically 32 kilobytes of data storage, and the microprocessor would need to be a high performance 8/16 bit device. The forward error correction subsystem would ideally be an independent element, but could be implemented in either hardware or software (for example Sinha 1983).

The data compression stage uses a dictionary with 2048 entries, with a seperate dictionary for the transmit and receive channels. The dictionary used for encoding uses 5 bytes per entry, whilst that used for decoding uses 7 bytes per entry. The source alphabet may have characters of any size up to eight bits.

The frame assembly process is similar to that in System A, but the dictionary will be updated for each string encoded. If the input buffer becomes empty during the encoding of a string, a *timeout* control character is appended to the string, and encoding terminated.

The adaptive ARQ section contains an encoder and decoder, which may be switched in or out. The code used will depend heavily on the ease with which a decoder can be constructed, and on the

availability of suitable integrated circuit support. The code needs to be suitable for correcting the types of errors discussed in Section 4.3.1., burst errors of the type produced by a modern modem. As the bursts will in general be sparse, a random error correcting code such as the BCH code used for performance comparison in Chapter 5 could be used. A number of integrated circuits suitable for decoding this type of code are being developed (Johnson 1983, Hsu 1984). An interleaved single error correcting code could be used (Section 4.5.4), which would be simple to decode, in software or hardware, although would not achieve the same error correcting potential as the BCH code.

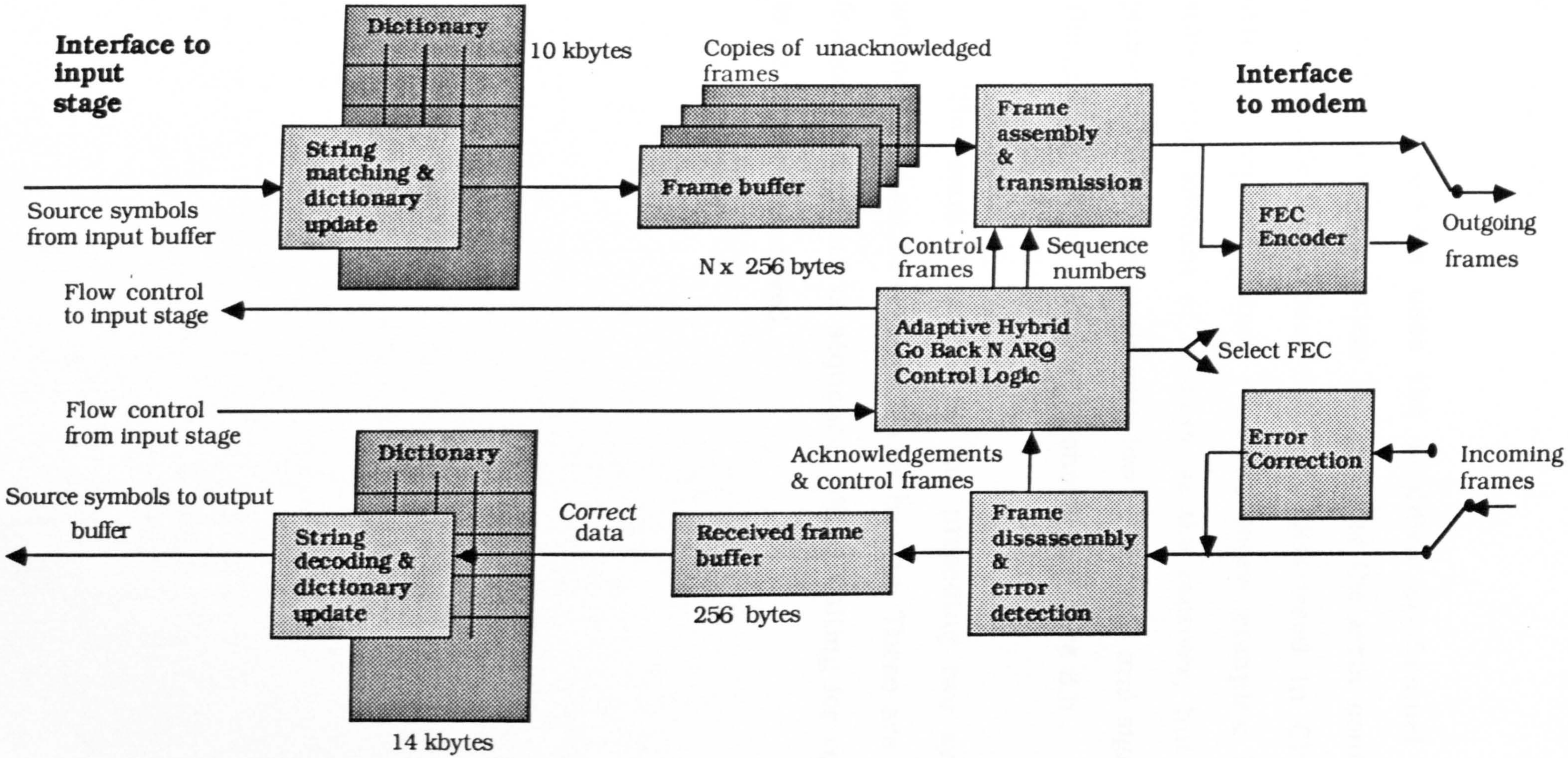

Figure 6.g Data Compression and ARQ section of Error Control System B

6.5.3 System C

This system uses the modified Ziv-Lempel compression algorithm, as with system 'B', however the error control algorithm used is selective repeat ARQ. As discussed in Chapter 5, the selective repeat system requires more complex logic, and a substantial amount of memory at the receiver, but offers better performance on channels with low error rates and significant delay. The general system structure is shown in Figure 6.h .

The main difference to the preceding two systems is the addition of a set of receiver frame buffers. These are used to store frames received out of sequence, whilst waiting for rejected frames to be correctly received.

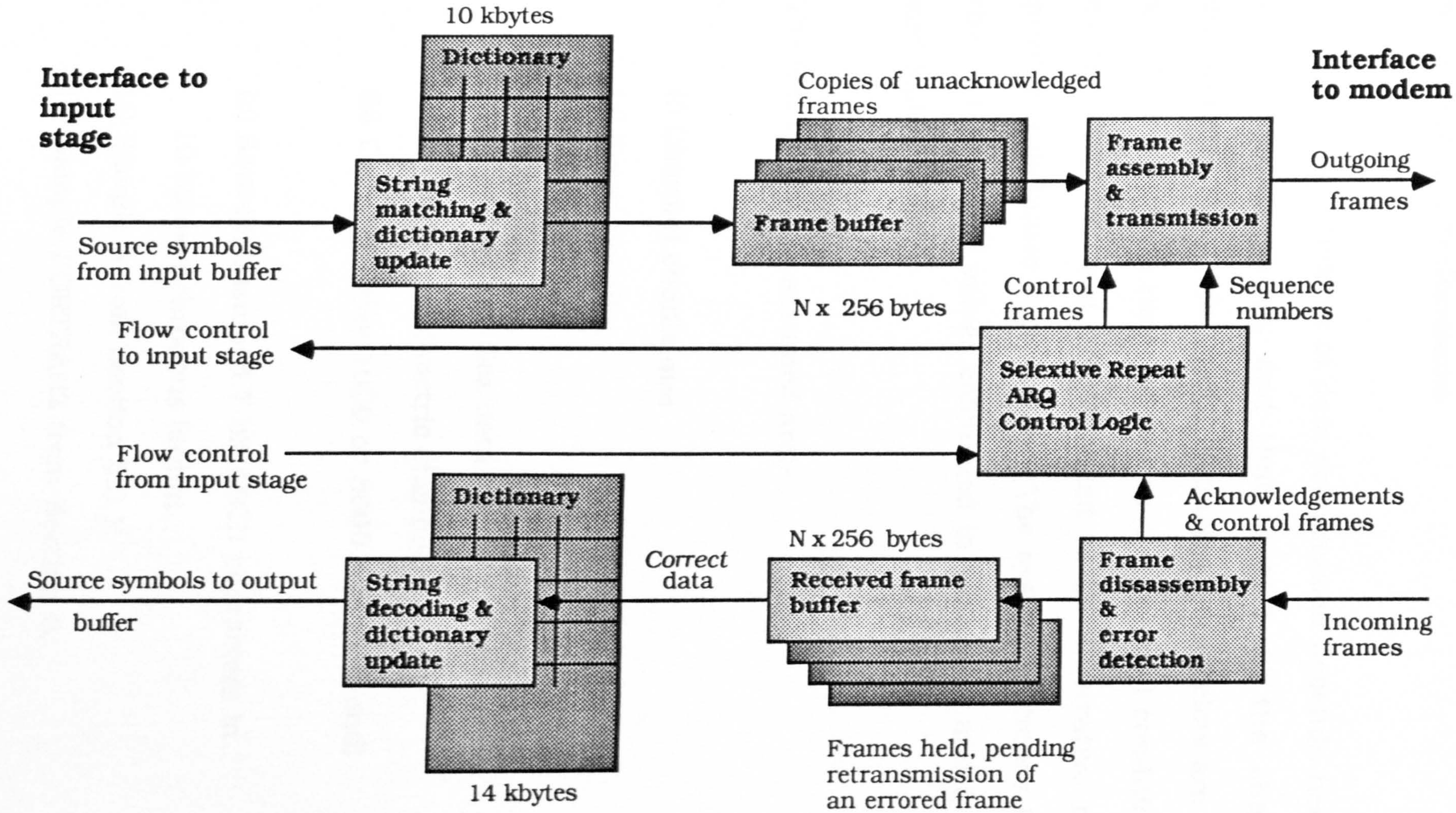

Figure 6.h Data Compression and ARQ section of Error Control System C

6.5.4 Expected Performance

The performance of data compression depends heavily on the source characteristics, and that of ARQ on the channel error distribution. The performance estimates given below are based on a small subset of the range of source and channel conditions used in earlier chapters, and are intended to be illustrative rather than provide a definitive comparison. The system efficiency is based on the corresponding values calculated in sections 3 and 5 (tables 3.q and 5.p).

The test conditions assumed are:-

(i) **Channel conditions**

(a) Error models

-error free

-Lewis and Cox data

-Binary symmetric channel.

(b) Channel delay 1000 or 5000 bits (end-end)

(ii) **Source**, assumed 7 bit ASCII characters in 10 bit asynchronous format.

(a) Sample A from Section 3.3.1.

(b) Sample FORTRAN2 from Section 3.

(iii) **Other assumptions.**

(a) Frame data field 875 bits

(b) Frame header 48 bits

(c) No processing overhead

(d) DTE continually supplies data.

Table 6.j shows that for the sample conditions given, the performance of system B is substantially better than that of system A. The ratio of input data rate to channel data rate, a measure of the gain of the system, is between 30 and 180 percent higher for system B than for system A. System C, employing selective repeat ARQ is slightly less efficient than system B under error conditions, but would be more effective if the channel delay were larger.

An example of the effective transmission speed for the three systems, when with a 2400 bit/s modem, is shown in Table 6.k. This essentially repeats the information in the preceding table, but illustrates the end result.

Source type	Channel type	Efficiency System A	System B	System C
Delay 1000 bits:				
'A'	no errors	1.66	3.86	3.86
'F2'	no errors	1.60	2.09	2.09
'A'	errored	1.34	3.25	3.13
'F2'	errored	1.08	1.76	1.69
Delay 5000 bits:				
'A'	no errors	1.66	3.86	3.86
'F2'	no errors	1.60	2.09	2.09
'A'	errored	0.79	2.84	3.13
'F2'	errored	0.76	1.54	1.70

Table 6.j Ratio of input data rate to channel data rate, for complete error control/ data compression system. Error free and Lewis and Cox error distributions.

Source type	Channel type	Effective transmission speed (kilobits/sec) System A	System B	System C
Delay 1000 bits:				
'A' no errors		4.0	9.3	9.3
'F2' no errors		3.8	5.0	5.0
'A' errored		3.2	7.8	7.5
'F2' errored		2.6	4.2	4.0
Delay 5000 bits:				
'A' no errors		4.0	9.3	9.3
'F2' no errors		3.8	5.0	5.0
'A' errored		1.9	6.8	7.5
'F2' errored		1.8	3.7	4.1

Table 6.k Effective transmission speed of the complete error control /data compression system, for a nominal channel (modem) bit rate of 2.4 kilobits/s. Error free and Lewis and Cox error distributions.

The expected system performance on the binary symmetric channel is shown in figures 6.m(i) and (ii) for source A, and figures 6.n(i) and (ii) for source FORTRAN2.

In Figure 6.m(i) the transmission rate for system A is approximately 4 kilobits per second at low error rates, falling below 2400 bits per second at a bit error rate of approximately 0.0002. Systems B and C have a transmission rate of over 9 kilobits per second at low error rates. System B sustains a throughput of over 8 kilobits for bit error rates of up to 0.01, whilst system C performs poorly at bit error rates in excess of 0.001.

The effects of delay may be seen in Figure 6.m(ii), in which the channel delay has been increased to 5000 bits (approximately two seconds at 2400 bits/s). The performance of selective repeat (system C) is unaffected, whilst systems A and B show some loss in throughput. System B is still the most effective at high error rates, although the throughput in the valley at a bit error rate of 0.0001 has dropped to around 7 kilobits per second.

The performance of systems A to C on the FORTRAN sample, shown in Figure 6.n(i) is not as impressive as the preceding example. Although systems B and C do achieve higher throughput than the simpler system A, at low error rates the improvement is only 25 percent. The results for a delay of 5000 bits shown in Figure 6.n(ii) indicate, as before, a small loss in throughput for systems A and B.

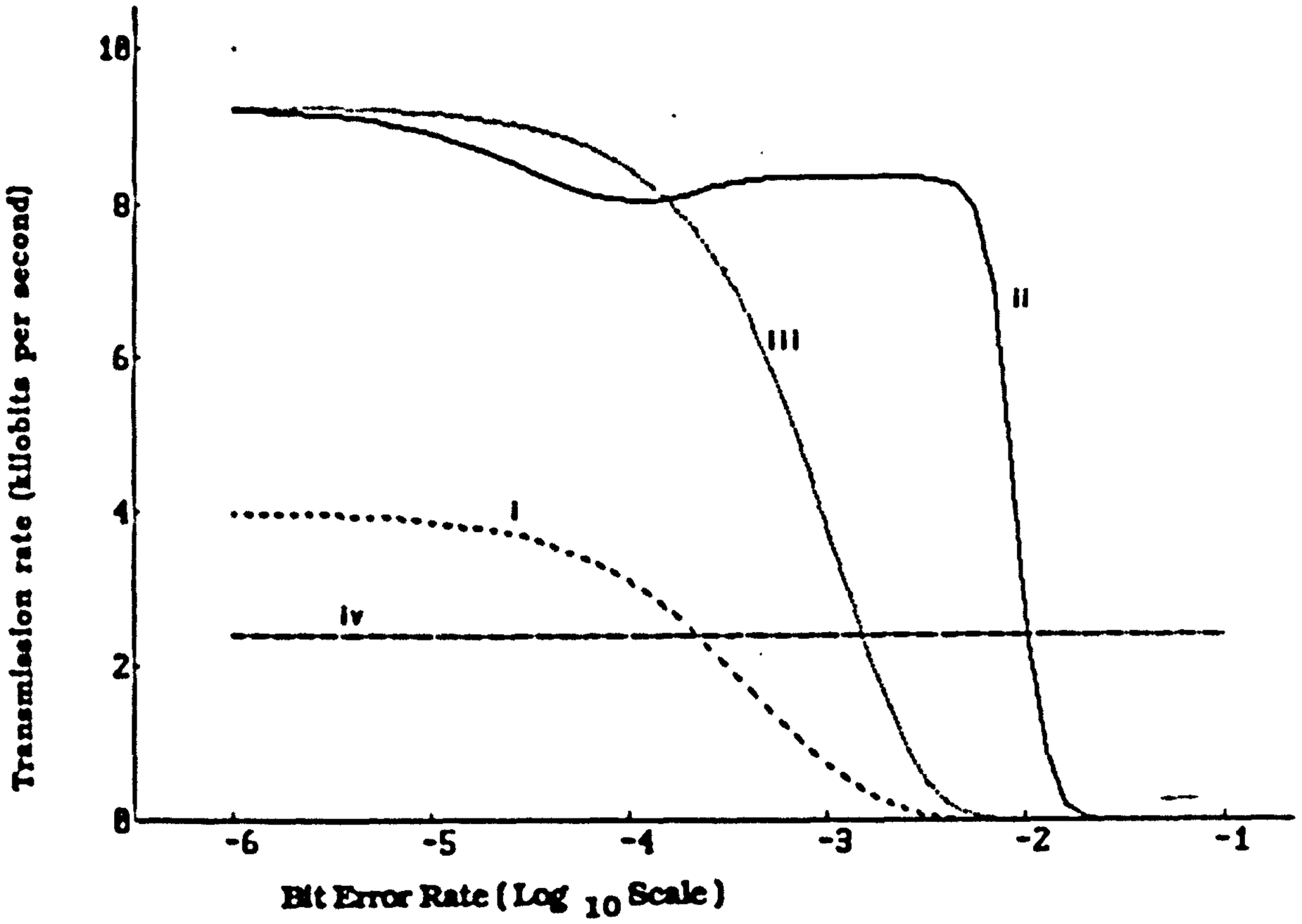

Figure 6.m(i) Effective Transmission Rate of the Error Control Unit over a 2400 bit/s binary symmetric channel, with source "A" and a 1000 bit channel delay

(i) Scheme A
(ii) Scheme B
(iii) Scheme C
(iv) Nominal channel tx rate

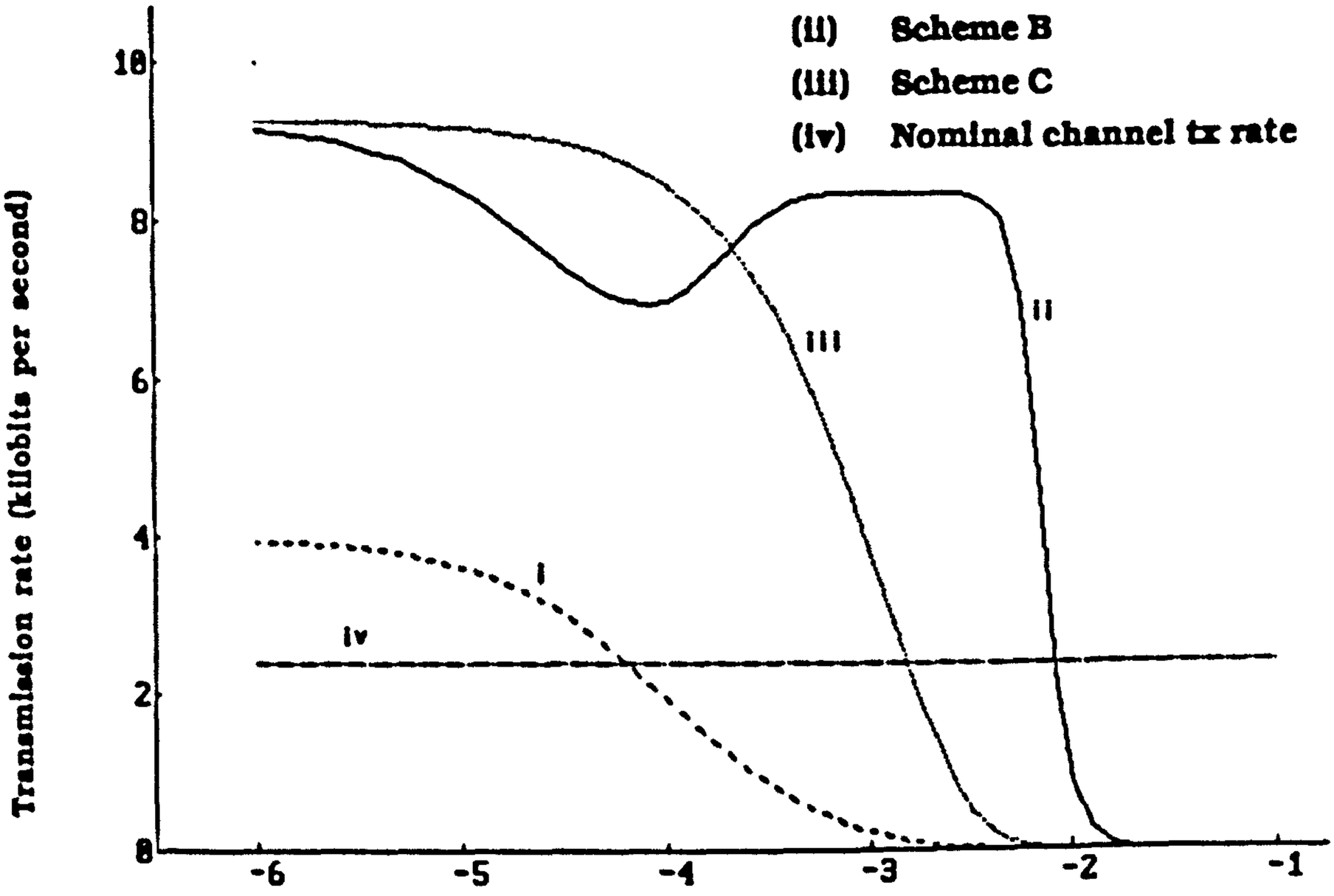

Figure 6.m(ii) Effective Transmission Rate of the Error Control Unit over a 2400 bit/s binary symmetric channel, with source "A" and a 5000 bit channel delay

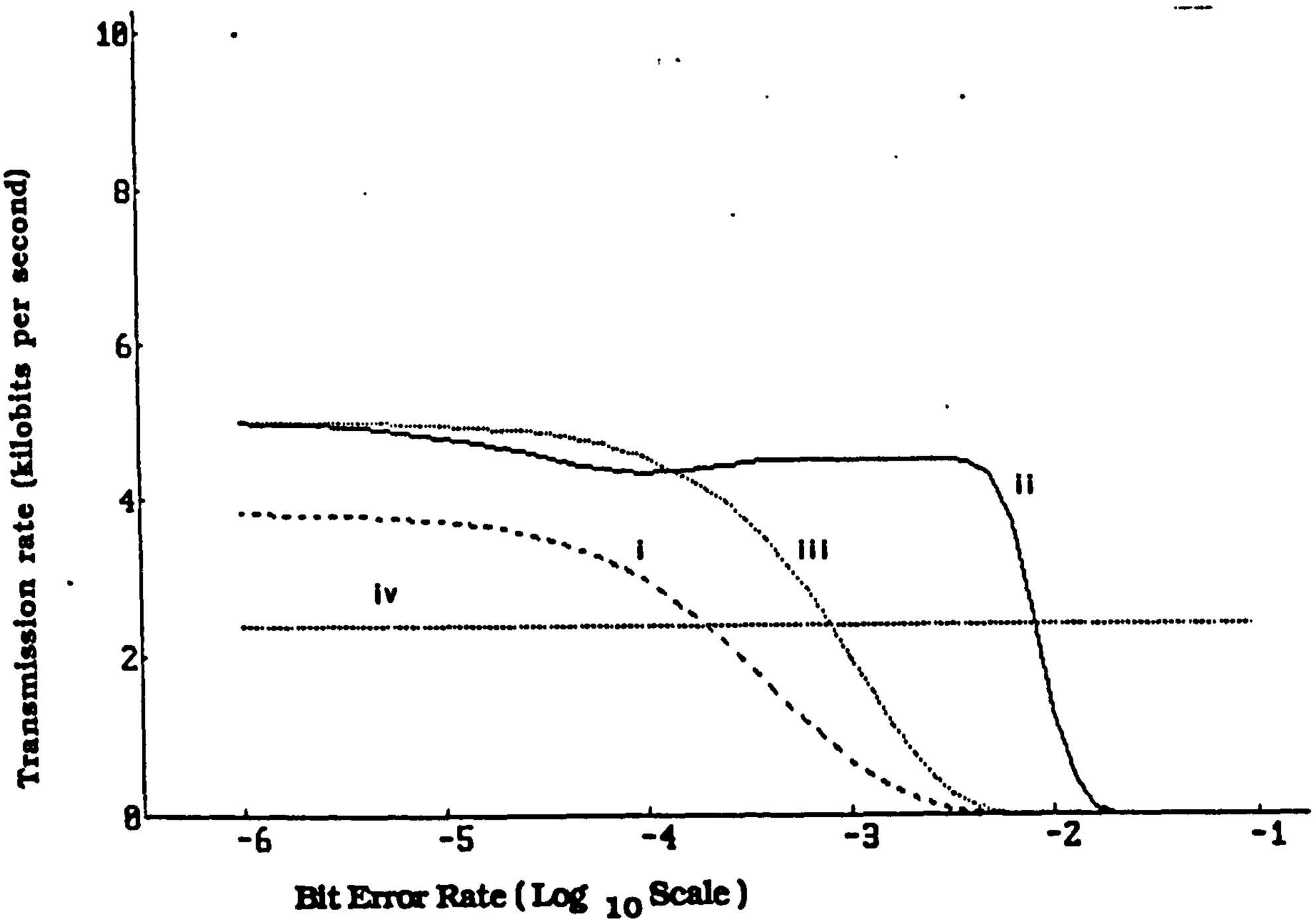

Figure 6.n(i) Effective Transmission Rate of the Error Control Unit over a 2400 bit/s binary symmetric channel, with source "F2" and a 1000 bit channel delay

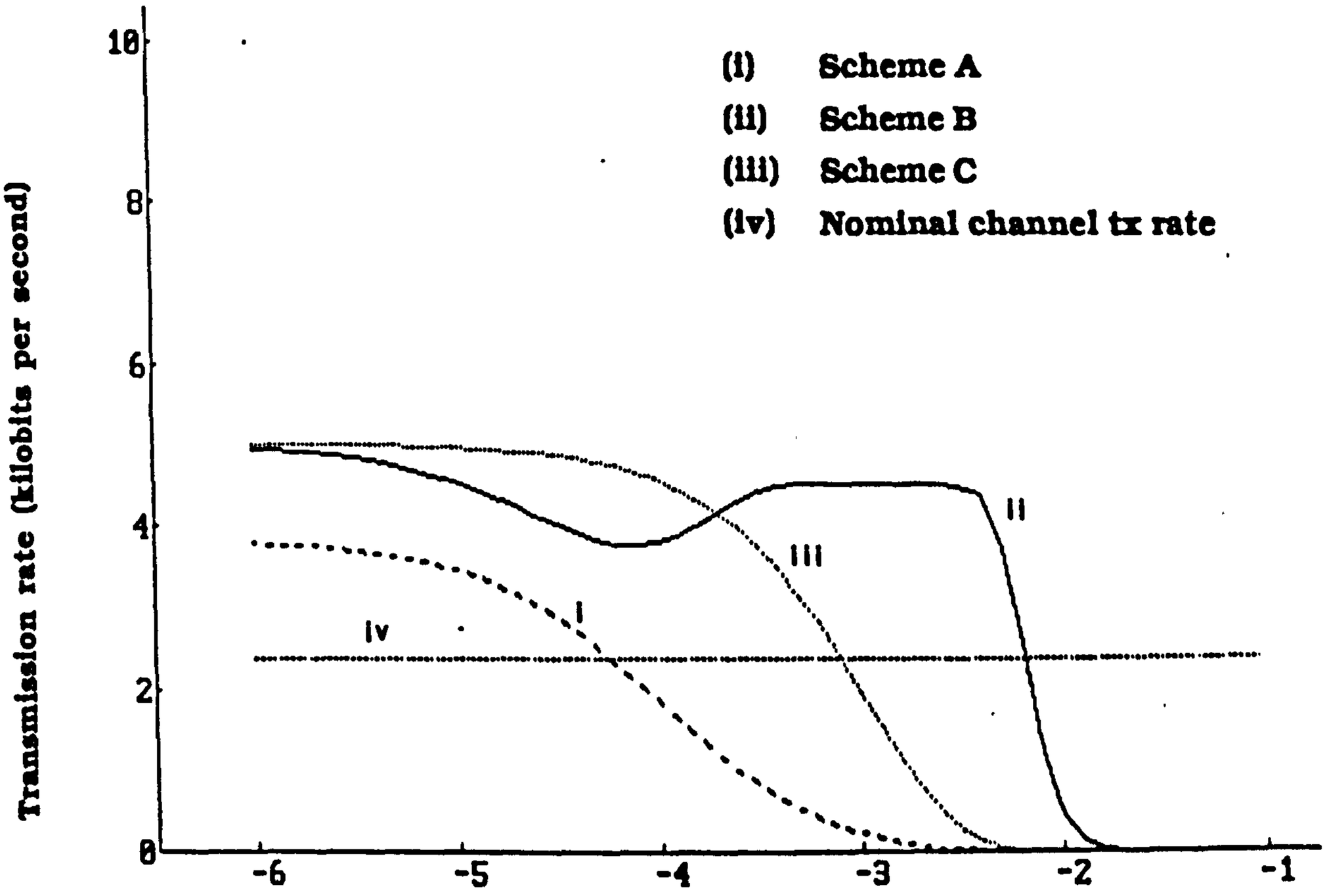

Figure 6.n(ii) Effective Transmission Rate of the Error Control Unit over a 2400 bit/s binary symmetric channel, with source "F2" and a 5000 bit channel delay

In the examples given above, system 'B', with the modified Ziv-Lempel compression algorithm and adaptive hybrid ARQ scheme, performs well. The FORTRAN sample used as one of the data sources was selected on the basis that it resulted in poor performance of the Ziv-Lempel compression algorithm in the tests of Chapter 3. It is expected that the system throughput would be generally between the results of Figure 6.m and 6.n, and could well be higher.

6.6 Summary and Discussion

This chapter has examined some of the issues involved in combining the data compression and error control techniques identified in Chapters 3 and 5 to form a system. The Open Systems Interconnection concept was reviewed, and the error control system defined in terms of this established protocol model. This was followed by a discussion of frame synchronization, flow control, signalling, and link establishment.

Section 6.3 considered the problems involved in incorporating adaptive data compression into the error control system; the possibility of loss of source decoder synchronization must be considered carefully.

The need for verification of communications protocols was briefly discussed in Section 6.4. Whilst not within the scope of this study, this subject forms a vital part of the protocol design process.

Three complete systems were described, and their performance compared. The most effective appears to be that employing the modified Ziv-Lempel compression algorithm and the adaptive hybrid ARQ error control scheme. The performance of this system under the test conditions indicates that the effective transmission rate of the terminal may be up to four times the modem transmission rate. The compression achieved by the Ziv-Lempel algorithm is not limited in the way that a Huffman code would be, and the effective system performance may well be far better than the results would indicate.

In addition to the transmission efficiency of the systems, there are other more subjective criteria by which they may be compared. If used in an interactive terminal application, the efficiency of the system would not be as noticeable to the user as the delay, or the irregularity in the data flow. For example, if text appears on the terminal screen at high speed but in a jerky fashion, this is likely to be more disturbing to the user than a slower but smoother flow of data. Barber and Lucas (1983) discuss the effects of system response time on terminal operators, but do not consider the effects described above.

Within the present study, the effects of errors on the data flow have not been considered, the main performance parameter used was transmission efficiency. Hybrid ARQ may be less efficient than ARQ under conditions of low channel error rate, however this is due to the overhead imposed by the additional parity bits, not generally to retransmissions. The data flow produced by an ARQ system will be more irregular, as each retransmission may result in a temporary halt in output to the terminal. One possible measure of the irregularity of the data flow is the variance of the overall system delay. This could be calculated directly for properly defined channel models, but should be supported by both objective and subjective tests using real or simulated systems.

7. CONCLUSIONS.

The objectives of the research project were to investigate the performance of data compression and error control techniques for use in an error control unit. The error control unit will be used to enhance the performance of a data communications link, established using high speed modems over the Public Switched Telephone Network. Throughout the research project, a 'realistic' approach was taken when considering performance and implementation issues. The data compression algorithms discussed were adaptive, and required little prior knowledge of the data source. The error control techniques were selected to give reasonable performance under a wide range of channel conditions. The practical problems of implementation were discussed at some length, for both the individual techniques and the overall system.

Two data compression algorithms were given, one a very simple form of the Huffman code, and the other a modification of the Ziv-Lempel algorithm. The simple scheme offers reasonable compression, and is of low complexity, whilst the second algorithm provides very good performance but at the expense of complexity. The modification resulted in a reduction in the memory requirement of the Ziv-Lempel algorithm, with negligible effect on performance.

The telephone channel is not sufficiently well known to enable one error control technique to be classed as optimum. A wide range of channel conditions were used to compare a number of different ARQ and hybrid ARQ error control schemes. Three

techniques were selected as candidates for the error control system of which two are existing ARQ protocols. The adaptive hybrid ARQ technique described in Chapter 5, gave excellent performance under both low and high error rate conditions. The choice of error control scheme will ultimately depend on the permissible complexity, however adaptive schemes seem particularly applicable to use on the telephone channel due to the non-stationary nature of the error distribution.

System design considerations were discussed in Chapter 6, which was intended to place the data compression and error control schemes in context. Three system configurations were given, and their complexity and performance compared. That employing the modified Ziv-Lempel data compression algorithm and the adaptive hybrid ARQ scheme provided excellent performance, although Selective Repeat ARQ may be a better choice for channels with long delay.

The project achieved the objectives; if, however, any one area of the work could have been extended further, the implementation of the forward error correcting code element of the adaptive hybrid ARQ scheme would have been considered in more detail. Realistically, the encoder and decoder should be implemented in software, possibly with a little hardware support, which presents interesting problems when the transmission speed is considered.

8. Further Work.

(i) Compression of synchronous data.

Noiseless data compression schemes in general require that the data stream is split into identifiable symbols. On synchronous data streams however, the symbol boundaries are not defined, and in packet networks the symbol sizes may vary from packet to packet. In Section 3.3.3, it was shown that compression may still be achieved if arbitrary symbol sizes are assumed, although the assertion was not well supported by test data. There are many potential applications for data compression schemes which can operate under these conditions.

(ii) Improvements to the Ziv Lempel algorithm.

In Chapter 3 the Ziv Lempel compression algorithm was discussed, and a number of improvements proposed. The resulting modified algorithm used a relatively small amount of memory but achieved excellent compression. It was also shown that reasonable compression could be obtained with a small dictionary, of as little as 512 entries. The improvement resulting from simple modifications to the basic algorithm suggests that further study of the algorithm should produce valuable results. Areas of particular interest are dictionary maintenance strategies, increasing the speed of adaption to a source, and improving tolerance to transmission errors.

(iii) Integration of ARQ and data compression

A fundamental problem with integrated ARQ / adaptive

data compression systems was discussed in Section 6.3.2., relating to the point at which the data compression is applied. If compression is applied immediately the data enters the system, loss of source encoder/decoder synchronization would result in the data being effectively lost. If compression is applied immediately prior to transmission, the source encoder may only be updated using information from acknowledged frames, which will slow the adaption process. Another alternative would be to store the data in compressed form, but to download the dictionary or code table to the decoder if synchronization is lost. This would take a considerable time for the Ziv Lempel source encoder, due to the size of the dictionary. This general problem needs further investigation, as it affects both the reliability and efficiency of the system.

(iv) ARQ protocol reliability.

In the immediate future, high speed modems will provide a valuable service for point-to-point links and gateways to packet networks. The SDLC/ HDLC/ X25/ X32 family of protocols is already widely used for public data networks and private wide area networks, despite some uncertainty concerning the reliability of the error detection methods used. A study of the error distribution produced by modern synchronous modems, for example those using V22, V26, V29, and V32/33, could form the basis of a detailed analysis of the error detection properties of the HDLC class of link level protocol, extending the work of Funk (1982) ,and others.

(v) Improvements to the adaptive hybrid ARQ scheme

The adaptive hybrid ARQ scheme developed in Section 5.7 has good performance under the test conditions applied. The method needs further development however, in several areas. The forward error correcting code used in the analysis of Section 5.7 was a shortened (1023,923) BCH code, however the comments made in Section (i) below, relating to decoder implementation, should be considered. A more extensive performance comparison should be made under a wider range of channel conditions, however this should be based on recorded channel error data if possible.

(vi) Adaptive frame length allocation

Section 5.5 briefly discussed the question of optimum frame length, and also a potential problem that may occur on channels with extended noisy periods (for example, either the telephone channel or a fading channel). If during the error free period, the ARQ transmitter is sending long frames (which are more efficient under these conditions), and the channel then switches to a noisy state, the transmitter will retransmit the original frames. The long frames will have a low probability of successful transmission, and hence the ARQ system is effectively locked up until the errors cease. The solution proposed was to retransmit the data rather than the frames, which implies that the retransmitted frames may be shorter than the original ones, and hence the probability of successful transmission improved. This is more likely to be useful on a fading channel than a simple burst error channel, due to the delays involved but is worthy of further consideration for the telephone

channel.

(vii) Subjective measures of ARQ system performance

In Section 6.6 a question was raised relating to the usefulness of transmission efficiency as a performance measure in interactive (terminal/computer) applications. The suggestion was made that the mean and the variance of the delay would be of more immediate relevance to the user, as these influence their perception of the operation of the system and could prove distracting. Hybrid ARQ schemes offer the promise of a reduced number of retransmissions and hence a more even data flow. This should be investigated further, but should be supported by experiments involving terminal users.

(viii) Alternative hybrid ARQ systems.

The assumption was made in Chapter 5 that the error correcting codeword consisted of the ARQ frame with additional parity bits. An alternative method would be to use an independent error correcting encoder/ decoder, i.e. one in which the codewords are not synchronized to the ARQ frames. This would simplify the design of the decoder but would render certain types of hybrid ARQ scheme difficult to implement (namely the adaptive scheme and the type II hybrid). The design of a variable rate encoder/ decoder able to support this type of operation would be an interesting area for development.

(ix) Software implementation of error correcting codes.

One area of practical difficulty encountered in implementing the results of this study, centred on the software implementation of error control codes. Most of the established methods are ideally suited to hardware implementation, but do not lend themselves readily to the software approach. Whilst advances in programmable, semi- and full-custom integrated circuits are being made, several fundamental problems still exist, namely the price/volume ratio and the relatively high cost of memory in terms of gates per cell. For many medium volume applications, the use of single chip microcomputers or microcontrollers is preferred, giving greater flexibility and moderate cost, and permitting processor to be shared between the error correction function and other control functions. For many data communications systems, the development of error detecting and correcting codes suited to software implementation would be of considerable advantage. Such codes do exist, but the number of hardware implementable techniques is far greater.

Appendix A References.

Abramson N. — Information Theory and Coding. McGraw Hill 1963

Agnostou M.E. Sykas E.D. Protonotarius E.N. — Steady-State and Transient Delay Analysis of ARQ Protocols Computer Commun 7,1 Feb 1984 pp 23-30

Alexander A.A. Gryb R.M. Nast D.W. — Capabilities of the Telephone Network for Data Transmission B.S.T.J. Vol 39,3 May 1960 pp 431-476

Balcovic M.D. Klancer H.W. Klare S.W. McGruther W.G. — High Speed Voiceband Data Transmission Performance on the Switched Telecommunications Network B.S.T.J. Vol 50,4 Apr 1971 pp 1349-1384

Barber R.E. Lucas H.C. — System Response Time, Operator Productivity and Job Satisfaction CACM 26,11 Nov 1983 pp 972-986

Barnard G.A. — Statistical Calculation of Word Entropies for Four Western Languages IRE Trans IT-1, 1955, pp 49-53

Bartlett M.S. — An Introduction to Stochastic Processes. Cambridge Univ. Press 1978

Batorsky D.V. Burke M.E. — 1980 Bell System Noise Survey of the Loop Plant B.S.T.J. ,63,5 May 1984 pp 775-818

Bell Labs (A.T. & T.)	Data Communications Using the Switched Telecommunications Network Bell System Technical Reference 41005 May 1971
Benice R.J. Frey A.H.	An Analysis of Retransmission Systems IEEE Trans COM-12,4 December 1964 pp135-145
Benice R.J. Frey A.H.	Comparisons of Error Control Techniques IEEE Trans COM-12,4 December 1964 pp 146-154
Berger J.M. Mandelbrot B.	A New Model for Error Clustering in Telephone Circuits IBM J. Res. Dev. July 1963 pp 224-236
Bjorner D.	Finite State Automaton Definition of Data Communication Line Control Procedures. AFIPS Proc. 37 1970 pp 477-491
Blahut R.E.	A Universal Reed-Solomon Decoder IBM J. Res. Dev. Mar 1984 pp 150-158
Blank H.A. Trafton P.J.	A Markov Error Channel Model Proc Nat Telecomm Conference 1973 Vol 1 pp 15B/1-8
Bochmann G.V.	Finite State Description of Communication Protocols. Computer Networks 2, 1978 pp 361-372
Bose R.C. Ray-Chaudhuri D.K.	On a Class of Error Correcting Binary Group Codes Inf Control 3, March 1960, pp 68-79

Brayer K.	Error Control Techniques Using Binary Symbol Burst Codes IEEE Trans COM-16, Apr 1968 pp 199-214
Brownlie J.D., Cusack E.L.	Duplex transmission at 4800 and 9600 bit/s on the PSTN and the use of channel coding with a partitioned signal constellation Br. Telecom Tech. J. Vol 2, No 4 Sept 1984, pp 64-73
Burton H.O. Sullivan D.D.	Errors and Error Control Proc. IEEE Nov 1972 pp 1293-1300
Bux W. Kummerle K. Truong H.L	Balanced HDLC Procedures: A Performance Analysis IEEE Trans COM-28,11 Nov 1980 pp 1889-1898
Bylanski P., Ingram D.G.W.	Digital Transmission Systems. Peter Peregrinus 1980
Cain J.B. Simpson R.S.	The Distribution of Burst Lengths on a Gilbert Channel IEEE Trans IT-15 Sep 1969 pp 624-627
Carey M.B.,Chen H.T., Descloux A. Ingle J.F., Park K.I.	1982/83 End Office Connection Study: Analog Voice and Voiceband Data Transmission Performance Characterization Switched Network B.S.T.J. Vol 63,9 Nov 1984 pp 2059-2119
CCITT	Red Book Recommendations -V22, V32, V41, X25, T.6 Published ITU, 1984

Chu W.W.
Optimal Message Block Size for Computer Communications with Error Detection and Retransmission Strategies.
IEEE Trans COM-22,10 Oct 1974
pp 1516-1525

Clark A.P.
Advanced Data Transmission Systems
Pentech press 1977

Cleary J.G.,
Witten I.H.
Data Compression using Adaptive Coding and Partial String Matching
IEEE Trans. COM-32,4 April 1984
pp 396-402

Cooper D.,Lynch M.F.
Text Compression Using Variable to Fixed Length Encodings
J. American Society for Inf.Science
Jan 1982, pp 18-31

Cooper L.,
Cooper M.W.
Introduction to Dynamic Programming
Pergamon 1981

Cox D.R.
Renewal Theory
Methuen 1962

Davies D.W.
Computer Networks and their Protocols
Wiley 1979

Deo N.
Graph Theory with Applications to Engineering and Computer Science.
Prentice Hall 1974

Diaz M.
Modelling and Analysis of Communication and Cooperation Protocols using Petri Net Based Models
Computer Networks 6, 1982 pp 419-441

Djikstra E.W.
Cooperating Sequential Processes
In:- Programming Languages
ed. F.Genuys
Academic Press 1968

Drajic D.
Vucetic B.
Evaluation of Hybrid Error Control Systems
IEE Proc F Vol 131,2 Apr 1984
pp 181-193

Drukarev A.
Costello D.J.
A Comparison of Block and Convolutional Codes in ARQ Error Control Schemes
IEEE Trans COM-30,11 Nov 1982
pp 2449-2455

Drukarev A.
Costello D.J.
Hybrid ARQ Error Control Using Sequential Decoding
IEEE Trans IT-29,4 Jul 1983 pp 521-535

Duffy F.P.,
Thatcher T.W.
Analog Transmission Performance on the Switched Telecommunications Network
B.S.T.J. Vol 50,4 Apr 1971 pp 1311-1347

Easton M.C.
Batch Throughput Efficiency of ADCCP/ HDLC/SDLC Selective Reject Protocols
IEEE Trans COM-28,2 Feb 1980
pp 187-195

Elliott E.O.
Estimates of Error Rates for Codes on Burst Noise Channels
B.S.T.J. Vol 42, Sep 1963 pp 1977-97

Elliott E.O.
A Model of the Switched Telephone Network for Data Communications
B.S.T.J. Vol 44,1 Jan 1965 pp 89-109

Enticknap R.G.
Errors in Data Transmission Systems
IRE Trans Comm Systems March 1961 pp 15-20

Faller N.
An Adaptive System for Data Compression
7th Asilomar Conf. on Circuits, Systems and Communications 1974 pp 593-597

Fano R.M.
A Theory of Impulse Noise in Telephone Networks
IEEE Trans COM-25,6 Jun 1977 pp 577-588

Farrell P.G.
Coding for Noisy Data Links
Ph.D. Dissertation
University of Cambridge 1969

Ferguson T.J.,
Rabinowitz J.H.
Self-Synchronizing Huffman Codes
IEEE Trans IT-30 July 1984 pp 687-693

Field J.A.
Efficient Computer-Computer Communication
Proc IEE, 123,8 August 1976 pp 756-760

Fleming H.C.
Hutchinson R.M.
Low-Speed Data Transmission Performance on the Switched Telecommunications Network
B.S.T.J. Vol 50,4 Apr 1971 pp 1385-1405

Fontaine A.B. Gallager R.G.	Error Statistics and Coding for Binary Transmission over Telephone Circuits Proc IRE vol 49 Jun 1961 pp 1059-1065
Fujiwara C., Kasahara M., Yamashita K., Namekawa T.	Evaluations of Error Control Techniques in Both Independent Error and Dependent Error Channels. IEEE Trans COM-26,6 Jun 1978 pp 785-793
Gallager R.G.	Information Theory and Reliable Communication John Wiley & Sons, 1968
Gallager R.G.	Variations on a Theme by Huffman IEEE Trans IT-24 1978 pp 668-674
Garten H.	On Variable Length Codes under Hardware Constraints IEEE Trans. COM-33,5 May 1985 pp 491-494
Gilbert E.N.	Capacity of a Burst Noise Channel B.S.T.J. Vol 39, Sep 1960 pp 1253-1265
Gilbert E.N.	Codes Based on Inaccurate Source Probabilities IEEE IT-17,3 May 1971 pp 304-314
Goodman R.M.F., Farrell P.G.	Data Transmission with Variable Redundancy Error Control over a High Frequency Channel Proc IEE Vol 122,2 Feb 1975 pp 113-118

Gottlieb D.,Hagerth S.A., Lehot P.G.H Rabinowitz H.S.
A Classification of Compression Methods and their Usefulness for a Large Data Processing Center
AFIPS Joint Computer Conf.
May 1975, pp 453-458

Hailpern B.
A Simple Protocol Whose Proof Isn't
IEEE Trans COM-33 1985 pp 330-337

Hamming R.W.
Error Detecting and Correcting Codes
B.S.T.J. 29, April 1950, pp 147-160

Hamming R.W.
Coding and Information Theory
Prentice Hall 1980

Harary F.
Graph Theory
Addison Wesley 1969

Hardy.G.H. Wright E.M.
An Introduction to the Theory of Numbers
Oxford University Press 1984

Hartley R.V.L.
Transmission of Information
Bell System Tech. J.
July 1928 pp 535-563

Hayes J.F.
Modeling and Analysis of Computer Communications Networks
Plenum Press 1984

Hoare C.A.
An Axiomatic Basis for Computer Programming
CACM 12, 1969 pp 576-580

Hocquenghem A.
Codes Corecteurs D'Erreurs
Chiffres 2, 1959, pp147-156

Hsu I., Reed I.S., Truong T.K., Wang K., Yeh C., Deutsch L.J.	The VLSI Implementation of a Reed-Solomon Encoder Using Berlekamp's Bit Serial Multiplier Algorithm IEEE Trans C-33,10 1984, pp906-911
Huffman D.A.	A Method for the Construction of Minimum Redundancy Codes Proc IRE Vol 40,9 1952 pp1098-1101.
Humblet P.A.	Generalization of Huffman Coding to Minimize Probability of Buffer Overflow IEEE Trans. IT-27,2 March 1981 pp230-232
Johnson B.L.	Design and Hardware Implementation of a Versatile Transform Decoder for Reed-Solomon Codes IEE Conf. VLSI for Communications, London, Dec 1983, pp 53-61
Karp R.M.	Minimum Redundancy Coding for the Discret Noiseless Channel IRE Trans IT-7 Jan 1961 pp27-38
Katona G.O., Nemetz T.O.H	Huffman Codes and Self Information. IEEE Trans IT-22,3 1976 pp337-340
Kitces S. Heller R. Bowen J.	A Technique for Improving the Transmission Reliability of a Fading Channel IEEE Int Convention Record 1963 Vol II,8 pp154-162

Klove T., Miller M.	The Detection of Errors After Error-Correction Decoding IEEE Trans COM-32,5 May 1984 pp 511-517
Knuth. D.	The Art of Computer Programming. Vols 1 and 3. Addison Wesley
Konheim A.G.	A Queueing Analysis of Two ARQ Protocols IEEE Trans COM 28,7 Jul 1984 pp 1004-1014
Labetoulle J. Pujolle G.	HDLC Throughput and Response Time for Bidirectional Data Flow with Nonuniform Frame Sizes IEEE Trans C-30,6 June 1981 pp 405-413
Lai W.S.	An Analysis of Piggybacking in Packet Networks Computer Networks 6 1982 pp 279-290
Langdon G.G.	An Introduction to Arithmetic Coding. IBM J. Res. Develop. Vol 28,2 March 1984 pp135-149
Lavelle P.J.	An Efficient Method for File Transmission. IEEE 1981 Communications Conf. vol 3, pp 53.4.1-53.4.4
Leung C.S.K., Lam A.	Forward Error Correction for an ARQ Scheme IEEE Trans COM-29,10 Oct 1981 pp 1514-1519

Lempel A., Ziv J.
On the Complexity of Finite Sequences.
IEEE Trans. IT-22,1 Jan 1976
pp75-81

Lewis P.A.W.
Cox D.R.
A Statistical Analysis of Telephone
Circuit Error Data
IEEE Trans COM-14,4 Aug 1966
pp 382-389

Lin S.
Costello D.J.
Error Control Coding: Fundamentals and
Applications
Prentice Hall 1983

Lin S., Yu P.S.
An Efficient Error Control Scheme for
Satellite Communications
IEEE Trans COM-28 Mar 1980
pp 395-401

Lin S., Yu P.S.
A Hybrid ARQ Scheme with Parity
Retransmission for Error Control of
Satellite Channels
Nat Telecom. Conf. 1981, Vol 4,
pp G10.3/1-8

Lucky R.W. Saltz J.
Weldon E.J.
Principles of Data Communication
McGraw Hill 1968

Maguire T.A.
Wright E.P.G.
The Examination of Error Distributions
for the Evaluation of Error Detection
and Error Correction Procedures
IRE Trans CS-9 Jun 1961 pp 101-106

Mandelbaum D.M.
An Adaptive-Feedback Coding Scheme
Using Incremental Redundancy
IEEE Trans IT-20,3 May 1974
pp 388-389

Manfrino R.L.	Printed Portugese (Brazilian) Entropy Statistical Calculation Int Symposium of Inf. Theory, Jan 1969.
Maxemchuk.N.F. Stuller.J.A.	Reduction of Transmission Error Propagation in Adaptively Predicted DPCM Encoded Pictures. B.S.T.J. 58,6/2, Jul 1979, pp 1413-23
Maxted J.C. Robinson J.P.	Error Recovery for Variable Length Codes IEEE Trans IT-31,6 Nov 1985 pp 794-801
Mertz P.	Model of Impulsive Noise for Data Transmission IRE Trans CS-9 June 1961 pp130-137
Mertz P.	Statistics of Hyperbolic Error Distributions in Data Transmission IRE Trans CS-9 Dec 1961 pp 377-382
Metzner J.J.	Improvements in Block-Retransmission Schemes IEEE Trans COM-27,2 Feb 1979 pp 524-532
Miller M.J.	Hybrid ARQ Systems for Data Networks IREECON Int. Conf. 1983 pp 44-46
Miller V.S., Wegman M.N.	Variations on a Theme by Lempel and Ziv. Draft, December 1982 IBM Thomas Watson Res. Center.
Milner R.	A Calculus of Communicating Systems Springer Verlag LNCS 92, 1980

Morris J.M.

Optimal Blocklengths for ARQ Error Control Schemes
IEEE Trans COM-27,2 Feb 1979
pp 488-493

Muntner M.
Wolf J.K.

Predicted Performance of Error-Control Techniques over Real Channels
IEEE Trans IT-14,5 Sep 1968
pp 640-650

Reiffer B. Schmidt W.G.
Yudkin H.L.

The Design of an Error Free Data Transmission System for Telephone Circuits
Trans AIEE (Comm & Electronics) 80,
July 1961 pp 224-231

Reiger S.H.

Codes for the Correction of Clustered Errors
IRE Trans IT-6 Mar 1960 pp16-21

Reiser M.

A Queueing Network Analysis of Computer Communication Networks with Window Flow Control
IEEE Trans COM-27,8 Aug 1979
pp1199-1209

Ridout I.B.
Harvie I.B.

The Principles of Scramblers and Descramblers Designed for Data Transmission Systems
British Telecommunications Eng. Vol 1
July 1982 pp 111-114

Rocher E.Y.
Pickholtz R.L.
An Analysis of the Effectiveness of Hybrid Transmission Schemes.
IBM J. Res. Dev. 14, July 1970 pp426-433

Sastry A.R.K.
Kanal L.N.
Hybrid Error Control Using Retransmission and Generalized Burst-Trapping Codes
IEEE Trans COM-24,4 Apr 1976 pp 385-393

Schreiber W.F.
The Measurement of Third Order Probability Distributions of Television Signals
IRE Trans IT-2,3 1956 pp 94-105

Schwartz E.S.
Kallick B.
Generating a Canonical Prefix coding
CACM 7,3 March 1964 pp166-169

*

Shannon C.E.
A Mathematical Theory of Communication.
Bell System Tech. J.
Vol 27 1948 pp 379-423
Vol 27 1948 pp 623-656

Shannon C.E.
Prediction and Entropy of Printed English
Bell System Tech. J.
Vol 30,1 Jan 1951 pp 50-64

Smith S.A.
A Generalization of Huffman Coding for Messages with Relative Frequencies given by Upper and Lower Bounds.
IEEE Trans. IT-20 1974 pp124-125

Stiffler J.J.
Theory of Synchronous Communication.
Prentice Hall 1971

Stuck B.W. Kleiner B.	A Statistical Analysis of Telephone Noise B.S.T.J. Vol 53,7 Sep 1974 pp 1263-1320
Sussman S.M.	Analysis of the Pareto Model for Error Statistics on Telephone Circuits IEEE Trans COM-11 Jun 1963 pp 213-221
Tan C.P.	On the Entropy of the Malay Language IEEE Trans IT-27,3 1981 pp383-4
Tarjan R.	Depth First Search and Linear Graph Algorithms SIAM J Comp. 1,2 Jun 1972, pp 146-160
Townsend R.L. Watts R.N.	Effectiveness of Error Control in Data Communication over the Switched Telephone Network B.S.T.J. Vol 43,6 Nov 1964 pp 2611-2638
Van Duuren H.C.A.	Typendruktelegrafic over Radioverbindiger TOR Tijdschrift Nederlands Radiogenootschap Mar 1951 pp 53-67
Van Voorhis D.C.	Constructing Codes with Bounded Codeword Lengths IEEE Trans. IT-20 March 1974 pp 288-290
Wanas M.A., Zayed A.I. Shaker M.M. Taha E.H.	First, Second and Third Order Entropies of Arabic Text IEEE Trans. IT-22 Jan 1976 pp 123-125

Wang Y. Lin S. — A Modified Selective-Repeat Type II Hybrid ARQ System and its Performance Analysis
IEEE Trans COM-31,5 May 1983
pp 593-607

Watanabe K., Inoue K., Sato Y. — A 4800 Bit/s Microprocessor Modem
IEEE Trans COM-26,5 May 1978
pp 493-498

Welch T.A. — A Technique for High-Performance Data Compression
Computer June 1984 pp 8-19

Wells M. — File Compression Using Variable Length Encodings
Computer Journal 1972 15,4 pp 308-313

Weng L,-J Sollman G.H. — Variable Redundancy Product Codes
IEEE Trans COM-15,6
Dec 1967 pp 835-838

Williams M.B. — The Characteristics of Telephone Circuits in Relation to Data Transmission
Post Office Elect. Eng. J. Vol 59,3
Oct 1966 pp 151-162

Yannakoudakis E.J., Goyal P. Huggill J.A. — The Generation and use of Text Fragments for Data Compression Information Processing and Management,
Vol 18,1 Jan 1982 pp 15-21

Yu Y. Gouda M. — Deadlock Detection for a Class of Communicating Finite State Machines
IEEE COM-30,12 1982 pp 2514-2518

Ziv J., Lempel A.

A Universal Algorithm for Sequential Data Compression
IEEE Trans. IT-23,3 May 1977
pp 337-343

*

Sinha V. Tambe M.A.
Baindur .

A Parallel Processing Hardware for Binary BCH Double Error Correcting Codec
J Inst Electronics & Telecomm Eng.
Vol 29,1 1983 pp 10-14

Appendix B

Extracts from the samples of data used for comparison of source codes.

In each case sufficient text is given to give a fair impression of the nature of the sample.

Sample Text1 Entropy 4.504 Length 40598 characters

Description - Technical paper

The first problem to be solved is that of finding a common point of interaction between the scientist and the machine. The reason is simple. There is no point in the scientist talking to the machine about one thing and the machine thinking that he is talking about something else, or vice versa.

Sample Text2 Entropy 4.641 Length 44685

Description: Documentation for software package.

```
----------------------------------------------------------------------

:                                                                    :
: CAPTURE LIVE IMAGE OR READ FROM DISK ? R                           :
:                                                                    :
: PLEASE INPUT NAME OF SAVED FILE E.G. DM1:IMDAT                     :
:                                                                    :
----------------------------------------------------------------------
```

If the user inputs the wrong reply then the IPL will prompt the user to input again, with the legal prompts shown to them.

Sample Text3 Entropy 4.456 Length 13873

Description: Technical document

In Artificial Intelligence (A.I) and related work, test rigs and other development tools are essential. The problem is that by its nature, the details of the research cannot be specified in advance and hence neither can the facilities which it requires.

Sample Text4 Entropy 4.722 Length 48753

Description: Technical document

The concept of template matching has found wide acceptance in segmentation applications mainly due to the simplicity of the method. The template is an array designed to detect some invariant regional characteristic. Many people have suggested various templates for detecting points, lines, arcs, etc [10,30-34].

Sample Text5 Entropy 4.019 Length 27198

Description: Technical document (containing a fair proportion of pseudo-Pascal program description.

Although any real time operating system should supply these facilities, execution of operating system functions should, in general, occupy only a small proportion of the processor time. For embedded system software, tasks will usually be created at system initialization, remain resident until system reset, and often (but not always) require a fixed resource allocation.

Sample FORTRAN1 Entropy 5.281 Length 5387

Description: FORTRAN source code.

```
WRITE(CON,600)
READ(CON,650)MARGIN
IF(MARGIN.EQ.0)MARGIN=8
WRITE(CON,670)
READ(CON,570)IDTIM,IDTIMP
WRITE(CON,590)IDTIM
```

Sample FORTRAN2 Entropy 4.773 Length 1971

Description : FORTRAN source code

```
C     INITIALISE EOF FLAG,CHARACTER COUNT(S)
      IEOF=0
      CHAR=0
      DO 50 I=1,130
 50       COUNT(I)=0
C
C     READ RECORDS UNTIL EOF
C
```

Sample Pascal1 Entropy 5.022 Length 7097

Description: Pascal source code

```
BEGIN (* PROCEDURE Correct *)
   Goodness := Dot(Vector, Rules[Chosen]);
  FOR j := 1 TO Noutputs DO
    IF (Decision[j] >= Goodness) AND (j <> Chosen) THEN
        Rules[j] := Vret[Diff(Rules[j], Vector)];
   Rules[Chosen] := Vret[Sum(Rules[j], Vector)];
END;  (* PROCEDURE Correct *)
```

Sample Pascal2 Entropy 4.420 Length 8656

Description: Pascal source code

```
      length:=length-1;
    UNTIL (length=0) OR (buflen=8);
  IF buflen=8 THEN buffull:=TRUE ELSE buffull:=FALSE;
END;{of bitpak}
```

Sample Prolog Entropy 4.747 Length 7218

Description: Prolog source code.

```
add:-write('unknown feature...please check word list '),nl
        retractall(feature(_)),retractall(featurelist(_)).
current:-context(X),entry(X,L),nl,write(X),nl,write(L),showmenu.
```

Sample Numbers Entropy 3.988 Length 2736

Description: Tables of decimal numbers

1	1.112	1.157	1.232	1.176
2	1.148	1.111	1.188	1.134
3	1.302	1.269	1.089	1.286
4	1.157	1.135	1.203	1.118
Mean length	1.176	std.dev	0.066	

Sample Image Entropy 4.734 Length 65664

Description: This was a grey level close-up image of the surface of a road. The sample was quantised to 7 bits.

Samples of text used for testing adaptive codes:

Sample type 1 Upper case text.

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG

Sample type 2 Mixed upper and lower case text

27 MORE THAN ONE RECORD

An attempt was made to read or write more than a single record in an encode or decode statement.

Sample type 3 Numeric data

.000 1.2468 19.531 .5363 39.063 -1.8771 58.594

78.125 -29.4091 97.656-6.2786 117.188 -1.2692 136.719

156.250 1.4597 175.781 .5815 195.313 -2.1197 214.844

234.375 -18.6578 253.906 -4.5225 2363.281 -21.6124

Appendix C. Derivation of ARQ Efficiency Equations.

C.1 Introduction

The derivations given below are generally well known, except perhaps for those in C.5 to C.7, but are included for completeness. The general approach to calculating ARQ efficiency is based on the expression:-

$$\text{Efficiency} = \frac{(k-h)}{k} \frac{1}{T}$$

where k is the length of the frame (and equal to n in non hybrid ARQ schemes), h the number of additional bits required to construct an ARQ frame, and T the expected number of transmissions required to send a frame.

The major difference between the efficiency equations given below is in T, the expected number of transmissions per frame.

An important element in the Stop and Wait, and Go Back N equations is N, the delay between the transmission and acknowledgement of a frame. This consists of the forward channel propagation delay, the backward channel propagation delay, and the time taken for the receiver to completely read the frame and check for errors. If piggybacking is used to transport the acknowledgements, the delay will be greater (see for example Lai 1982).

Acknowledgement delay $N = 1 + 2\,D/n$ frames

(round trip)

where D is the end to end propagation delay of the channel, and n the frame length (both D and n in are in bits).

The major assumptions made below are:-

(i) The forward and return channel have the same propagation delay.

(ii) An acknowledgement takes a negligibly small period of time to encode and decode.

(iii) The return channel is error free.

The block error probability is given in the form P(m>t,n), where n is the block length, t is the number of correctable errors and m the random variable denoting the number of errors within the block or frame. The expression P_E will be used to denote P(m>0,n).

C.2 Stop and Wait ARQ

In the Stop and Wait scheme, the transmitter will send a frame, wait N frame intervals, and then either retransmit the frame or send the next in sequence. The expected number of frame intervals required to send one frame successfully is thus:-

$$T = (1-P_E)N + P_E\,(1-P_E)2N + P_E{}^2(1-P_E)3N + \ldots$$

$$= N \sum_{i=1}^{\infty} i\,(1-P_E).P_E{}^{(i-1)}$$

$$= \frac{N}{(1-P_E)}$$

the efficiency is therefore

$$E = \frac{(k-h)}{k}\,\frac{(1-P_E)}{N}$$

C.3 Selective Repeat ARQ

In the Selective Repeat scheme, the transmitter sends continuously and only retransmits rejected frames. The average number of transmissions required to send one frame is:-

$$T = (1-P_E) + P_E\,(1-P_E).2 + P_E^{\,2}\,(1-P_E).3 + \ldots.$$

$$= \sum_{i=1}^{\infty} i\,(1-P_E).P_E^{(i-1)}$$

$$= \frac{1}{(1-P_E)}$$

the efficiency is therefore

$$E = \frac{(k-h)}{k}\,(1-P_E)$$

C.4 Go Back N ARQ

The Go Back N scheme is slightly less efficient than Selective Repeat as the transmitter retransmits a sequence of N frames when a reject is received.

$$T = (1-P_E) + (1+N)P_E(1-P_E) + (1+2N)P_E^2(1-P_E) + \ldots$$

$$= \sum_{i=0}^{\infty} (i.N+1).(1-P_E).P_E^i$$

$$= (1-P_E)\left(\frac{N.P_E}{(1-P_E)^2} + \frac{1}{(1-P_E)}\right)$$

$$= \frac{N.P_E}{(1-P_E)} + 1$$

the efficiency is therefore

$$E = \frac{(k-h)}{k} \frac{1}{\left(1 + \frac{N.P_E}{(1-P_E)}\right)}$$

C.5 Type I Hybrid GBN ARQ

The type I hybrid ARQ scheme employs an (n,k,t) forward error correcting code. This reduces the effective error rate, but also reduces the efficiency by the rate of the code.

The expression for T is basically the same as that given in section C.4, however P_E is replaced by P(m>t,n).

$$T = \frac{N.P(m>t,n)}{(1-P(m>t,n))} + 1$$

the efficiency is therefore

$$E = \frac{k}{n}\,\frac{(k-h)}{k}\,\frac{1}{\left(1 + \frac{N.P(m>t,n)}{(1-P(m>t,n))}\right)}$$

$$= \frac{(k-h)}{n}\,\frac{1}{\left(1 + \frac{N.P(m>t,n)}{(1-P(m>t,n))}\right)}$$

If desired, the value of N may be increased to allow for decoding delay.

C.6 Type II Hybrid GBN ARQ

The type II hybrid, or parity retransmission ARQ is based on the principal that error correcting code parity is sent in the retransmitted frame instead of data. This has the advantages that no additional parity is sent with the first transmission, which improves efficiency under error free conditions, but if errors are present the system effectively employs a half rate code which is capable of correcting a large number of errors.

The forward error correcting code is effectively a (2k,k,t) code, of which k data bits are sent in the first transmission, and k parity bits in the second. The effective number of transmissions is thus:-

$$T = (1-P_E) + (1+N).P_E.(1-P(m>t,n)) +$$

$$(1+2N).P_E.P(m>t,n).(1-P(m>t,n)) +$$

$$(1+3N).P_E.P(m>t,n)^2.(1-P(m>t,n)) +...$$

Note that n = 2k in this case, as the effective block length of the FEC code is 2k.

$$T = (1-P_E) + \sum_{i=1}^{\infty} (iN+1).P_E.(1-P(m>t,n)).P(m>t,n)^{(i-1)}$$

$$= (1-P_E) + P_E\,(1-P(m>t,n))\left(\frac{N}{(1-P(m>t,n))^2} + \frac{1}{(1-P(m>t,n))}\right)$$

$$= 1 + \frac{P_E N}{(1-P(m>t,n))}$$

the efficiency is therefore

$$E = \frac{(k-h)}{k}\,\frac{1}{\left(1 + \frac{N.P_E}{(1-P(m>t,2k))}\right)}$$

C.7 Adaptive Type I Hybrid GBN ARQ

The adaptive hybrid scheme has some similarity to the parity retransmission scheme discussed above, in that the retransmission probability for the second and subsequent frames is less than that of the first. This is because the forward error correction element of the system is switched in when a retransmission request is received.

The probability of a block being rejected depends on whether the block was encoded using the FEC code or not. The probability of a block being encoded (assuming that block errors are independent) is equivalent to the probability that the preceding c blocks are error free, where c is the number of frames that must be sent before the FEC code is switched out (the *sustain factor*).

$$P_f = 1 - (1-P_E)^c$$

The first transmission thus has a probability of success (i.e. no retransmission) of:-

$$P_{nr} = (P_f.(1-P(m>t,n)) + (1-P_f).(1-P_E))$$

Note that the block length is k for uncoded frames and n for encoded frames, therefore $P_E = P(m>0,k)$

The expected number of transmissions is thus:-

$$T = P_{nr} + (1+N).(1-P_{nr}).(1-P(m>t,n)) +$$

$$(1+2N).(1-P_{nr}).P(m>t,n).(1-P(m>t,n)) + \ldots$$

$$= P_{nr} + \sum_{i=1}^{\infty} (iN+1).(1-P_{nr}).(1-P(m>t,n)).P(m>t,n)^{(i-1)}$$

the next step follows C.6 to give:-

$$T = 1 + \frac{(1-P_{nr}).N}{(1-P(m>t,n))}$$

The expected frame length will be between k and n, as the uncoded frame has length k, and the encoded length n. Under the assumption given above in calculating P_f, the expected frame length will be:-

$$n' = P_f.n + (1-P_f).k$$

the efficiency is therefore:-

$$E = \frac{(k-h)}{(P_f.n+(1-P_f).k)} \frac{1}{(1 + \frac{(1-P_{nr}).N}{(1-P(m>t,n))})}$$

Appendix D. Derivation of Optimum Frame Length Equations for ARQ.

D.1 Introduction

The basic approach to calculating the optimum frame length is given by Chu (1974) and Morris (1979), although they give expressions in terms of wasted time rather than efficiency. The efficiency equation is differentiated with respect to block length, and the derivative set to zero. The optimum block length will be a root of the equation.

It should be noted that the conditions stated in Appendix C apply, and that both P_E and N are functions of the block length k. As $P_E = P(m>0,n)$ the derivative of P_E depends on the channel model, and will therefore be denoted merely as:-

$$\frac{d P_E}{dk} = P_E'$$

The derivative of N, the acknowledgement delay is easily found from the expression given in Appendix C Section C.1.

$$\frac{dN}{dk} = \frac{(1-N)}{k}$$

D.2 Stop and Wait ARQ

$$E = \frac{(k-h)}{k}\frac{(1-P_E)}{N}$$

$$\frac{dE}{dk} = \frac{(1-P_E)}{N}\frac{d}{dk}\frac{(k-h)}{k} + \frac{(k-h)}{k}\frac{d}{dk}\frac{(1-P_E)}{N}$$

$$= \frac{h}{k^2}\frac{(1-P_E)}{N} - \frac{(k-h)}{k}\left(\frac{P_E'}{N} + (1-P_E).\frac{(1-N)}{k.N^2}\right)$$

when the derivative is set to zero,

$$k^2 - k.\left(\frac{(N-1).(1-P_E)}{N.P_E'} + h\right) - h.\frac{(1-P_E)}{N.P_E'} = 0$$

The optimum frame length for Stop and Wait ARQ is a root of this equation. For example, the optimum frame length for a binary symmetric channel, with the parameters PT=0.001, D=1000 bits, and a header length of 48 bits, may be determined from the above as 783 bits.

D.3 Selective Repeat ARQ

$$E = \frac{[k-h]}{k}(1-P_E)$$

$$\frac{dE}{dk} = (1-P_E)\frac{d}{dk}\frac{[k-h]}{k} + \frac{[k-h]}{k}\frac{d}{dk}(1-P_E)$$

$$= \frac{h}{k^2}(1-P_E) + \frac{[k-h]}{k}P_E'$$

when the derivative is set to zero,

$$k^2 - k.h - h.\frac{(1-P_E)}{P_E} = 0$$

The optimum frame length for Selective Repeat ARQ is a root of this equation. Under the conditions given in D.2, the optimum frame length would be 244 bits.

D.4 Go Back N ARQ

$$E = \frac{(k-h)}{k} \frac{1}{\left(1 + \frac{N.P_E}{(1-P_E)}\right)}$$

$$= \frac{(k-h)}{k} \frac{(1-P_E)}{(1 + P_E.(N-1))}$$

$$\frac{dE}{dk} = \frac{(1-P_E)}{(1+P_E.(N-1))} \frac{d}{dk}\frac{(k-h)}{k} + \frac{(k-h)}{k} \frac{d}{dk} \frac{(1-P_E)}{(1+P_E.(N-1))}$$

$$= \frac{(1-P_E).(h + k.P_E.(N-1)) - (k-h).k.N.P_E'}{k.(1 + P_E.(N-1))^2}$$

when the derivative is set to zero,

$$(1-P_E)(h + k.P_E.(N-1)) - (k-h).k.N.P_E' = 0$$

$$k^2 - k\left(\frac{P_E.(1-P_E).(N-1)}{N.P_E'} + h\right) - h.\frac{(1-P_E)}{N.P_E'} = 0$$

The optimum frame length for Go Back N ARQ is a root of this expression. Under the conditions given in D.2 the optimum frame length would be 287 bits.