COMMUNICATION PROTOCOL CHANNEL UTILIZATION AND THE DESIGN OF THE MAX2 DATA LINK PROTOCOL

by

KENNETH LEE

B.Sc., The University of British Columbia, 1985

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming to the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

September 1991

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of COMPUTER SCIENCE

The University of British Columbia
Vancouver, Canada

Date SEPTEMBER 24, 1991

ABSTRACT

In this thesis, we first surveyed and analyzed the Kermit, XMODEM, YMODEM, and ZMODEM file transfer protocols. A number of theoretical channel utilization equations were then developed that would predict the effective utilization, and hence effective throughput, of stop and wait protocols and streaming protocols. A series of experiments were performed that measured the effective throughput of the protocols at various baud rates and error rates and showed the utilization equations to be within ±5% of the measured values. Based upon these results, a full-streaming communications protocol, MAX2, was designed. A subset of the MAX2 protocol was then implemented and tested at different baud rates and error rates. These results were within ±1.5% of the theoretical values and showed that the MAX2 protocol to be more efficient than the Kermit, XMODEM, YMODEM, and ZMODEM protocols at all baud rates and error rates when using comparable maximum packet sizes . These results also indicated that in order to achieve maximum effective throughput, the packet size must change as the communication channel's error rate changes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF GRAPHS

ACKNOWLEDGEMENT

The work in this thesis was funded in part by the National Research Council of Canada under IRAP Grant #9-8275-H-19. The equipment used in the experiments was provided by IPC Systems (3D Micro) and Telimax Software Corporation. Thanks must also be given to Dr. Sam Chanson who provided valuable guidance during the derivation of the throughput equations and the design of the throughput experiments.

1. INTRODUCTION

1.1 MOTIVATION AND OBJECTIVES

There are a large variety of protocols that can be used to transfer a file between two computer systems. Although many of these protocols are proprietary, there are several that are in the public domain and suitable for use on microcomputers. Protocols such as the Kermit, XMODEM, YMODEM, and ZMODEM protocols are in the public domain and are designed to perform end-to-end file transfers. The Kermit, XMODEM, and YMODEM are stop and wait protocols while the ZMODEM protocol is a streaming protocol with go back N retransmission.

Each of these protocols have different capabilities and characteristics that determine its channel utilization under various communication channel conditions. After examining these protocols, it was felt that none of the protocols make optimal use of the communications channel under all conditions and that a better protocol could be designed. This can be confirmed experimentally or by using mathematical equations that analyze the channel utilization of these and other similar protocols under different communication channel error rates, error distributions, and transmission speeds.

Work performed by Zacharov [16] and Field [3] showed that a protocol's packet size affects the channel utilization. As a packet's size increases, the chance that it is corrupted by transmission errors increases and the cost of recovering from a corrupted packet also increases. However, for a given communication channel error rate and distribution, there is a packet size that will allow for maximum channel utilization. Unfortunately, neither Field nor Zacharov considered the effect of a protocol's timeout period on the channel utilization of stop and wait protocols.

Miller [11], however, analyzed the throughput of several link level protocols as a function of the communication channel's bit error rate. The protocols examined used either a selective or nonselective retransmission scheme. As a result of his research, Miller, concluded that there is little performance difference between different retransmission strategies with a reliable communications channel. However, Miller's analysis did not take into account the possibility that a reply packet could be corrupted.

A set of simple channel utilization equations for stop and wait protocols and sliding window protocols have been derived by Stallings [13]. However, the equations assume that reply packets are always transmitted correctly. In addition, the channel utilization equations do not consider the effect that timeouts and reply packet sizes may have on a protocol's channel utilization. A more accurate set of channel utilization equations have been derived by Tanenbaum [16]. Although these equations take more factors into consideration, they do not differentiate between the possibilities that a packet can be lost or corrupted.

1.2 OUTLINE

A summary of the public domain file transfer protocols, Kermit, XMODEM, YMODEM, and ZMODEM is presented in Chapter 2. The packet sizes, packet formats, and encoding algorithm of each of the protocols and their variants are described. In addition, the capabilities and strengths of these protocols were compared.

In Chapter 3, a number of mathematical models are derived that analyzes the channel utilization of stop and wait protocols, streaming protocols with selective retransmission, and streaming protocols with go back N retransmission. These models show the effect that a protocol's data packet size, reply packet size, timeout period, and data encoding algorithm and the communication channel's error rate, error distribution, and

transmission speed have on a protocol's effective channel utilization. Once a protocol's effective channel utilization has been determined, the protocol's effective data throughput can be calculated by taking the product of the communication channel's transmission speed and the protocol's channel utilization. The validity of these mathematical models were verified by measuring the throughput of the Kermit, XMODEM, YMODEM, and ZMODEM protocols under different communication channel transmission speeds, error rates, and error distributions.

A new data link protocol, MAX2, is presented in Chapter 4. Using the information obtained from the work performed in Chapter 3, the MAX2 protocol is designed to be a powerful, versatile protocol that provides better channel utilization than the protocols described in Chapter 2. A prototype of the MAX2 protocol was implemented and its effective data throughput measured under different communication channel transmission speeds and error rates. These results were used to validate the channel utilization model for streaming protocols with selective retransmission and to show that it offers better channel utilization than the public domain file transfer protocols described in Chapter 2.

2. PROTOCOL SURVEY

2.1 INTRODUCTION

There are many protocols that can be used to transfer information over serial telecommunication lines. Several of these protocols have been placed in the public domain and are used by many users to transfer files between machines of possibly different architectures and operating systems.

This section summarizes the capabilities, strengths, and weaknesses of the following public domain file transfer protocols: Kermit, XMODEM, YMODEM, and ZMODEM. Although these protocols do not contain all seven layers of the OSI Model[1], they contain aspects of the application and data link layers. However, since each of these protocols are fully described in their respective protocol documents, only those aspects of the protocols that have a direct bearing on the channel utilization are described here.

2.2 KERMIT

2.2.1 Introduction

In 1981, a new file transfer protocol, Kermit, was developed at Columbia University by Bill Catchings and Frank da Cruz [3] to allow the exchange of data between a diverse mixture of computer systems. Unlike existing protocols, Kermit could be implemented on any computer system capable of transparently transmitting ASCII characters over seven bit serial lines. In addition to placing the Kermit protocol and its implementations in the public

---

1. Data and Computer Communications [13], page 389

domain, Columbia University also took on the task of coordinating the maintenance and distribution of the Kermit protocol and its implementations.

2.2.2 Classic Kermit

The original Kermit protocol, Classic Kermit, was a simple stop and wait protocol with selective retransmission. To ensure that communications between extremely dissimilar systems and possibly over seven bit serial lines will be successful, the Kermit protocol packets contain only seven bit printable ASCII characters, values 32 to 126.

In order to do so, the Kermit protocol applied two encoding algorithms to its packets. The first algorithm, "tochar", converted one byte unsigned integer field values into one byte encoded ASCII values by adding the value 32 to them. Consequently, Kermit integer field values were restricted to the values 0 to 94.

The second encoding algorithm, prefix encoding, is applied to the user data. It was a three pass algorithm that performs, in order, repeat encoding (optional), eighth bit encoding, and control encoding. These algorithms are shown in Figures 1, 2, and 3, respectively.

```
BEGIN Repeat Encoding Algorithm
   count = 1;
   WHILE (more data) AND (value = next value) AND (count < 94) DO
      count = count + 1
      value = next value
   END WHILE
   IF (count > REPEAT_THRESHOLD) THEN
      IF (tochar (count) = REPEAT_PREFIX_CHAR) THEN
         encoded count = REPEAT_PREFIX_CHAR, REPEAT_PREFIX_CHAR
      ELSE
         encoded count = tochar (count))
      END IF
      IF (value = REPEAT_PREFIX_CHAR) THEN
         RETURN (encoded count, value, value)
      ELSE
         RETURN (encoded count, value)
      END IF
   ELSE
      WHILE (count > 0) DO
         encoded value = encoded value, value
         IF (value = REPEAT_PREFIX_CHAR) THEN
            encoded value = encoded value, value
         END IF
         count = count - 1
      END WHILE
   END IF
END Repeat Encoding Algorithm
```

Figure 1. Kermit Repeat Encoding Algorithm

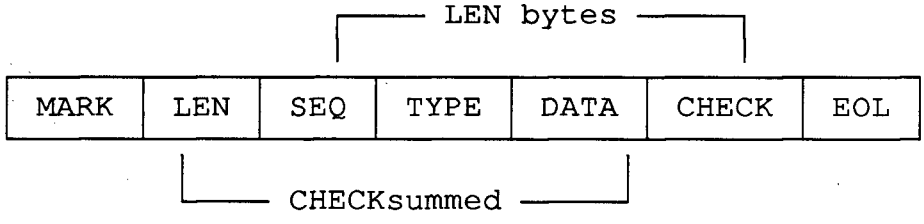
```
BEGIN Eighth Bit Prefix Encoding
   IF (value = EIGHTH_BIT_PREFIX_CHAR) THEN
      RETURN (EIGHTH_BIT_PREFIX_CHAR, EIGHTH_BIT_PREFIX_CHAR)
   ELSE IF (value > 7Fh) AND (7 bit communications) THEN
      RETURN (EIGHTH_BIT_PREFIX_CHAR, value & 80h)
   ELSE
      RETURN (value)
   END IF
END Eighth Bit Prefix Encoding
```

Figure 2. Kermit Eighth Bit Encoding Algorithm

```
BEGIN Control Prefix Encoding
   IF (value == CONTROL_PREFIX_CHAR) THEN
      RETURN (CONTROL_PREFIX_CHAR, CONTROL_PREFIX_CHAR)
   ELSE IF (32 <= (value AND 7Fh) <= 126) THEN
      RETURN (value)
   ELSE
      RETURN (CONTROL_PREFIX_CHAR, value XOR 40h)
   END IF
END Control Prefix Encoding
```

Figure 3. Kermit Control Prefix Encoding Algorithm

Once the data has been encoded, it is partitioned and inserted into the DATA field of packets, whose format is shown in Figure 4, such that the value of each packet's LEN field does not exceed 94 and no encoded sequence is split across two packets. If Kermit's CRC-CCITT checksum is used, each data packet could contain a maximum of 89 bytes of encoded data.




```
MARK    - indicates start of packet, usually CTL-A
LEN     - length of remainder of packet - 1 <= 94
SEQ     - sequence number MOD 64
TYPE    - packet type
DATA    - 0 to 91 bytes of encoded user data
CHECK   - 1 to 3 byte checksum
EOL     - end of line character, usually carriage return
```

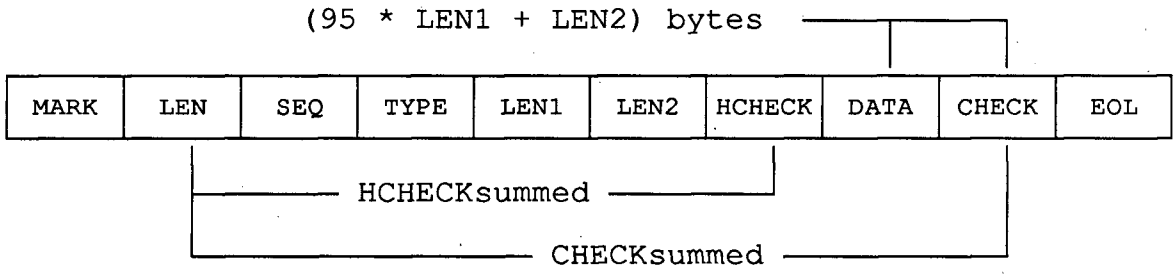
Figure 4. Classic Kermit Packet Format

The Kermit protocol's error mechanism is very simple. After sending a data packet, the sender waits for the receiver to send an acknowledgement. Since all needed information

is contained within the reply packet's header, the DATA field is empty. If the reply is a positive acknowledgement, the next data packet is sent; otherwise, the current data packet is retransmitted. However, if the sender does not receive a reply within a preset timeout period, it resends the current data packet. Conversely, if the receiver does not receive a data packet within its preset timeout period, it either resends its last reply or sends a negative acknowledgement for the expected data packet.

2.2.3 Extended Length Packet Extension

In 1986, extended length packets were added to the Kermit protocol. This extension was fully backwards compatible with the original protocol and provided a greater effective data throughput by allowing larger data packets to be used. The format of this extended length Kermit packet is shown in Figure 5.




```
MARK     - indicates start of packet, usually CTL-A
LEN      - set to 0
SEQ      - sequence number MOD 64
TYPE     - packet type
LEN1     - 0 to 94
LEN2     - 0 to 94
HCHECK   - one byte arithmetic checksum of SEQ, TYPE, LEN1,
           and LEN2 fields
DATA     - 0 to 9023 bytes of prefix encoded user data
CHECK    - 1 to 3 byte checksum
EOL      - end of line character, usually carriage return
```

Figure 5. Extended Length Kermit Packet Format

Although the use of extended length packets must be agreed upon by both the sender and receiver during the connection phase, both standard and extended length packets may be intermixed within the negotiated session. If the LEN field of a packet is greater than three, the packet is a standard length packet; otherwise, if the LEN field is zero, the packet is an extended length packet. Consequently, both extended length data packets and standard length reply packets can and are used during the data transfer phase.

2.2.4 Sliding Window Extension

In 1985, an extension was added to the original Kermit protocol that would allow sliding windows to be used with the existing Kermit packets. When the sliding windows extension is being used, the sender is allowed to send successive data packets without

waiting for their replies as long as the window is not full. The receiver simply sends acknowledgements for each data packet received. In addition, negative acknowledgements are sent to the sender for any missing packets that are detected by the receiver.

2.3 XMODEM

2.3.1 Introduction

The original version of the XMODEM protocol was developed by Ward Christensen in 1977 [4]. It is also referred to as "MODEM", "MODEM7", and the "Christensen Protocol". Since then, a number of modifications have been made that have produced XMODEM-CRC and XMODEM-1K. However, each of these modified XMODEM protocols, like the original, require an asynchronous transmission medium that is capable of providing 8 data bits, no parity bit, and one stop bit.

2.3.2 XMODEM (Original)

The XMODEM protocol is a simple stop and wait protocol. It is comprised of only two types of packets: control and data packets. The control packets, ACK, NAK, CAN, and EOT, are simple one byte packets. Meanwhile, data packets, shown in Figure 6, are 132 bytes long. Furthermore, the XMODEM protocol does not perform any encoding.

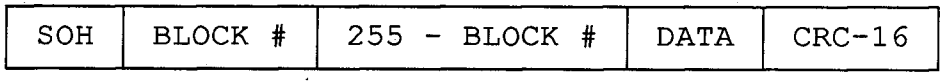

```
SOH            - 01H, indicates start of packet
BLOCK #        - sequence number MOD 256
255 - BLOCK #  - ones complement of sequence number
DATA           - 128 bytes of user data
CHECKSUM       - bytewise checksum
```

Figure 6. XMODEM DATA Packet Format

After the sender has sent a DATA packet, it waits for a reply from the receiver. If the reply is an ACK, the next DATA packet is sent; otherwise, if the reply is an NAK, the DATA packet is resent. However, if the reply is a CAN, the transfer is aborted. If the sender does not receive a reply before its timeout period expires, it resends the current DATA packet. Conversely, if the receiver times out, it sends an NAK.

2.3.3 XMODEM-CRC

The XMODEM-CRC protocol is a simple extension of the XMODEM protocol to provide CRC-16 checksums. During the connection phase, if the receiver sends the C packet, a new control packet, instead of the NAK packet, the sender uses the DATA-CRC packets shown in Figure 7 instead of the normal DATA packets.




```
SOH            - 01H, indicates start of packet
BLOCK #        - sequence number MOD 256
255 - BLOCK #  - ones complement of sequence number
DATA           - 128 bytes of user data
CRC-16         - 16 bit CRC
```

Figure 7. XMODEM-CRC DATA-CRC Packet Format

2.3.4 XMODEM-1K

In 1982, Chuck Forsberg improved the throughput of XMODEM and XMODEM-CRC by extending the size of the DATA packets and created the XMODEM-1K protocol [4]. This protocol required a simple modification to the XMODEM-CRC protocol and the addition of a DATA-1K packet shown in Figure 8.

STX	BLOCK #	255 - BLOCK #	DATA	CHECKSUM

```
STX            - 02H, indicates start of packet
BLOCK #        - sequence number MOD 256
255 - BLOCK #  - ones complement of sequence number
DATA           - 1024 bytes of user data
CHECKSUM       - CRC16 (strongly recommended) or
                 bytewise arithmetic checksum
```

Figure 8. XMODEM-1K DATA-1K Packet Format

The checksum used in the DATA-1K packets is determined by the receiver. Like the XMODEM-CRC protocol, if the receiver sends the C packet at start up, the sender uses a CRC-16 checksum in its data packets; otherwise, a bytewise arithmetic checksum is used. Like the other XMODEM packets, DATA-1K packets are fixed length and its contents are not encoded.

2.4 YMODEM

The YMODEM protocol is a modified XMODEM-1K protocol developed by Chuck Forsberg that allows multiple files to be transferred after a connection has been established [4]. When the sender receives a C packet, it sends a DATA packet containing the name (and possibly size) of the file about to be transferred. Upon receiving this DATA

packet, the receiver acknowledges it appropriately. When the file information has been received correctly, if it is possible to receive and store the specified file, another C packet is sent; otherwise, a CAN packet is sent to abort the transfer of the specified file.

Once the sender receives the second C packet, the file transfer begins and is identical to the XMODEM-1K protocol. When the file transfer is complete, the receiver sends a C packet to request information on the next file to be transferred. If there is no file to be transferred, the sender responds with a DATA packet containing a blank file name. Once this packet is successfully ACK'ed the connection is terminated.

A variant of the YMODEM protocol, YMODEM-g, is a streaming version of YMODEM with error detection but no error recovery. It is initiated by the receiver sending a G packet instead of the C or NAK packet at start up. In this mode, the sender simply sends data packets as fast as the transfer medium will accept them. If the receiver detects a transmission error in any data packet, it aborts the transfer.

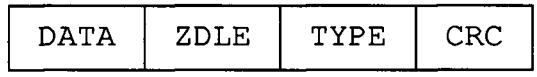
2.5 ZMODEM

In 1986, Telenet, a Public Data Network, funded a project to develop a public domain file transfer protocol. The purpose of this new protocol was to provide Telenet customers with a protocol capable of greater throughput than what was possible with existing protocols such as XMODEM, YMODEM, and Kermit. This new protocol, ZMODEM, was designed by Chuck Forsberg [5].

The ZMODEM protocol is a streaming protocol with a "go back N" error recovery mechanism. During the data phase, the sender sends a ZDATA control packet followed by data packets. If the receiver detects a transmission error, it sends a ZRPOS control packet to request retransmission of file data beginning at the position specified. If the ZRPOS packet

was received by the sender correctly, it responds with a ZDATA packet and then data packets containing the requested information.

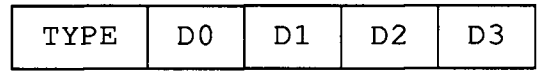
Although a ZMODEM data packet, shown in Figure 9, is always "binary" encoded, a control packet, shown in Figures 10 and 11 can be "binary" or "hex" encoded.




```
DATA - 0 - 1024 bytes of user data
ZDLE - indicates end of user data
TYPE - indicates data packet type
CRC  - either a 16 or 32 bit CRC, same
       as preceding packet
```

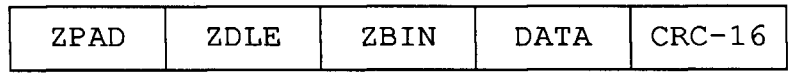
Figure 9. ZMODEM Data Packet Format

Although the binary encoding algorithm is similar to Kermit's control encoding algorithm, ZMODEM's algorithm only prefix encodes the following control values: 16, 17, 19, 24, 127, 144, 145, 147, and 255. In addition, each character in the Telnet command string, "CR-@-CR", is prefix encoded. The operator may also request that all control values be prefix encoded. If the control packet is "hex" encoded, each byte is represented by a two character ASCII (lower case) string which corresponds its hexadecimal value. In either case, the checksum of the packets shown in Figure 11 is also similarly encoded.

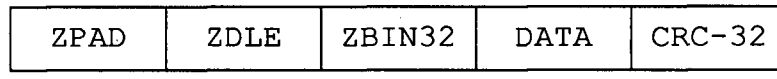


TYPE - packet type
D0 - position byte 0 (P0) or flag byte 3 (F3)
D1 - position byte 1 (P1) or flag byte 2 (F2)
D2 - position byte 2 (P2) or flag byte 1 (F1)
D3 - position byte 3 (P3) or flag byte 0 (F0)

Figure 10. ZMODEM Control Packet Data Field



16 bit CRC Binary Encoding



32 bit CRC Binary Encoding

16 bit CRC HEX Encoding

LF - ASCII value 10 (^J)
CR - ASCII value 13 (^M)
XON - ASCII value 17 (^Q)
ZDLE - ASCII value 24 (^X)
ZPAD - ASCII value 42 (*)
ZBIN - ASCII value 65 (A)
ZHEX - ASCII value 66 (B)
ZBIN32 - ASCII value 67 (C)
DATA - ZMODEM Control Packet

Figure 11. ZMODEM Control Packet Formats

2.6 SUMMARY

The protocols surveyed each offer their own unique combination of features. This often makes one protocol more suitable than others under different conditions. These features are summarized in Table 1.

The strength of the Kermit protocol is its ability to reliably transfer data between widely dissimilar systems. However, in order to do so, the Kermit protocol used packets

consisting of only printable ASCII characters. Unfortunately, this resulted in a high protocol overhead and ultimately low effective channel utilization.

The XMODEM and YMODEM protocols and their variants, on the other hand, do not perform any encoding. However, although these protocols offer a high effective channel utilization, they require eight bit data communication channels that allow all eight values to pass through transparently. Unfortunately, these protocols are not very reliable since their single byte acknowledgement packets are unprotected.

While the other protocols surveyed are stop and wait protocols, the ZMODEM protocol is a streaming protocol with a "go back N" error recovery mechanism. This protocol offers control character encoding and protected acknowledgements like Kermit but with a protocol overhead comparable to that of the XMODEM and YMODEM protocols. Consequently, it can offer reliable data transfers and a high effective channel utilization.

FEATURE	Kermit Classic	Kermit Extended	XMODEM CRC	YMODEM	ZMODEM
FLOW CONTROL:					
· Stop and Wait	✔	✔	✔	✔	o
· Sliding Windows	o	o	✗	✗	o
· Streaming	✗	✗	✗	o	✔
ERROR DETECTION:					
· 1 byte sum	✔	✔	✔	✔	✗
· 2 byte sum	✔	✔	✗	✗	✗
· 16 bit CRC	✔	✔	✔	✔	✔
· 32 bit CRC	✗	✗	✗	✗	✔
ERROR RECOVERY:					
· selective retransmission	✔	✔	✔	✔	✗
· go back N	✗	✗	✗	✗	✔
PACKET SIZES[1]:					
· Data Packet (Min)	8	8	133	1029	3
· Data Packet (Max)	97	1033[2]	133	1029	1027
· Reply Packet	8	8	1	1	21
· Max. Data Field	89	1024	128	1024	1024
OVERHEAD:					
· Bytes Encoded[3]	70	70	0	0	9
· Encoding Factor	1.273	1.273	1.000	1.000	1.035
· Raw Data / Packet	70.0	804.1	128.0	1024.0	989.2
· Overhead To Send One Data Packet[4]	50.0%	29.5%	4.7%	0.6%	3.8%

✔ - supported
o - supported but not normally used
✗ - not supported
1 - based on 16 bit CRC checksum
2 - for data field of 1024 bytes
3 - assuming minimum encoding on stream of 0 to 255
4 - 1 - (Data Packet + ACK Packet) / Raw Data

Table 1. Protocol Survey Feature Summary

2.7 CONCLUSION

Of the protocols surveyed, ZMODEM has become the protocol of choice whenever it is available. It offers the reliability of the Kermit protocol, efficiency comparable to YMODEM, and the ability to resume an incomplete file transfer.

However, despite Kermit's low efficiency, it is often the protocol used to transfer data between dissimilar systems. In many cases, it is the only protocol implemented on both systems. In addition, because of its aggressive encoding which uses only printable ASCII characters, its packets often make it through communication channels successfully where other protocols fail.

3. PROTOCOL CHANNEL UTILIZATION

3.1 INTRODUCTION

When several communication protocols are available and each capable of successfully transferring data between two systems, it is often not clear which protocol is the most appropriate. Some protocols offer a higher effective data throughput over an error free communications channel but degrade rapidly as the error rate increases. However, since a protocol's behaviour and characteristics can be represented mathematically, we set out to derive a set of equations that would calculate a protocol's effective data throughput or channel utilization under different mathematically representable conditions.

The equations that were developed ignored the overhead of the connection establishment and the disconnect phases of the protocols. The equations also assumed that an unreliable communications channel only corrupts data and its behaviour can represented by its error rate and distribution. However, although no data is lost, a packet may be missed due to the corruption of the packet's header byte or byte sequence. The equations also take the turnaround time and propagation delay into account. If a protocol's theoretical throughput is desired, it can be computed by simply taking the product of the channel's transmission speed and the protocol's theoretical channel utilization.

3.2 ANALYSIS

The operation of a protocol can be broken down into six basic events. When a data packet is transmitted, it is either received correctly (Data OK), received with transmission errors (Data BAD), or lost (Data LOST) due to header corruption. The probability of these events occurring is determined by the error rate and distribution of the communications

channel and the size of the packet. Similarly, a reply packet can be received correctly (Reply OK), received incorrectly (Reply BAD), or lost (Reply LOST).

Channel utilization equations will be derived for stop and wait protocols, streaming protocols with selective retransmission, and streaming protocols that "go back N" for uniformly distributed byte errors and randomly distributed bit errors. Each of these equations will be expressed as a function of the probability of the six basic events occurring. This allows the utilization equations to handle any error distribution provided probabilities can be assigned to each of the basic events.

3.2.1 Uniform Deterministic Error Distribution

Although not realistic, one of the error distributions examined is uniform and deterministic. In this distribution, it is assumed that an error occurs every K bytes that are placed on the communications channel. Consequently, if P is the number of bytes in a packet, the probability that it is received correctly is

$$P(\text{Packet OK}) = 1 - P / K \qquad \textbf{Eq. 1}$$

However, if we assume that a packet has a header byte which must be received correctly in order for the receiver to not miss the packet, then the probability that a packet is lost is

$$P(\text{Packet LOST}) = 1 / K \qquad \textbf{Eq. 2}$$

Therefore, the probability that a packet is received with transmission errors is

$$P(\text{Packet BAD}) = 1 - P(\text{Packet OK}) - P(\text{Packet LOST}) = 1 - (1 - P / K) - 1 / K = (P - 1) / K \quad \textbf{Eq. 3}$$

Although these equations offer a very simplistic model of the packet error distribution, it should be sufficient provided the size of the data packet is many times greater than the reply packet. Under these constraints, transmission errors will occur primarily in the data packets.

3.2.2 Poisson Error Distribution

A more realistic error distribution is the poisson distribution. In this distribution, each bit placed on the communications channel has an equally likely probability Q that it will be corrupted. Thus, if a packet has P bytes and each byte contains W bits (including the start and stop bits if necessary), then the probability that a packet is received correctly is

$$P(\text{Packet OK}) = (1 - Q)^{(P * W)} \quad \textbf{Eq. 4}$$

Again, if we assume that a packet has a header byte which must be received correctly so as to not be missed by the receiver, then the probability that a packet is lost is

$$P(\text{Packet LOST}) = 1 - (1 - Q)^{W} \quad \textbf{Eq. 5}$$

Consequently, the probability that a packet is received with transmission errors is

$$P(\text{Packet BAD}) = 1 - P(\text{Packet OK}) - P(\text{Packet LOST}) = (1 - Q)^{W} - (1 - Q)^{(P * W)} \quad \textbf{Eq. 6}$$

3.3 CHANNEL UTILIZATION EQUATION FOR STOP AND WAIT PROTOCOLS

With stop and wait protocols, in order for the sender to consider a data packet successfully received, the receiver must have received it correctly and the positive acknowledgement must be received correctly by the sender. Thus, the probability of a data packet being successfully received is

$$P(\text{Success}) = P(\text{Data OK}) * P(\text{Reply OK}) \quad \textbf{Eq. 7}$$

The data packet may also be resent immediately by the sender upon receipt of the receiver's reply. If the reply is negative, the data packet is resent. However, if the reply is corrupted, it is assumed that the sender resends the data packet immediately since this is more efficient than waiting for the timeout period to expire. Thus the probability that a data packet is resent upon receipt of a reply, P(Resend), is the sum of three products.

$$\begin{aligned} P(\text{Resend}) = {} & P(\text{Data OK}) * P(\text{Reply BAD}) + \\ & P(\text{Data BAD}) * P(\text{Reply OK}) + \\ & P(\text{Data BAD}) * P(\text{Reply BAD}) \end{aligned} \quad \textbf{Eq. 8}$$

If either the data or reply packet is lost, the data packet is resent after the timeout period expires. The probability that a timeout will occur is

```
P(Timeout) = P(Data OK)   * P(Reply LOST) +
             P(Data BAD)  * P(Reply LOST) +
             P(Data LOST)
```

Eq. 9

After the sender has sent the data packet, there will be an elapsed time before the receiver's reply is received. This time period, Turn, encompasses the propagation delay and the time required by the receiving system to decode the data packet and to send the appropriate reply. Thus, if B represents the channel speed in bytes per second, the channel bandwidth required to send a data packet of D bytes and receive a reply packet of R bytes is

```
D + R + Turn * B
```

Eq. 10

Similarly, if Time is the timeout period in seconds, the bandwidth occupied by a timeout is

```
D + Time * B
```

Eq. 11

Since the probability of sending a data packet correctly and receiving the positive acknowledgement correctly is P(Success), the expected number of trials before a success is

```
Trials = 1 / P(Success)
       = 1 / (P(Data OK)  * P(Reply OK))
```

Eq. 12

The encoding overhead of the protocol is calculated as a ratio of the number of bytes needed to represent the byte stream of values from 0 to 255 versus the number of raw bytes, 256. This ratio, E, is greater than or equal to 1. Using this figure, the amount of raw data that is in a data packet containing DF bytes of encoded data is simply

$$\text{Raw Data} = DF / E \qquad \textbf{Eq. 13}$$

With the above figures and equations, the bandwidth required to successfully transfer a data packet can be calculated as

$$\begin{aligned}\text{Total Bandwidth} &= (P(\text{Success}) * (D + R + \text{Turn} * B) + \\ &\quad P(\text{Retry}) * (D + R + \text{Turn} * B) + \\ &\quad P(\text{Timeout}) * (D + \text{Time} * B)) * \text{Trials} \\ &= ((1 - P(\text{Timeout})) * (D + R + \text{Turn} * B) + \\ &\quad P(\text{Timeout}) * (D + \text{Time} * B)) * \text{Trials} \\ &= \frac{(1 - P(\text{Timeout})) * (D + R + \text{Turn} * B) + P(\text{Timeout}) * (D + \text{Time} * B)}{P(\text{Data OK}) * P(\text{Reply OK})}\end{aligned} \qquad \textbf{Eq. 14}$$

where $P(\text{Success}) + P(\text{Retry}) = 1 - P(\text{Timeout})$
and Trials is given by Equation 12

Consequently, the theoretical channel utilization, U, is

$$U = \frac{\text{Raw Data}}{\text{Total Bandwidth}} \qquad \textbf{Eq. 15}$$

$$= \frac{(DF / E) * P(\text{Data OK}) * P(\text{Reply OK})}{\left[\begin{array}{c}[(1 - P(\text{Data LOST})) * (1 - P(\text{Reply LOST}))] * \\ (D + R + \text{Turn} * B) + \\ [(1 - P(\text{Data LOST}) * P(\text{Reply LOST}) + \\ P(\text{Data LOST})] * (D + \text{Time} * B)\end{array}\right]}$$

Figure 12. Utilization Equation For Stop and Wait Protocols

3.4 CHANNEL UTILIZATION EQUATION FOR STREAMING PROTOCOLS

With many streaming protocols, when a data packet is received with no detectable transmission errors, no positive acknowledgement is required. However, there are some streaming protocols that require a positive acknowledgement to be sent for some or all correctly received data packets before the data packet is considered to be received successfully. Thus, if AR represents the percentage of data packets that need to be positively acknowledged, the probability that a data packet is successfully received is

$$P(\text{Success}) = P(\text{Data OK}) * P(\text{ACK OK}) * AR + P(\text{Data OK}) * (1 - AR) \quad \textbf{Eq. 16}$$

However, if a data packet is corrupted, a negative acknowledgement is sent back to the sender. If it is received correctly, the sender resends the corresponding data packet. A negative acknowledgement is also sent when the receiver detects a lost data packet. The probability of this occurring is

$$\begin{aligned} P(\text{Resend}) &= P(\text{Data BAD}) * P(\text{NAK OK}) + P(\text{Data LOST}) * P(\text{NAK OK}) \\ &= (1 - P(\text{Data OK})) * P(\text{NAK OK}) \end{aligned} \quad \textbf{Eq. 17}$$

In order to handle the case when the receiver only sends negative acknowledgements, we will assume that only the receiver will timeout. This occurs when it does not receive the data packet requested before its timeout period expires. Thus, the probability of a timeout occurring is

$$
\begin{aligned}
P(\text{Timeout}) &= P(\text{Data OK}) * P(\text{ACK BAD}) * AR + \\
&\quad P(\text{Data OK}) * P(\text{ACK LOST}) * AR + \\
&\quad P(\text{Data BAD}) * P(\text{NAK BAD}) + \\
&\quad P(\text{Data BAD}) * P(\text{NAK LOST}) + \\
&\quad P(\text{Data LOST}) * P(\text{NAK BAD}) + \\
&\quad P(\text{Data LOST}) * P(\text{NAK LOST}) \\
&= P(\text{Data OK}) * (1 - P(\text{ACK OK})) * AR + \\
&\quad (1 - P(\text{Data OK})) * (1 - P(\text{NAK OK}))
\end{aligned}
\qquad \textbf{Eq. 18}
$$

Like stop and wait protocols, the number of trials needed to send a data packet successfully is given by Equation 12 and the amount of raw data in a data packet is given by Equation 13.

Unlike stop and wait protocols, there is more than one error recovery mechanism used by streaming protocols. If the receiver keeps the correct packets received after the detected error, the sender only retransmits the packet in question. This error recovery mechanism is referred to as selective retransmission. Conversely, with a "go back N" error recovery mechanism, the receiver discards all packets received since the detected error and the sender retransmits the packet in question and all packets that were subsequently transmitted. Unfortunately, a channel utilization equation must be derived for each error recovery mechanism.

3.4.1 Selective Retransmission

If data and reply packets are exchanged in half-duplex mode, then the channel bandwidth required to send a data packet and receive its reply is simply the sum of the packet sizes. However, if the data packet is lost, there is no corresponding reply packet. Thus, if D, A, and N represent the size of the data, positive acknowledgement, and negative acknowledgement packets, respectively, the total bandwidth required to successfully transfer a data packet is

$$\text{Total Bandwidth} = [P(\text{Data OK}) * (D + A * AR) + P(\text{Data BAD}) * (D + N) + P(\text{Data LOST}) * (D + N)] * \text{Trials}$$

$$= \frac{[P(\text{Data OK}) * (D + A * AR) + P(\text{Data BAD}) * (D + N) + P(\text{Data LOST}) * (D + N)]}{P(\text{Data OK}) * P(\text{Reply OK})}$$

Eq. 19

where Trials is given by Equation 12

Thus the channel utilization of a half-duplex streaming protocol using selective retransmission is

$$U = \frac{\text{Raw Data}}{\text{Total Bandwidth}}$$

$$= \frac{(DF / E) * P(\text{Data OK}) * P(\text{ACK OK})}{\left[\begin{array}{l} P(\text{Data OK}) * (D + A * AR) + \\ P(\text{Data BAD}) * (D + N) + \\ P(\text{Data LOST}) * (D + N) \end{array}\right]}$$

Eq. 20

Figure 13. Utilization Equation For Half-Duplex Streaming Protocols (Selective Retransmission)

However, if data and reply packets are exchanged in full-duplex mode, the reply packets do not occupy any channel bandwidth in the direction of the data packet transfers and hence the channel utilization equation, Equation 20, reduces to

$$U = (DF / E) * P(\text{Data OK}) * P(\text{ACK OK}) / D$$

Eq. 21

Figure 14. Utilization Equation For Full-Duplex Streaming Protocols (Selective Retransmission)

3.4.2 Go Back N

With a go back N error recovery mechanism, it is sufficient for the receiver to send only negative acknowledgements. Knowing which packets have been correctly received is irrelevant when all data packets transmitted since the corrupted data packet are retransmitted. Thus, the channel bandwidth occupied when a data packet is correctly received is the size of the data packet while the number of trials required before a data packet is successfully transferred is

```
Trials = 1 / P(Data OK)
```

Eq. 22

When a data packet is corrupted, however, a negative acknowledgement is sent to the sender. The time, Turn, that elapses from the moment the corrupted data packet was sent until a correct negative acknowledgement packet is received determines the number of additional data packets that were sent. Thus, if the data packet size is D bytes, the reply is N bytes, and the channel speed is B bytes per second, then the channel bandwidth occupied is

```
D + N + D * CEILING(Turn * B / D)
```

Eq. 23

If the negative acknowledgement is corrupted, it is discarded by the sender. Until the receiver times out in Time seconds and the sender receives a correct reply packet, it continues to send data packets. Hence, since 1 / P(NAK OK) is the expected number of trials before the reply packet is correctly received, the bandwidth wasted until the sender receives a correct reply packet is

```
D * (1 / P(NAK OK) - 1) * CEILING(Time * B / D)
```
Eq. 24

Thus, the channel bandwidth used in recovering from a corrupted data packet is

```
D + N + D * CEILING(Turn * B / D) + N * D *
(1 / P(NAK OK) - 1) * CEILING(Time * B / D)
```
Eq. 25

If a data packet is lost, it is detected by the reception of the next data packet. Consequently, the channel bandwidth needed to recover from a lost data packet is only D greater than when the data packet is corrupted. Thus, the total bandwidth required to send a data packet correctly can be calculated as

```
Total Bandwidth = [P(Data OK) * D +                          Eq. 26
                   P(Data BAD) * Band BAD +
                   P(Data LOST) * Band LOST] * Trials

                = [P(Data OK) * D +
                   P(Data BAD) * Band BAD +
                   P(Data LOST) * Band LOST] / P(Data OK)

 where     TurnD = D * CEILING(Turn * B / D)
           TimeD = D * CEILING(Time * B / D)
       Bad NAKS = 1 / P(NAK OK) - 1
       Band BAD = (D + N + TurnD + N * Bad NAKS * TimeD)
      Band LOST = D + Band BAD
            and Trials is given by Equation 22
```

Hence, the channel utilization of a half-duplex streaming protocol using a go back N error recovery mechanism is

$$U = \frac{\text{Raw Data}}{\text{Total Bandwidth}} = \frac{(DF / E) * P(\text{Data OK})}{\left[\begin{array}{l} P(\text{Data OK}) * D + \\ P(\text{Data BAD}) * \text{Band BAD} + \\ P(\text{Data LOST}) * \text{Band LOST} \end{array}\right]}$$

Eq. 27

where
$$\text{TurnD} = D * \text{CEILING}(\text{Turn} * B / D)$$
$$\text{TimeD} = D * \text{CEILING}(\text{Time} * B / D)$$
$$\text{Bad NAKS} = 1 / P(\text{NAK OK}) - 1$$
$$\text{Band BAD} = (D + N + \text{TurnD} + N * \text{Bad NAKS} * \text{TimeD})$$
$$\text{Band LOST} = D + \text{Band BAD}$$

Figure 15. Utilization Equation For Half-Duplex Streaming Protocols (Go Back N)

For a full-duplex streaming protocol using a go back N error recovery mechanism, the data bandwidth occupied by the reply packets is zero. Hence, the channel utilization equation becomes

$$U = \frac{\text{Raw Data}}{\text{Total Bandwidth}} = \frac{(DF / E) * P(\text{Data OK})}{\left[\begin{array}{l} P(\text{Data OK}) * D + \\ P(\text{Data BAD}) * \text{Band BAD} + \\ P(\text{Data LOST}) * \text{Band LOST} \end{array}\right]}$$

Eq. 28

where
$$\text{TurnD} = D * \text{CEILING}(\text{Turn} * B / D)$$
$$\text{TimeD} = D * \text{CEILING}(\text{Time} * B / D)$$
$$\text{Bad NAKS} = 1 / P(\text{NAK OK}) - 1$$
$$\text{Band BAD} = (D + \text{TurnD} + \text{Bad NAKS} * \text{TimeD})$$
$$\text{Band LOST} = D + \text{Band BAD}$$

Figure 16. Channel Utilization For Full-Duplex Streaming Protocols (Go Back N)

3.5 EQUATION VERIFICATION

In order to verify the correctness of the channel utilization equations derived, a series of throughput experiments were performed. In each of the experiments, throughput measurements were taken for the Kermit, XMODEM, YMODEM, and ZMODEM protocols under a variety of baud rates, error rates, and error distributions. These experiments were performed using three IBM PC compatible computers interconnected by null modem RS-232 serial cables as shown in Figure 17.

Figure 17. Throughput Test Computer Configuration

The SENDER machine, a 20 MHz 386, was used to send either the 100 kilobyte test file or the one megabyte test file. The content of these test files was a repeated sequence of the values 0 to 255. The RECEIVER machine, a 12 MHz 286, was used to receive the test file. A 100 kilobyte test file was selected to amortize to time required to establish a connection and to close the connection. In addition, it would allow several errors to occur during the transfer at low error rates. However, a one megabyte test file was used when the error distribution was random to ensure that a representative error sample was encountered.

The ERROR GENERATOR machine, a 12 MHz 286, was equipped with two RS-232 serial ports and ran an error generator program. Three different error generator programs, each providing a different error distribution, were written using Borland's Turbo Assembler 1.0 and Turbo C 2.0. Each program would receive data from either serial

port and send it out on the other serial port. However, before the data was forwarded, it was modified to produce the desired error rate and distribution.

3.5.1 Experiment Description

The implementation of the Kermit protocol that was used was a patched version of the IBM PC implementation from Columbia University, Kermit-MS 2.32/A. The patch was necessary to increase the resolution of the file transfer timer and to prevent the sender from aborting a file transfer when an acknowledgement packet's type field was corrupted. For the XMODEM and YMODEM protocols, both the Procomm-Plus 1.1B and the DSZ 11-14-89 implementations were used. Finally, the ZMODEM implementation that was used was DSZ 11-14-89.

With the exception of DSZ, which reported the effective throughput after completing a file transfer, the throughput of the various implementations of the protocols was determined by recording the time required to transfer the test file. For the Procomm implementation of the XMODEM and YMODEM protocols, the file transfer on the sending system was started before the receiving system. The elapsed time was then measured by using a Procomm script to store the start and stop time of the file transfer on the receiving system. On the other hand, the Kermit-MS 2.32/A program on the receiving system was started before the sending system's program. In this case, since the elapsed time of the file transfer was recorded by the program, the throughput was calculated using the elapsed time reported by the sending program.

For each set of conditions, an experiment was performed to determine the effective throughput of transferring either the 100 kilobyte or one megabyte test file using a protocol. The test file was transferred three times and the elapsed time or throughput was recorded

for each transfer. If the protocol offered different levels of encoding, the lowest level of encoding was selected to minimize the protocol's overhead and increase its channel utilization. In addition, if the protocol offered several types of checksums, the most complex checksum was selected to minimize the probability that a transmission error would escape detection. This was repeated for each protocol.

After calculating the average measured throughput and channel utilization for each experiment, it was compared to the theoretical throughput and channel utilization predicted by the corresponding channel utilization equation. The theoretical channel utilization of the protocols examined have been plotted in Graphs 1 to 4 and Graphs 23 to 30 as a function of the uniform byte error rate and the random bit error rate, respectively. In addition, the theoretical throughput of the protocols have also been plotted as a function of the uniform byte error rate and random bit error rate in Graphs 5 to 8 and Graphs 31 to 38, respectively.

The effect of the baud rate, error rate, and error distribution on the measured throughput of the protocols was examined by plotting a series of graphs (Graphs 9 to 15 and 39 to 40) which plots the measured throughput as a function of the error rate for each protocol at a baud rate. Finally, the percentage difference between the measured throughput and theoretical throughput values were calculated and plotted against the error rate in Graphs 16 to 22 and 41 to 42.

However, unlike the other parameters of Equation 15, the Turn term cannot be measured directly. Consequently, the value of the Turn term was calculated by solving the Turn term in Equation 15 after setting all other parameters to their appropriate values. This was done for each protocol implementation using an error rate of 0 using the average of the three measured throughput figures for the throughput value. Naturally, Turn terms were calculated for each protocol at the different baud rates.

The analysis of the experimental data collected was performed initially using Borland's Reflex 2.0, a "database management, graphics, and analysis" program. Although Reflex handled the data manipulation and calculations well, the graphs that it generated were not clearly labelled. Consequently, Reflex was replaced with Borland's Quattro Pro 3.0, a spreadsheet program, which generated better labelled graphs.

3.5.2 Uniform Byte Error Distribution Experiments

The uniform byte error distribution was provided by the BLACK10 program. This error generator program would simply count the bytes that it received from either serial port and then modify every K^{th} byte to produce a bytewise error rate of 1/K.

The throughput experiments were performed at 2400, 4800, 9600, and 19200 baud and at error rates of 0, 1/10000, 1/7500, 1/5000, 1/2500, and 1/1000 errors/byte for each implementation of the protocols. Table 2 lists the minimum, maximum, and average percentage difference for all combinations of baud rates and error rates for each protocol and implementation.

PROTOCOL	MINIMUM	MAXIMUM	AVERAGE
Classic Kermit	-3.3	0.8	-1.0
Extended Kermit[1]	-3.3	1.8	0.2
XMODEM-CRC (Procomm)	-4.0	1.4	-0.2
YMODEM (Procomm)	-1.8	0.5	-0.3
XMODEM-CRC (DSZ)	-86.7	0.0	37.4
YMODEM (DSZ)	-78.9	0.0	-32.1
ZMODEM (DSZ)	-33.8	1.0	-13.0

[1]1000 byte data packets.

Table 2. Percentage Difference of Measured Utilization Relative to Theoretical Utilization For Uniform Byte Error Distributions Produced By BLACK10

In the first set of experiments, the effective throughput of only the MS-Kermit 2.32/A implementation of Kermit and the DSZ 11-14-89 implementation of the XMODEM, YMODEM, and ZMODEM protocols were measured. As shown in Table 2, the measured throughput of the Kermit protocol using classic packets and 1000 byte extended length data packets, was between -4% to +2% of the throughput predicted by the theoretical equations at all tested baud rates and error rates. However, the measured throughput of the XMODEM, YMODEM, and ZMODEM protocols decreased faster than predicted as the error rate increased. Since the general shape of the percentage difference graphs, Graphs 16, 18, 20, and 22, were similar for each of the protocols at all baud rates, it was suspected that the DSZ implementation was adversely affecting the throughput of the protocols as the error rate increased.

Attempts were made to locate alternate implementations of the XMODEM, YMODEM, and ZMODEM protocols in order to confirm this supposition. One of the alternate implementations that was tested was PCZ 2.11.89. Unfortunately, it failed to function over a null modem cable connecting two RS-232 serial ports. A second alternative, DGTERM, functioned in the experiment's system configuration, but aborted when an error was detected during a file transfer.

Although Procomm-Plus 1.1B only implemented the XMODEM and YMODEM protocols, it did function in the experimental configuration. In addition, as shown in Table 2, the measured throughput of the XMODEM protocol was between -4% to 2% of the theoretical value and the YMODEM protocol was within ±2% for all error rates at 4800, 9600, and 19200 baud. Thus, it was concluded that the DSZ implementation was restricting the throughput as the error rate increased.

3.5.3 Random Bit Error Distribution (BLACK30) Experiments

The BLACK30 error generator program generated randomly distributed bitwise errors. In order to ensure a random and uniform error distribution, the random number generator used must have a period that is much greater than the required number of random numbers needed for a single throughput test. Thus, the random number generator selected was

$$Random_{N+1} = (Random_N * 4,826,809) \text{ MOD } (2^{31} - 1) \qquad \textbf{Eq. 29}$$

Figure 18. BLACK30 Random Number Generator

This random number generator is identical to the one used by the PRAND routine described on page 12 of "UBC Random" [12]. It has a period of $2^{31} - 2 = 4,294,967,293$ and "all bits in the unnormalized fraction are equally random"[2]. Unfortunately, this generator requires 32 bit arithmetic and without a floating point coprocessor on a 12 MHz 286, the error generator could not process the incoming data fast enough to handle baud rates greater than 9600 baud

A random number was generated for each bit of the received byte. If the random number was less than the product of the generator's period and the desired bit error rate, the bit was inverted. After all bits had been processed, the possibly modified byte was transmitted on the serial port different from the one the original byte was received.

The throughput experiments were performed on the Kermit protocol using classic and extended length data packets, the XMODEM-CRC protocol, and the YMODEM protocol

---

2. UBC Random [12], page 3

at 9600 and 4800 baud. The experiments were performed at error rates of 0.00E-5, 3.00E-5, 4.60E-5, 6.10E-5, 7.60E-5, 9.15E-5, 1.05E-4, 1.50E-4, 2.10E-4, 2.55E-4, 3.00E-4 errors/bit using a one megabyte test file. The results of these experiments are summarized in Table 3 which lists the minimum, maximum, and average percentage difference for all combinations of baud rates and error rates for each protocol.

PROTOCOL	MINIMUM	MAXIMUM	AVERAGE
Classic Kermit	-2.9	3.2	-0.1
Extended Kermit[1]	-1.3	3.6	0.5
XMODEM-CRC	-2.4	0.2	-0.8
YMODEM	-2.1	0.3	-0.5

[1]1000 byte data packets.

Table 3. Percentage Difference of Measured Utilization Relative to Theoretical Utilization For Random Bit Error Distributions Produced by BLACK30

As shown by the graphs in Appendix 8, all but two measurements were within ±3% of the theoretical throughput values. These anomalies, however, were between 3% and 4% of the theoretical throughput values.

3.6 SUMMARY

The results of the throughput experiments indicate that the equations that were developed are quite accurate in predicting the throughput of the stop and wait protocols that were tested. The experiments performed using the BLACK10 and BLACK30 error generator programs showed that the equations were capable of predicting the throughput of the protocols tested within ±5% for uniform deterministic error distributions and random poisson error distributions.

It was disheartening to discover that the widely used DSZ implementation of the XMODEM, YMODEM and ZMODEM protocols exhibited such poor performance under error rates greater than 1/2500 errors/byte. It was hypothesized that DSZ's internal logic was attempting to maximize throughput by adjusting packet sizes, transmission rates, and timing parameters when the number of detected errors exceed an internal limit. Unfortunately, the logic may have assumed that the communication medium was a packet switched network since that is what the ZMODEM protocol was designed for.

Upon further examination of the result graphs, it can be seen that a stop and wait protocol's behaviour is essentially unaffected by the baud rate other than being scaled by a constant factor. However, the relationship between the baud rate and the throughput is not a linear function. This was expected since the throughput equations are nonlinear functions of the baud rate.

The theoretical graphs indicate that for error rates less than 1.00E-5 errors/bit, the YMODEM and ZMODEM protocols theoretically have the greatest throughput. However, as the error rate increases, the XMODEM protocol has a higher throughput since its smaller packet size increases the probability that it is transmitted correctly and if it is corrupted, less channel bandwidth is expended on error recovery.

The throughput of the Kermit protocol using classic length packets and 1000 byte extended length packets show that the encoding overhead, as expected, is a major factor in determining the effective throughput. Of the protocols examined, it has the largest encoding overhead. Consequently, although its packet sizes are comparable to those of the XMODEM and YMODEM protocols, the throughput of the Kermit protocol using classic and 1000 byte extended length packets is significantly less than those of the XMODEM and YMODEM protocols, respectively.

3.7 CONCLUSION

The results of the throughput experiments allow several conclusions to be drawn. The most important conclusion is that the set of equations derived to predict the channel utilization, and hence throughput, of stop and wait protocols under random error distributions is valid. Unfortunately, no conclusion can be drawn regarding the set of equations derived regarding streaming protocols since there was no implementation that could be used to verify the correctness of the equations.

Furthermore, there is no single packet size or error recovery mechanism that can maximize channel utilization under different line conditions. If the error rate is low, a larger packet size is desirable. However, as the error rate increases, the cost of error recovery is reduced by using smaller packets. In addition, when the channel bandwidth occupied by the turn around time, propagation delay or timeout period is large, streaming protocols offer a significant throughput advantage over stop and wait protocols since no channel bandwidth is wasted waiting for an acknowledgement packet.

4. MAX2 PROTOCOL

4.1 INTRODUCTION

The MAX2 protocol was designed as a data link level protocol to support a large variety of communication channels while maximizing channel utilization under various line conditions. It combines many of the features and capabilities that are available in public domain protocols while adding several of its own unique capabilities.

In order to maximize the effective data throughput, the MAX2 protocol was designed as a streaming protocol using selective retransmission. In addition, it performs no data encoding except for SOH byte stuffing to minimize the encoding overhead. Furthermore, the MAX2 protocol calculates its timeout value based upon measurements taken during the connection phase to minimize the channel bandwidth wasted by an excessively long timeout period. The MAX2 protocol also supports bidirectional data transfer over full-duplex channels. This allows data to be transferred in both directions simultaneously.

However, a streaming protocol with large data packets at high baud rates may require more resources than a system can provide. If this is the case, the MAX2 protocol can also operate in a stop and wait or sliding windows mode and use data packets that are as small as 16 bytes. Furthermore, since some communication channels do not transmit all data values transparently, the MAX2 protocol offers several levels of data encoding. Each successive encoding level encodes a greater set of data values at the expense of increased protocol overhead. If the approproate level of data encoding is selected, data will be transmitted over the communications channel transparently with minimal overhead.

4.2 SPECIFICATION

The MAX2 protocol offers the user a choice of many encoding and checksum algorithms. These algorithms may be used together in any combination without altering the packet format or protocol.

4.2.1 Encoding Algorithms

The MAX2 protocol offers several encoding algorithms that may be selected by the operator or application. Each algorithm performs a different level of encoding. However, as the encoding level increases, so does the overhead. Thus, in order to maximize the effective throughput, the minimal level of encoding should be selected that will still allow data to be successfully exchanged between two systems.

4.2.1.1 Hex Encoding

The Hex Encoding algorithm converts binary data into upper case ASCII character representations of its hexadecimal values. Currently, this encoding algorithm is used to convert numeric field values of selected packets into printable ASCII characters. Hex Encoded values are transmitted in most to least significant digit order.

The following byte sequence

Byte 1	Byte 2	Byte 3	Byte 4
17h	B3h	4Fh	A0h

is Hex Encoded as "A04FB317"

Figure 19. Example of MAX2 Hex Encoding

4.2.1.2 Network Control Character Encoding

The Network Control Character Encoding algorithm prefix encodes a subset of control characters. These characters, values 3, 4, 16, 17, 19, and 127, are converted into pairs of printable ASCII values by XOR'ing them with the value 64 and then prefixing them with the printable control prefix character. Optionally, the values 131, 132, 144, 145, 147, and 255, can be encoded as well.

4.2.1.3 Full Control Character Encoding

The Full Control Character Encoding algorithm prefix encodes all control characters. These characters, values 1 to 31 and 129 to 255, are converted into pairs of printable ASCII values by XOR'ing them with the value 64 and then prefixing them with the printable control prefix character.

4.2.1.4 Eighth Bit Character Encoding

The Eighth Bit Character Encoding algorithm prefix encodes all characters whose eighth bit is set. These characters are converted into pairs of seven bit ASCII values by setting their eighth bit to zero and then prefixing them with the printable control prefix character.

4.2.2 Checksum Types

The MAX2 protocol supports several checksum algorithms. As shown in Table 4, there are two major classes of algorithms. Bytewise arithmetic checksums, Types 0-3, are offered in three sizes: one byte, two bytes, and four bytes. The calculation of the checksum is

performed using unsigned modular arithmetic. In the case of the Type 0 checksum, it is Hex Encoded.

TYPE	VALUE	SIZE	ALGORITHM
0	"0"	2	Hex Encoded one byte bytewise arithmetic sum
1	"1"	1	one byte bytewise arithmetic sum
2	"2"	2	two byte bytewise arithmetic sum
3	"3"	4	four byte bytewise arithmetic sum
4	"4"	2	CRC-16 checksum
5	"5"	4	CRC-32 checksum

NOTE 1: Algorithms are listed in increasing order of complexity
NOTE 2: All integer values are transmitted in least to most significant byte order.

Table 4. MAX2 Checksum Types and Field Values

In addition to the bytewise checksums, the MAX2 protocol also supports Cyclic Redundancy Check (CRC) checksums. Currently, the user has a choice between the two byte CRC-16 checksum, Type 4, and the four byte CRC-32 checksum, Type 5.

4.2.3 Packet Format

The format of a MAX2 packet is shown in Figure 20.

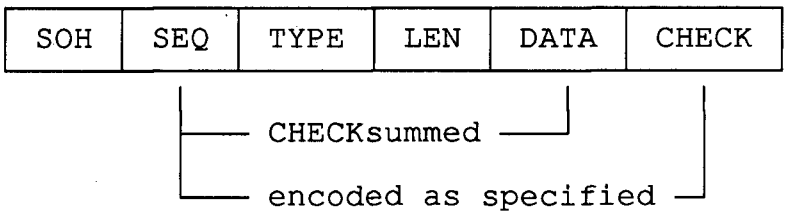


Figure 20. MAX2 Packet Format

4.2.3.1 SOH Field

The SOH field is a one byte value used to indicate the start of a MAX2 packet. It is assigned the value of 01H and cannot be modified. If this value occurs elsewhere in a MAX2 packet, is represented by two SOH characters.

4.2.3.2 SEQ Field

The SEQ field contains either a one or two byte unsigned sequence number. The size of these sequence numbers is negotiated during the connect phase. Two byte sequence numbers are transmitted in least to most significant byte order.

4.2.3.3 TYPE Field

The TYPE field is a single byte value that uniquely identifies the purpose of the packet. For further details, please refer to the section on packet types.

4.2.3.4 LEN Field

The LEN field is either a one or two byte unsigned integer value that specifies the number of bytes of data in the DATA field. A two byte LEN field is used unless the

maximum packet size negotiated is 128 bytes or less. Two byte LEN field values are transmitted in least to most significant byte order.

4.2.3.5 DATA Field

The DATA field is a varying length field that contains user data. The length of the field is indicated by the LEN field of the packet. The maximum size of this field is constrained by the maximum packet size which is determined during the connect phase.

4.2.3.6 CHECK Field

The CHECK field contains the checksum of the contents of the entire packet with the exception of the SOH and the CHECK fields. Its length is determined by the checksum type that is negotiated during the connect phase.

4.2.4 Packet Types

The MAX2 protocol defines eighteen (18) packet types which are listed and summarized in Table 5. These packets can be classified as either a Control or Data packet. Control packets are assigned sequence numbers from the control sequence number stream and Data packets are assigned sequence numbers from the data sequence number stream. All packets are transmitted and delivered in first in, first out order (FIFO). However, Control packets are transmitted and delivered ahead of any Data packets in the queue.

PACKET TYPE	TYPE FIELD VALUE	CONTAINS DATA	PACKET CLASS	DELIVERY MODE
ABORT	A	y	control	expedited
ABORT-A	a	y	control	expedited
ACK	Y	n	data	normal
ACK-C	y	n	control	expedited
CHANNEL	c	n	data	normal
CONNECT	C	y	data	normal
DATA-BLOCK	B	y	data	normal
DATA-REPLY	R	y	data	normal
DATA-STREAM	S	y	data	normal
DISCONNECTED	d	n	data	normal
DISCONNECTING	D	n	data	normal
FEEDBACK	F	y	control	expedited
NAK	N	n	data	normal
NAK-A	x	n	control	expedited
NAK-C	n	n	control	expedited
PAUSE	p	n	control	expedited
PROBE	?	n	control	expedited
RESUME	r	y	control	expedited
TIMEOUT	T	y	control	expedited

Table 5. MAX2 Packet Types

When a packet's delivery is expedited, it is inserted into the send or receive queue ahead of all normal delivery packets. However, if there are expedited delivery packets already in the queue, it is inserted after the newest expedited delivery packet. A detailed description of the packets and their purpose can be found in the sections that follow.

4.2.4.1 ABORT Packet

The ABORT packet is used to indicate an abnormal termination of the connection. The sender of this packet should place a brief ASCII string in the DATA field of the packet explaining to the operator on the remote system the reason for the abnormal termination. Once the ABORT packet has been sent, the sending system may immediately disconnect.

4.2.4.2 ABORT-A Packet

The ABORT-A packet is a version of the ABORT packet that contains only printable ASCII characters. Its LEN field is Hex Encoded, its SEQ field is set to the value 32, and it uses a Type 0 checksum. It is intended to be used to terminate the establishment of a connection if any of the connection parameters specified by the remote system are not acceptable.

4.2.4.3 ACK Packet

The ACK packet is used to indicate that the Data packet sent by the remote system was received with no detectable transmission errors. The sequence number of the ACK packet is set to the same value as the received Data packet.

4.2.4.4 ACK-C Packet

The ACK-C packet is used to indicate that the Control packet sent by the remote system was received with no detectable transmission errors. The sequence number of the ACK-C packet is set to the same value as the received Control packet.

4.2.4.5 CHANNEL Packet

The CHANNEL packet is only used if the data channel is unidirectional. It is used to determine which system is allowed to transmit data. When the sending system has no data to send or needs data from the receiving system, it transmits the CHANNEL packet, relinquishing control of the data channel. Once the receiving system correctly receives the CHANNEL packet and acknowledges it, the roles of the two systems reverse and the new sending system may begin sending.

4.2.4.6 CONNECT Packet

The CONNECT packet is used to negotiate the connection parameters. The CONNECT packet, with its DATA field set to indicate the desired connection parameters, is sent by the calling system to the remote system. Upon receiving a CONNECT packet, the remote system replies with its own CONNECT packet to indicate to the calling system its connection parameters. The lowest common denominator is then used for the connection.

Since the CONNECT packets are exchanged before the connection parameters have been determined, the entire packet is comprised of only printable ASCII characters. This ensures that the packet will be transmitted unaltered through seven bit data lines. Consequently, the LEN and CHECK fields are Hex Encoded and stored in most to least significant digit order. In addition, the SEQ field of these CONNECT packets is equal to the ASCII zero character and the checksum is a Type 0 checksum.

The contents and structure of the CONNECT packet's DATA field is shown in Figure 21 while Table 6 summarizes the function of each of the fields.

PSIZE	BSIZE	SSIZE	CHECK	ENCODE	CTL	8TH	DUPLEX

Figure 21. MAX2 CONNECT Packet Data Field Format

FIELD	BYTES	FUNCTION
PSIZE	4	maximum packet size
BSIZE	1	maximum buffer size measured in PSIZE bytes
SSIZE	1	size of sequence numbers
CHECK	1	checksum type
ENCODE	1	encoding algorithm to use
CTL	1	control prefix encoding character
8TH	1	eighth bit prefix encoding character
DUPLEX	1	indicates half or full duplex
TWO-WAY	1	indicates if data can flow in both directions simultaneously

Table 6. MAX2 CONNECT Packet Data Field Types

4.2.4.6.1 PSIZE Field

The PSIZE field indicates the maximum packet size (<= 65535 bytes), that may be used. The packet size is Hex Encoded and stored in most to least significant digit order. The maximum packet size of the connection shall be the smaller of the two PSIZE values. However, the minimum packet size must be at least equal to sixteen (16) bytes.

4.2.4.6.2 BSIZE Field

The BSIZE field indicates the amount of buffer space, PSIZE * BSIZE bytes, that is available when receiving. If there is unlimited buffer space, this field is set to zero. The size of the buffer space is used to determine the window size when operating in sliding window mode. Like the PSIZE field, this field is Hex Encoded and is stored in most to least significant digit order.

4.2.4.6.3 SSIZE Field

The SSIZE field indicates the size of the sequence numbers to be used. Valid values for this field are the ASCII values "1" and "2" indicating one and two byte sequence numbers, respectively. The size of the sequence numbers of the connection shall be the larger of the two SSIZE values. If one of the systems is not capable of supporting the requested sequence number size, it must ABORT the connection. Two byte sequence numbers are recommended when running in streaming mode or sliding windows mode with a large window size.

4.2.4.6.4 CHECK Field

The CHECK field indicates the type of checksum algorithm to be used. The algorithms that are supported and their values are listed in Table 4. The checksum algorithm of the connection shall be the more complex of the two algorithms specified. If one of the systems is not capable of supporting the checksum algorithm, it must ABORT the connection.

4.2.4.6.5 ENCODE Field

The ENCODE field is used to indicate the level of encoding that is required to transmit eight bit data over the current communications line correctly. As shown in Table 7, there are several encoding algorithms. These encoding algorithms may be combined by placing Hex Encoded sum of their corresponding values in the ENCODE field. If the encoding algorithms are combined, the algorithms shall be performed in descending order according to their values listed in Table 7. Finally, the encoding algorithm(s) used on the connection shall be the algorithm(s) that correspond to the higher ENCODE value. If one of

the systems is not capable of supporting the encoding algorithm, it must ABORT the connection.

VALUE	DESCRIPTION
0	No encoding (except SOH stuffing) is to be performed.
1	Network Control Character Prefix Encoding
2	Network Control Character Prefix Encoding (including optional characters)
4	Full Control Character Prefix Encoding
8	Eighth Bit Character Prefix Encoding

Table 7. MAX2 ENCODE Field Values

4.2.4.6.6 CTL Field

The CTL field is used to indicate to the remote system the printable ASCII character that should be used as the control prefix encoding character. If this character is encountered in the data and control character prefix encoding is being performed, it shall be represented using two such characters. In addition, this character cannot be equal to the eighth bit prefix encoding character. The default value is ASCII 94, the caret character (^).

4.2.4.6.7 8TH Field

The 8TH field is used to indicate to the remote system the printable ASCII character that should be used as the eighth bit prefix encoding character. If this character is encountered in the data and eighth bit character prefix encoding is being performed, it shall be represented using two such characters. In addition, this character cannot be equal to the control prefix encoding character. The default value is ASCII 96, the accent grave character (`).

4.2.4.6.8 DUPLEX Field

The DUPLEX field is used to indicate whether the system is capable of sending and receiving simultaneously. A value of "H" indicates that the system is only capable of half-duplex communication; otherwise it is capable of full-duplex communication. A value of "U" indicates that the data channel is unidirectional while a value of "B" indicates that the channel is bidirectional. If the data channel is unidirectional, it is controlled by the sending system. Until the sending system relinquishes control, the receiving system can only receive data. However, with a bidirectional data channel, both systems can send data simultaneously.

4.2.4.7 DATA-BLOCK Packet

The DATA-BLOCK packet is used to transmit data to the remote system. The local system will then wait for the remote system to acknowledge the packet before proceeding. However, the remote system will not send an ACK for the DATA-BLOCK packet until all previously sent DATA packets have been received correctly. The DATA-BLOCK packet will be retransmitted only if it was NAK'ed by the remote system. Thus, if a DATA-STREAM packet was sent before the DATA-BLOCK packet and was NAK'ed, the remote system would not send an ACK for the DATA-BLOCK packet until the DATA-STREAM packet was retransmitted with no detectable transmission errors. If no communication is received from the remote system before the timeout period expires, a PROBE packet should be sent to determine the status of the remote system.

4.2.4.8 DATA-REPLY Packet

The DATA-REPLY packet is used to transmit data to the remote system. It indicates to the remote system that a reply, ACK or NAK, is required. This would be the DATA packet to be used in sliding window mode or in streaming mode if the transmission medium was error prone. If no reply is received before the timeout period expires, the DATA-REPLY packet is resent unless the application set retry limit is exceeded.

4.2.4.9 DATA-STREAM Packet

The DATA-STREAM packet is used to transmit data to the remote system. It indicates to the remote system that if the packet was received with no detectable transmission errors, a reply is not required. This would be the normal DATA packet used in streaming mode.

4.2.4.10 DISCONNECTED Packet

The DISCONNECTED packet is used by the receiving system to indicate to the sending system that it has finally received all of the data that was sent to it. This event occurs when the sequence number of a DATA packet is one less than the sequence number of the DISCONNECTING packet that was sent by the sending system. When the sending system receives the DISCONNECTED packet, it sends an acknowledgement to the receiving system and terminates the connection.

4.2.4.11 DISCONNECTING Packet

The DISCONNECTING packet is used by the sending system to indicate to the receiving system that it has completed its transfer of data. Like any other packet, it is

acknowledged by the receiver. However, before the sending system can disconnect, it must wait for the receiving system's DISCONNECTED packet. If the data channel is unidirectional, the DISCONNECTING packet can only be sent when the system has the CHANNEL packet.

4.2.4.12 FEEDBACK Packet

The FEEDBACK packet is used to transmit application level control data to the remote application. Upon receiving a FEEDBACK packet, the contents of the DATA Field are interpreted and the appropriate action taken by the application. In a file transfer application, the FEEDBACK packet could be used to send a cancel request to the remote system.

4.2.4.13 NAK Packet

The NAK packet is used to indicate that the Data packet sent by the remote system was received with detectable transmission errors and must be resent by the remote system. If the requested Data packet is not received within the timeout period, the NAK packet is resent. The sequence number of the NAK packet will be set to the same value as the received Data packet. There is no retry limit defined by the MAX2 protocol; it is determined by the application.

4.2.4.14 NAK-A Packet

The NAK-A packet is a version of the NAK packet that contains only printable ASCII characters. Its LEN field is Hex Encoded, its SEQ field is set to the value 32, and it uses a Type 0 checksum. It is intended only to be used prior to the establishment of a connection

when the encoding level and checksum type have yet to be determined. There is no retry limit defined by the MAX2 protocol; it is determined by the application.

4.2.4.15 NAK-C Packet

The NAK-C packet is used to indicate that the Control packet sent by the remote system was received with detectable transmission errors and must be resent by the remote system. The sequence number of the NAK-C packet will be set to the same value as the received Control packet. There is no retry limit defined by the MAX2 protocol; it is determined by the application.

4.2.4.16 PAUSE Packet

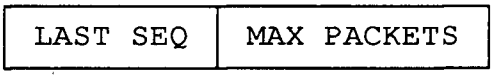
The PAUSE packet is used by the receiving system to ask the sending system to stop transmitting until it receives a RESUME packet. Typically, this would be sent when the receiving system is in danger of being overrun by the sending system in streaming mode.

4.2.4.17 PROBE Packet

The PROBE packet is used to determine whether a remote system is still alive. Upon receiving a PROBE packet, the system immediately acknowledges it.

4.2.4.18 RESUME Packet

The RESUME packet is used by the receiving system to tell the sending system that it may resume sending. The manner in which its DATA field is interpreted is shown in Figure 22. Both of these fields are two byte unsigned integers that are transmitted in least to most significant byte order.



LAST SEQ - sequence number of last DATA packet received correctly
MAX PACKETS - maximum number of packets of maximum size before overrun

Figure 22. MAX2 RESUME Packet Data Field Format

When the sending system receives the RESUME packet, it begins transmitting again starting with the packet whose sequence number is equal to "LAST SEQ + 1". The other parameter, "MAX PACKETS", can be used by the sending system to moderate the amount of data that is transmitted before it pauses to give the receiving system a chance to catch up. This reduces the amount of bandwidth wasted due to lost DATA packets and the transmission of the RESUME packet.

4.2.4.19 TIMEOUT Packet

The TIMEOUT packet is used to indicate to the receiver of the packet the timeout value chosen by the sender. This value is a one byte unsigned integer in the DATA field. It indicates the number of seconds that the sender will wait for a reply before it retransmits a packet. This packet allows the MAX2 protocol to determine a suitable timeout period to use to minimize the channel bandwidth wasted due premature timeouts and idle periods waiting for the timeout period to expire in stop and wait mode.

4.2.5 Protocol Description

Although the MAX2 protocol was designed with full-duplex bidirection transmission capabilities, the description of the protocol will be given in one direction only. In the

description that follows, it is assumed that the calling system, the CALLER, wishes to transfer data to the called system, the CALLEE.

4.2.5.1 CONNECT Phase

The first step in the CONNECT phase is for the CALLER to establish a physical connection (i.e. telephone connection) to the remote system. Once the physical connection has been established, the MAX2 protocol starts with the CALLER sending a CONNECT packet, with its fields set appropriately, to the CALLEE. Until the CALLER receives a CONNECT packet from the CALLEE or the application determined retry limit is exceeded, it continues to transmit CONNECT packets at 5 second intervals.

If the CALLEE detects a transmission error in the CALLER's CONNECT packet, it can simply wait for the CALLER to retransmit the CONNECT packet. However, to speed up the error recovery process, the CALLEE can send a NAK-A packet to the CALLER.

If the CALLEE is capable of supporting the connection parameters specified in the CONNECT packet, it responds with its own packet to the CALLER. Its CONNECT packet specifies connection parameters that do not exceed the requirements of the CALLER's. However, if the CALLER's CONNECT packet specifies a requirement that the CALLEE is not capable of supporting (eg. eighth bit prefix encoding), it responds with an ABORT-A packet containing a short ASCII string explaining to the CALLER's system operator why the CONNECT request was declined.

The CALLER shall also record the elapsed time in seconds between the transmission of the last byte of a CONNECT packet and the reception of the first byte of a reply. This interval shall be referred to as T and is used to calculate the timeout interval when waiting

for a reply to a packet. The timeout interval calculated is placed in a TIMEOUT packet and sent to the CALLEE.

The timeout interval should be sufficient to allow the largest data packet to be sent and for the reply to be received by the sender. This timeout interval should also take into account the time required by the receiver to process the data packet and to generate the appropriate reply as well as the propagation delay. A simple calculation of the timeout interval can be performed using the following equation

$$\text{Timeout Interval} = P / B + T \qquad \textbf{Eq. 30}$$

where P = maximum data packet size in bytes
B = transmission speed in bytes/second

Similarly the CALLEE measures the elapsed time between sending the last byte of its CONNECT packet and receiving the first byte of the CALLER's TIMEOUT packet. After calculating its timeout period, it sends its timeout period to the CALLER in a TIMEOUT packet. Once the CALLER acknowledges this packet, the connect phase is complete. If the data channel is unidirectional, it is controlled by the CALLER.

4.2.5.2 DATA Transfer Phase

During the data transfer phase, any of the three DATA packets may be transmitted. If the data channel is bidirectional, the CALLER and CALLEE may transmit DATA packets simultaneously. However, acknowledgements are not piggy-backed. This makes the implementation of the MAX2 protocol much simpler. If the data channel is unidirectional, the DATA packets can only be sent by the system holding the CHANNEL packet.

A sliding window mode of operation can be achieved by using DATA-REPLY packets in conjunction with the BSIZE field of the CONNECT packet or the MAX PACKETS field of a RESUME packet. If a RESUME packet has been received, the window size shall be the smaller of BSIZE*PSIZE and MAX PACKETS*PSIZE. The sending system simply transmits DATA-REPLY packets as long the packets can be inserted into the window. When the the oldest packet in the window is acknowledged, it is removed.

In the normal streaming mode, DATA-STREAM packets are continuously transmitted. If the receiving system detects a transmission error, a NAK packet is sent to the sending system, requesting that the corrupted DATA-STREAM packet be retransmitted.

However, if the sending system has overrun the receiving system, the receiving system can send a PAUSE packet to temporarily halt the sending system's flood of DATA packets. Once the receiving system has freed enough of its buffers to accept DATA packets again, it sends a RESUME packet to the sending system, indicating the sequence number of the last packet that it correctly received. Naturally, NAK packets will be sent for any packets that were received with transmission errors.

If the sending system repeatedly overruns the receiving system, a large amount of bandwidth may be wasted on overrun packets. Bandwidth would also be wasted while the sending system is waiting for the RESUME packet before resuming transmission. The sending system may be able to increase the channel utilization by changing to sliding window mode or adjusting the size of the window based upon the MAX PACKETS field of the RESUME packet.

In addition, if the sending system has been PAUSE'd and has not received any communication from the receiving system for a period greater than or equal to its timeout period, it should send a PROBE packet to the receiving system. If the receiving system fails

to reply to the PROBE within the timeout period, the PROBE is resent. If the retry limit set by the application is exceeded, the sending system should assume the receiving system has become hung and it should ABORT the connection.

At times, it may be necessary for both systems to become fully synchronized. This may be achieved by sending a DATA-BLOCK packet. Upon receiving this packet, a positive acknowledgement for this packet will not be sent to the sending system until all outstanding DATA packets have been received correctly. The sending system shall resend a DATA-BLOCK only if it receives a negative acknowledgement for the packet.

During the exchange of DATA packets and acknowledgements, if either system deems it necessary to modify its timeout parameter, it may do so. However, it must notify the remote system of its new timeout value by using the TIMEOUT packet. Moreover, either system may also choose to send less than the maximum amount of data per DATA packet to minimize the cost of error recovery.

4.2.5.3 DISCONNECT Phase

Once the CALLER has sent all of its data, it sends a DISCONNECTING packet. However, the CALLEE should not reply with a DISCONNECTED packet unless it has received all data packets whose sequence number is less than the sequence number of the CALLER's DISCONNECTING packet. As soon as the CALLER receives a DISCONNECTED packet from the CALLEE, the connection is terminated. This disconnect sequence must be performed in both directions before the physical link is terminated.

If the data channel is unidirectional, the CALLER sends the CHANNEL packet to the CALLEE immediately after the DISCONNECTING packet. This allows the CALLEE to begin sending data or to initiate its own disconnect sequence.

4.3 IMPLEMENTATION

In order to measure the performance of the MAX2 protocol, two programs were written in ANSI C and compiled using Borland's Turbo C++ compiler. The MAX2SEND program would use a subset of the MAX2 protocol to send a file, while the MAX2RECV program would receive the file sent. Both programs assume a full-duplex channel, a unidirectional data channel, use CRC-16 packet checksums, use two byte sequence numbers, and perform no data encoding. Consequently, the theoretical channel utilization is given by Equation 20. In addition, since only a subset of the protocol's features are required, the prototype only recognizes the packets listed in Table 8.

ABORT	ABORT-A	ACK	ACK-C
CONNECT	DATA-STREAM	DISCONNECTED	DISCONNECTING
NAK	NAK-A	NAK-C	TIMEOUT

Table 8. MAX2 Packet Types Recognized By MAX2SEND and MAX2RECV

Since the two programs transmit a file in streaming mode, a mechanism was needed to limit the system resources consumed as the error rate increased. Both programs maintain a retry counter for each packet sent. When a packet is added to the program's send queue, it is inserted in front of all packets whose retry counter is lower but is inserted after any packets whose retry counter is greater than or equal to its retry counter. This allows older packets to be resent before newer ones and hence minimizes the memory requirements of both programs.

Both the MAX2RECV and MAX2SEND programs accept command line parameters. These parameters allow the user to optionally specify the baud rate, communications port, retry limit, and maximum data packet size for the file transfer. In addition, the MAX2RECV

program requires the user specify the name of the file to be received and the MAX2SEND program requires the user specify the name of the file to be sent.

Upon completion of the file transfer, both programs display the number of bytes sent and received, packets sent and received, NAKs sent and received, errors detected, timeouts, the size of the file transferred, and the elapsed time of the file transfer. The programs also compute and display the effective throughput and the channel utilization.

The robustness of the programs developed was tested by sending various file types through the BLACK30 error generator at an error rate of 0.00E-0 and 3.00E-4 errors/bit. Upon completion of the file transfer, a checksum was computed for the received file and compared to the checksum of the original file. After successfully transferring the one megabyte test file, the program executable files, and the program source files, the programs were considered error free and robust enough for throughput testing.

4.4 THROUGHPUT EXPERIMENTS

A decision was made to test the MAX2 protocol under only random error distribution conditions using the BLACK30 error generator since the theoretical equations were so accurate for the protocols that were tested under uniform error distributions. Futhermore, experiences with communication over telephone lines have shown that a uniform error distribution is not realistic.

4.4.1 Random Bit Error Distribution (BLACK30) Experiments

The throughput of the MAX2 protocol prototype implementation was tested using the BLACK30 program as the error generator. Throughput experiments were performed using

the one megabyte test file at 9600 and 4800 baud using 1024 byte data packets at error rates of 0.00E-5, 3.00E-5, 4.60E-5, 6.10E-5, 7.60E-5, 9.15E-5, 1.05E-4, and 1.50E-4, errors/bit.

Experiments were also performed using 128 byte data packets at the same error rates as the 1024 byte data packets and at error rates of 2.10E-4, 2.55E-4, 3.00E-4 errors/bit. The results of these experiments are summarized in Table 10 which lists the minimum, maximum, and average percentage difference for all error rates for each data packet size.

PROTOCOL	MINIMUM	MAXIMUM	AVERAGE
128 byte packets	-0.6	0.2	-0.2
1024 byte packets	-1.4	0.3	-0.4

Table 9. MAX2 Percentage Difference of Measured Utilization Relative to Theoretical Utilization For Random Bit Error Distributions Produced by BLACK30

The average measured throughput of all protocols tested at random bit error rates of 0.00E-0, 4.50E-5, 9.15E-5, and 3.00E-4 is listed in Table 10. The throughput numbers in this table show that when using comparable data packet sizes, the MAX2 protocol achieves a higher effective data throughput than the Kermit, XMODEM, and YMODEM protocols. However, as the error rate increases to 4.50E-5 errors/bit, protocols such as XMODEM with a low encoding overhead and 133 byte data packets, provide a higher throughput than the MAX2 protocol using a 1024 byte data packet. A complete view of the throughput measurements taken is provided by Graphs 39 and 40.

PROTOCOL	0.00E+0 errors/bit	4.50E-5 errors/bit	9.15E-5 errors/bit	3.00E-4 errors/bit
At 4800 Baud:				
Classic Kermit	298.5	272.5	249.3	181.0
MAX2 (128 bytes)	451.5	431.7	410.5	330.5
XMODEM (Procomm)[2]	440.4	412.6	388.3	289.8
Extended Kermit[1]	354.9	245.6	171.6	N/A
MAX2 (1024 bytes)	473.9	324.9	222.5	N/A
YMODEM (Procomm)	466.7	317.5	219.6	N/A
At 9600 Baud:				
Classic Kermit	580.6	532.4	470.4	281.6
MAX2 (128 bytes)	903.0	860.7	821.3	665.3
XMODEM (Procomm)[2]	850.2	789.4	733.4	525.0
Extended Kermit[1]	698.6	483.7	339.8	N/A
MAX2 (1024 bytes)	947.7	949.4	447.2	N/A
YMODEM (Procomm)	913.9	621.0	420.2	N/A

[1]1000 byte data packets.
[2]CRC checksum

Table 10. MAX2 Average Measured Throughput Comparison At Random Bit Error Rates Produced by BLACK30

As shown by Graphs 41 and 42, the MAX2 protocol's measured throughput is within ±1.5% of the theoretical throughput for both data packet sizes and at all error rates. This is by far the best correlation between the measured throughput and the theoretical throughput predicted by the derived equations.

4.5 SUMMARY

The MAX2 protocol is a powerful, versatile, and configurable communications protocol. As summarized in Table 11, the MAX2 protocol offers several levels of data encoding, error detection mechanisms, flow control methods, and low packet overhead. However, Table 11 does not show MAX2's ability to dynamically adjust its timeout period

and packet size (within the negotiated maximum packet size), that it has a maximum packet size of 64 kilobytes, and its ability to support data transfer in both directions simultaneously. The MAX2 protocol also allows the user or application to select the features that would be used during a communications session.

As with any streaming protocol using a selective retransmission mechanism for error correction, the MAX2 protocol could conceivably consume vast amounts of memory buffering data packets while waiting for corrupted data packets to be retransmitted correctly. However, the priority send queue and transmission mechanism used by the prototype implementation kept the MAX2RECV program's packet buffer requirements to within four data packets during the throughput experiments.

FEATURE	MAX2 (128)	MAX2 (256)	MAX2 (512)	MAX2 (1024)
FLOW CONTROL:				
· Stop and Wait	o	o	o	o
· Sliding Windows	o	o	o	o
· Streaming	✔	✔	✔	✔
ERROR DETECTION:				
· 1 byte sum	✔	✔	✔	✔
· 2 byte sum	✔	✔	✔	✔
· 16 bit CRC	✔	✔	✔	✔
· 32 bit CRC	✔	✔	✔	✔
ERROR RECOVERY:				
· selective retransmission	✔	✔	✔	✔
· go back N	✗	✗	✗	✗
PACKET SIZES[1]:				
· Data Packet (Min)	7	7	8	8
· Data Packet (Max)	128	256	512	1024
· Reply Packet	7	7	8	8
· Max. Data Field	121	249	504	1016
OVERHEAD:				
· Bytes Encoded[2]	1	1	1	1
· Encoding Factor	1.004	1.004	1.004	1.004
· Raw Data / Packet	120.5	248.0	510.0	1012.0
· Overhead To Send One Data Packet[3]	6.2%	3.2%	2.0%	1.2%

✔ - supported
o - supported but not normally used
✗ - not supported
1 - based on 16 bit CRC checksum
2 - assuming minimum encoding on stream of 0 to 255
3 - 1 - (Data Packet) / Raw Data

Table 11. MAX2 Protocol Feature Summary

The throughput experiments with the MAX2 prototype have shown that the protocol is robust and its performance degrades gracefully as the communication channel's error rate

increases. Furthermore, as shown by the graphs in Appendix 6, the MAX2 protocol provides higher effective throughputs at all error rates than the Kermit, XMODEM, YMODEM, and ZMODEM protocols.

4.6 CONCLUSION

The prototype implementation of the MAX2 protocol has shown that the protocol is feasible, robust, and more efficient than the Kermit, XMODEM, YMODEM, and ZMODEM protocols. In addition, the throughput experiments have validated the channel utilization equation for full-duplex streaming protocols using selective retransmission error correction.

The results of the throughput experiments and the channel utilization equation have shown that as the error rate increases, the data packet size should be decreased to minimize the error recovery cost and hence increase the throughput. Since the MAX2 protocol allows variable length data packets within the negotiated maximum packet size, an implementation of the MAX2 protocol could maximize its effective throughput by monitoring the error rate of the communications channel and making the appropriate adjustments to the size of the data packet.

5. CONCLUSION

A number of public domain file transfer protocols were surveyed. An analysis of these protocols showed that the Kermit protocol using classic packets had the highest overhead of 50.0% while the YMODEM protocol had the lowest overhead of 0.6%. The overhead and packet sizes of these protocols as well as the MAX2 protocol is listed in Table 12.

PROTOCOL	DATA PACKET SIZE	REPLY PACKET SIZE	PROTOCOL OVERHEAD (%)
Classic Kermit	97	8	50.0
Extended Kermit	1033	8	29.5
XMODEM-CRC	133	1	4.7
YMODEM	1029	1	0.6
ZMODEM	1027	21	3.8
MAX2 (1024)	1024	8	1.2
MAX2 (512)	512	8	2.0
MAX2 (256)	256	7	3.2
MAX2 (128)	128	7	6.2

Table 12. Protocol Overhead Summary

A number of equations were derived to calculate the effective channel utilization, and hence the effective data throughput, of stop and wait protocols and streaming protocols for any given transmission speed and error rate. These utilization equations are expressed in terms of the probabilities of the six basic events. These probabilities are determined by the packet sizes and the error distribution. Currently, equations have been derived for uniform byte error and random bit error distributions.

However, no utilization equation was derived for sliding window protocols since the initial work that was performed indicated that it would be difficult to derive an equation that

would accurately model the behaviour of all sliding window protocols. Furthermore, before an equation can be derived for sliding window protocols, an implementation of a well documented sliding window protocol is needed in order to verify the validity of the equation.

The validity of the derived utilization equations was proven by performing a series of experiments that measured the effective data throughput of the Kermit, MAX2, XMODEM, and YMODEM protocols under different baud rates, error rates, and error distributions (uniform byte error and random bit error). The results of these experiments showed that the equations were within ±5% of the measured values. In addition, these experiments showed that the effective throughput of the DSZ implementation of the XMODEM, YMODEM, and ZMODEM protocols degrade rapidly as the error rate increases.

The utilization equations have shown that protocols with larger packets have higher channel utilization when the error rate is low. However, as the error rate increases, the channel utilization of protocols with smaller packets degrade at a slower rate and at some point offer better utilization than protocols with larger packets. In addition, the equations indicate that stop and wait protocols with fixed timeout intervals needlessly waste channel bandwidth as the transmission speed increases. Thus, in order to maximize channel utilization, a protocol should decrease its packet size as the error rate increases and decrease its timeout period at higher transmission speeds.

The protocol that was designed, MAX2, is a full-duplex streaming protocol that can also run in stop and wait or sliding windows mode. The MAX2 protocol also allows the user or application to select the level of data encoding, the checksum method, the maximum packet size, and the sequence number size. In addition, the MAX2 protocol determines the appropriate timeout value to use as part of its connection phase. Furthermore, unlike the

public domain file transfer protocols examined, MAX2 allows data to be transmitted in both directions simultaneously over a full-duplex communications channel.

However, the MAX2 prototype only implements a subset of the protocol. Nonetheless, the throughput experiments using the prototype implementation showed that the MAX2 protocol was robust and makes better utilization of the channel than any of the protocols examined. It also proved that the channel utilization equation derived for full-duplex streaming protocols using selective retransmission error correction was valid.

Since the prototype implementation was designed to transfer a file using a subset of the MAX2 protocol, the next step would be to develop a general purpose MAX2 protocol driver that can be used by any application to exchange information with a remote application. This would require that a set of services be defined that can be invoked by an application to establish a connection, to disconnect, to send data, and to receive data.

Currently, the MAX2 protocol has an error recovery problem shared by the Kermit, XMODEM, and YMODEM protocols. Namely, if the error rate increases to a level such that the probability that P(Success) approaches zero, these protocols will probably fail due to excessive retries before the data packet is successfully transmitted. In order to recover from this situation, the MAX2 protocol must provide a mechanism to partition or segment the packet such that each partition or segment has a reasonable chance of being transmitted successfully. An intelligent implementation of the MAX2 protocol would monitor the packet retry rate and increase the data packet size when the packet retry rate decreases and conversely, decrease the data packet size when the packet retry rate increases.

6. BIBLIOGRAPHY

[1] Uyless D. Black, *Data Communications, Networks, and Distributed Processing*, Reston Publishing Company, Inc., Reston, Virginia, 1983

[2] Frank da Cruz, *Kermit Protocol Manual, Sixth Edition*, Columbia University Centre for Computing Activities, New York, New York, Jun 1986

[3] J.A. Field, *Efficient Computer-Computer Communication*, Proceedings of the IEE, Volume 123, Number 8, August 1976, pages 756-760

[4] Chuck Forsberg, *XMODEM/YMODEM Protocol Reference*, Omen Technology Inc., Portland, Oregon, Feb 1988

[5] Chuck Forsberg, *The ZMODEM Inter Application File Transfer Protocol*, Omen Technology Inc., Portland, Oregon, Oct 1988

[6] Peter Grogono, Programming in PASCAL, *Revised Edition*, Addison-Wesley Publishing Company, Inc., Don Mills, Ontario, 1980

[7] Donald E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1968

[8] Donald E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Second Edition*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1981

[9] Richard J. Larsen and Morris L. Marx, *An Introduction to Mathematical Statistics and Its Applications*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981

[10] Herbert Maisel and Giuliano Gnugnoli, *Simulation of Discrete Stochastic Systems*, Science Research Associates, Inc., USA, 1972

[11] Leslie Jill Miller, *An Analysis of Link Level Protocols For Error Prone Links*, Proceedings, Seventh Data Communications Symposium, 1981

[12] Tom Nicol, *UBC Random*, Computing Centre, The University of British Columbia, Vancouver, B.C., May 1986

[13] William Stallings, *Data and Computer Communications, Second Edition*, Macmillan Publishing Company, New York, New York, 1988

[14] George W. Struble, *Assembler Language Programming: the IBM System/360 and 370, Second Edition*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1975

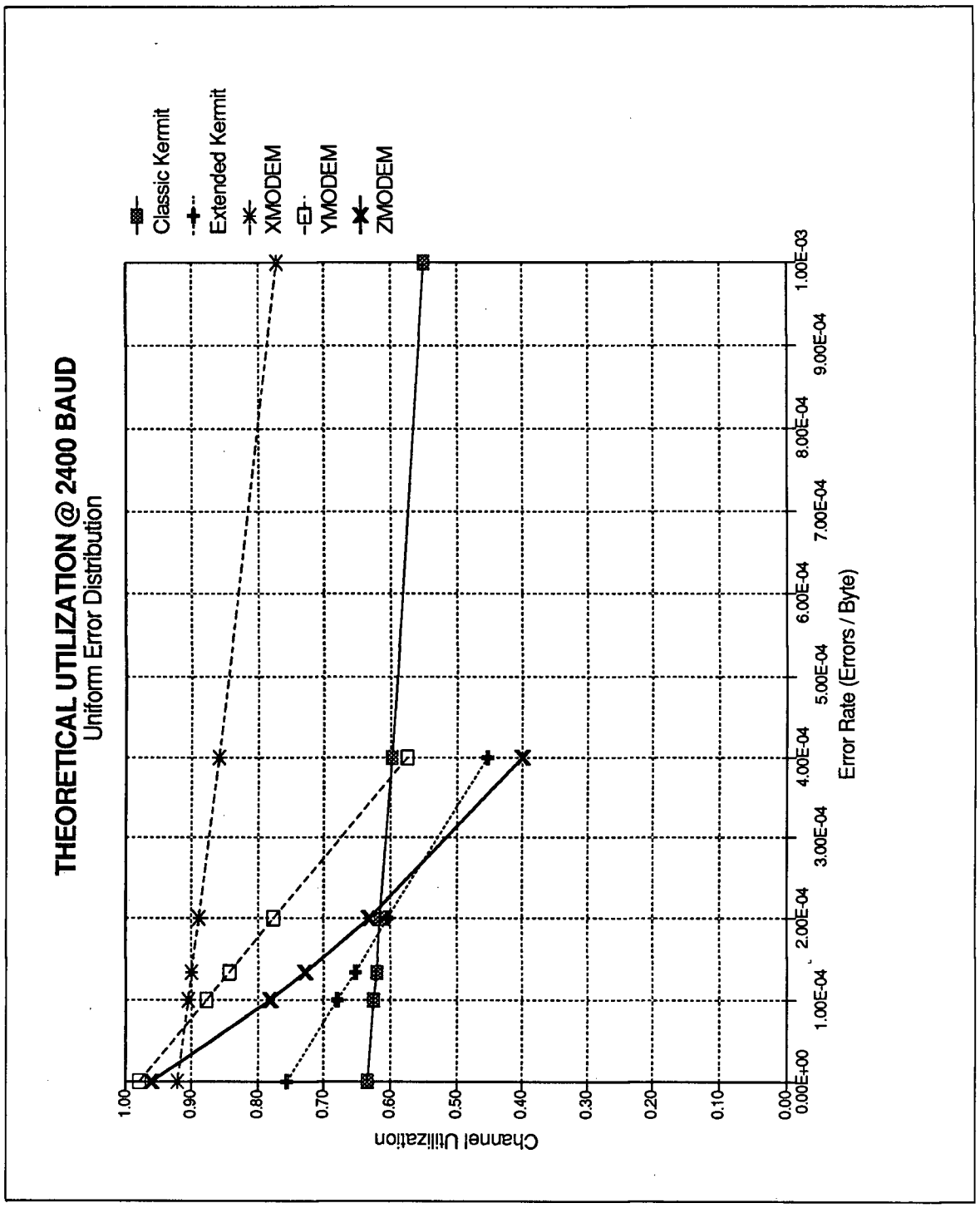
[15] Andrew S. Tanenbaum, *Computer Networks, Second Edition*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1988

[16] B. Zacharov, *Transmission Strategy and Optimal Block Size in High-speed Data Communication*, Proceedings of the IEE, Volume 120, Number 8, August 1973, pages 846-851

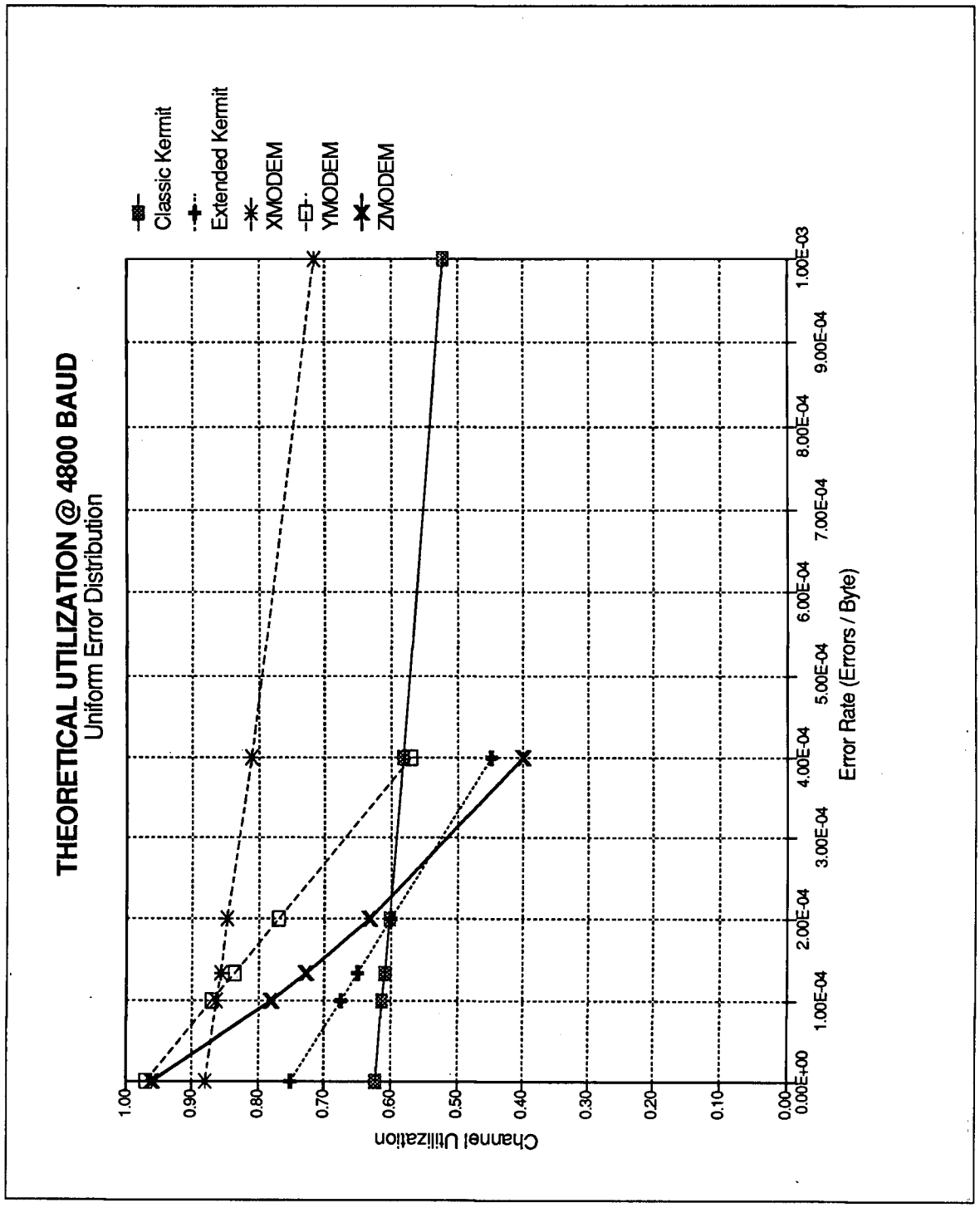
APPENDIX 1. Theoretical Utilization Graphs, Uniform Error Distribution

When examining the graphs in this appendix, please keep the following in mind:

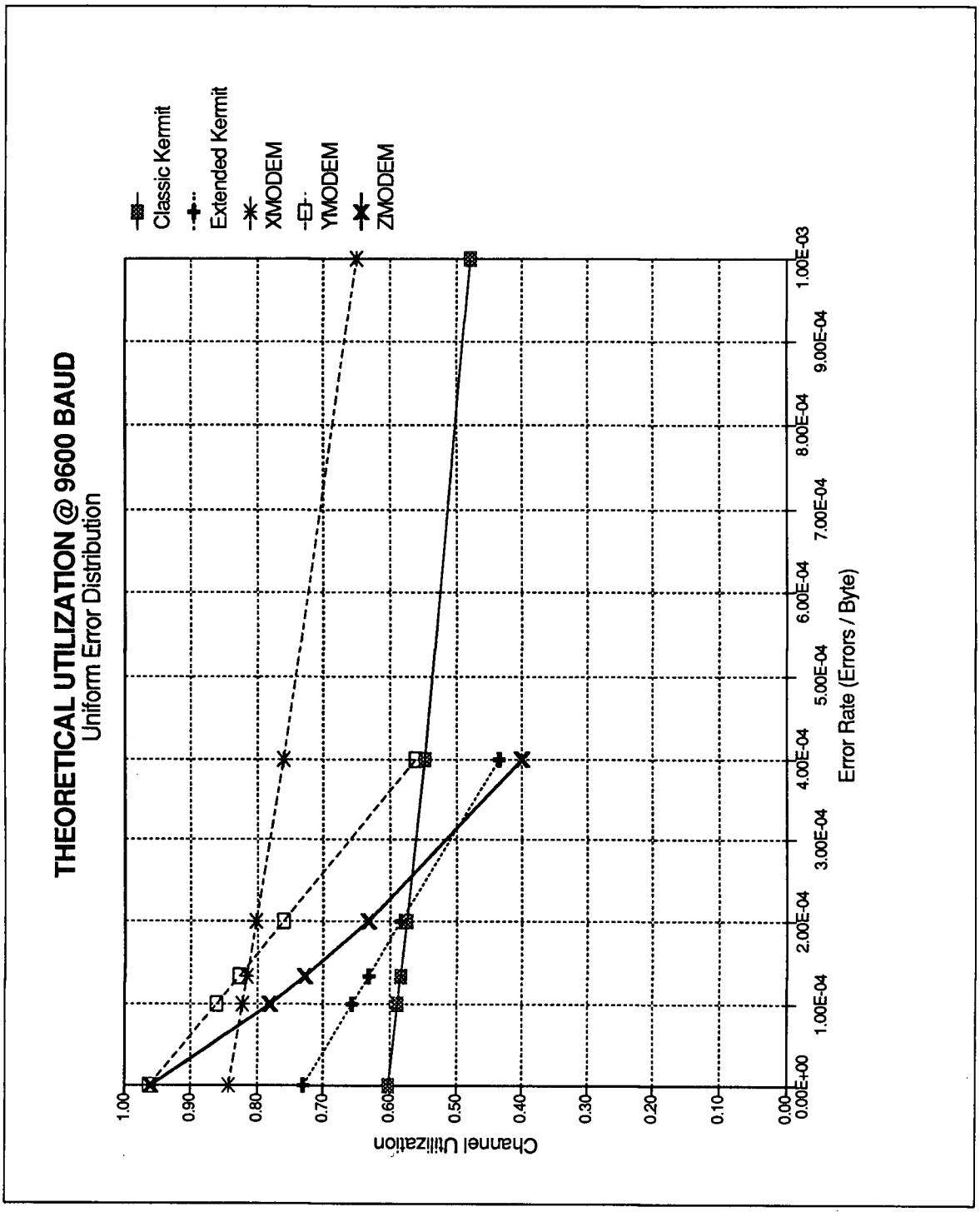
PROTOCOL	DATA PACKET SIZE	REPLY PACKET SIZE
Classic Kermit	97	8
Extended Kermit	1033	8
XMODEM-CRC	133	1
YMODEM	1029	1
ZMODEM	1027	21



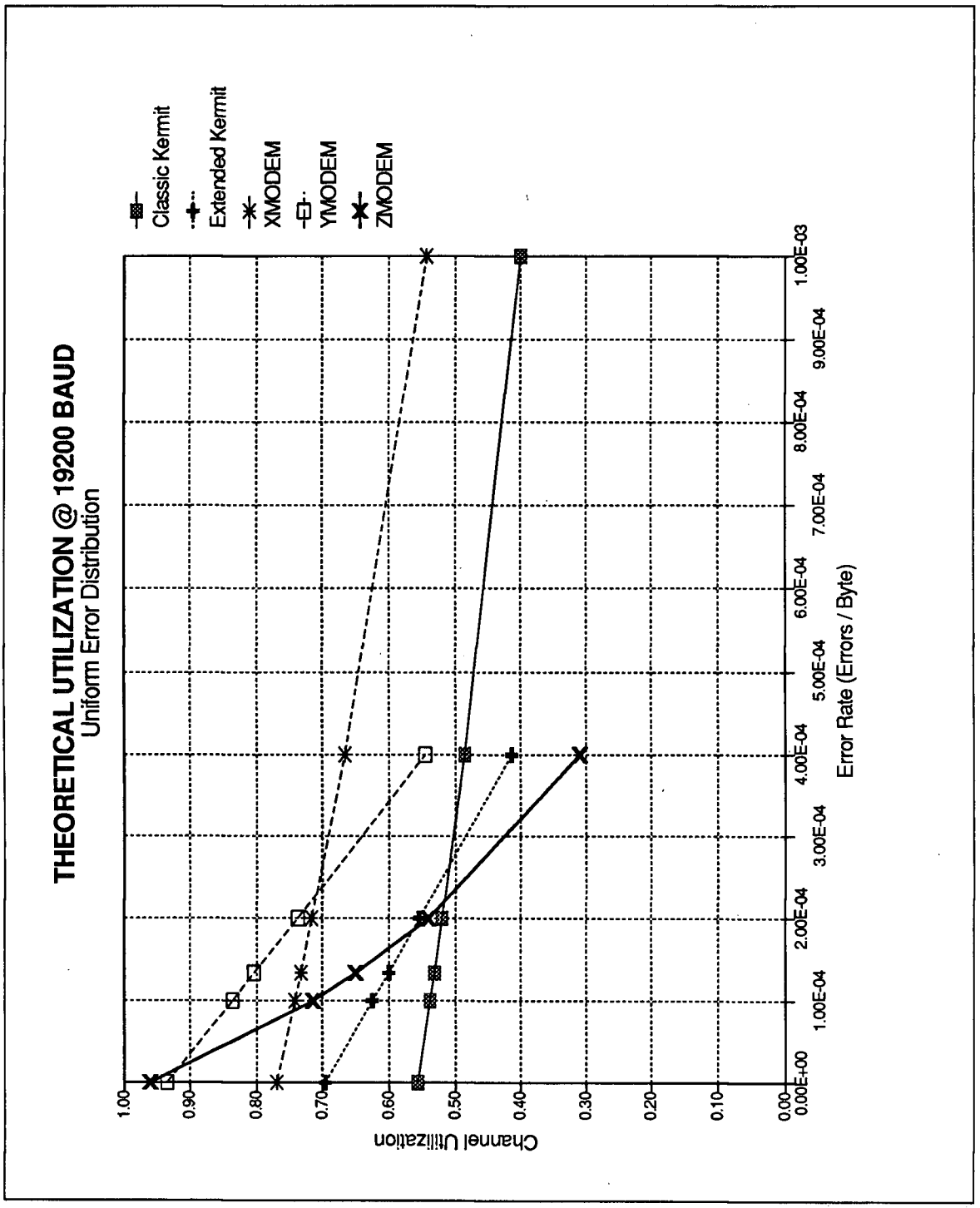
Graph 1. Theoretical Utilization At 2400 Baud And Uniform Error Distribution



Graph 2. Theoretical Utilization At 4800 Baud And Uniform Error Distribution



Graph 3. Theoretical Utilization At 9600 Baud And Uniform Error Distribution

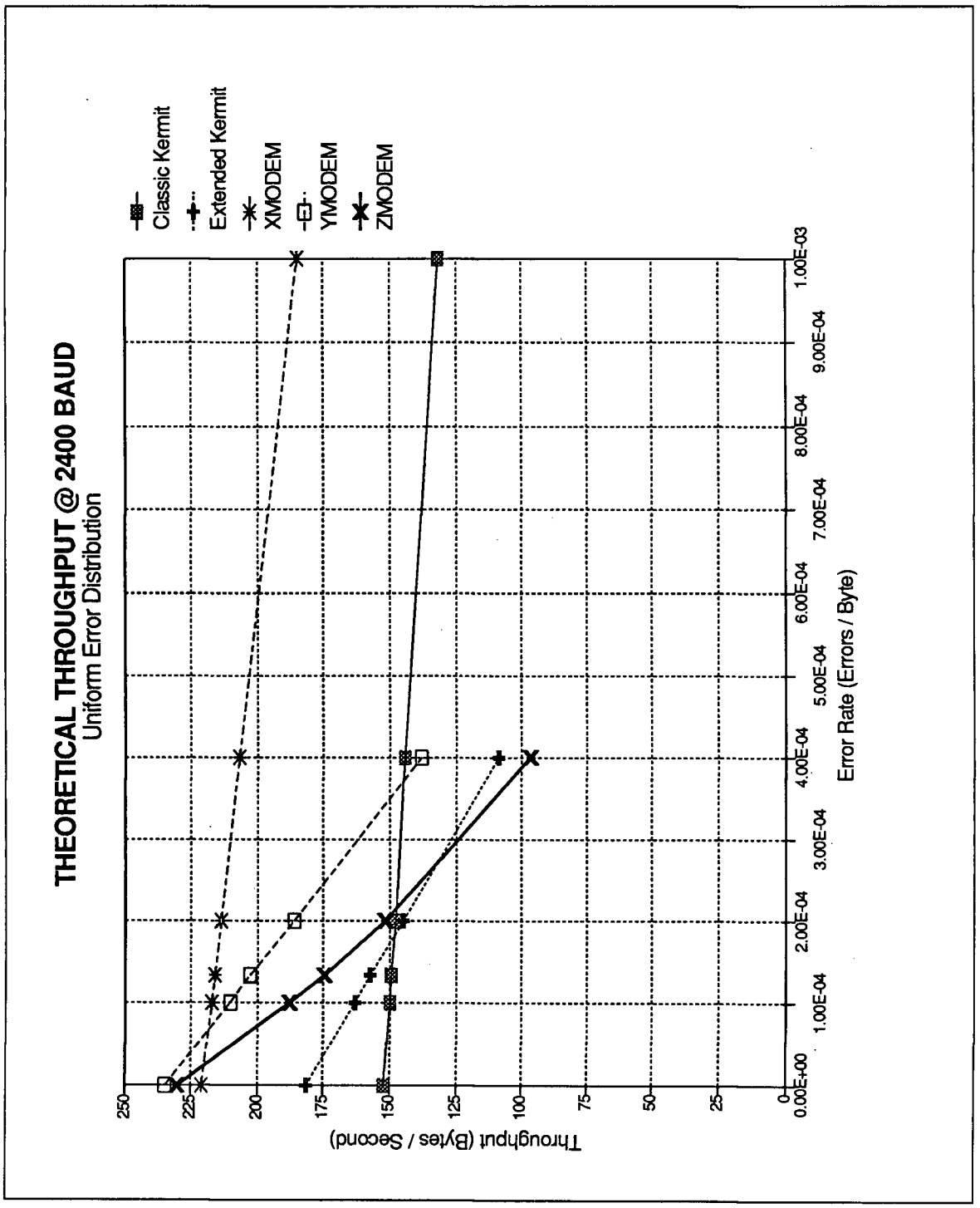


Graph 4. Theoretical Utilization At 19200 Baud And Uniform Error Distribution

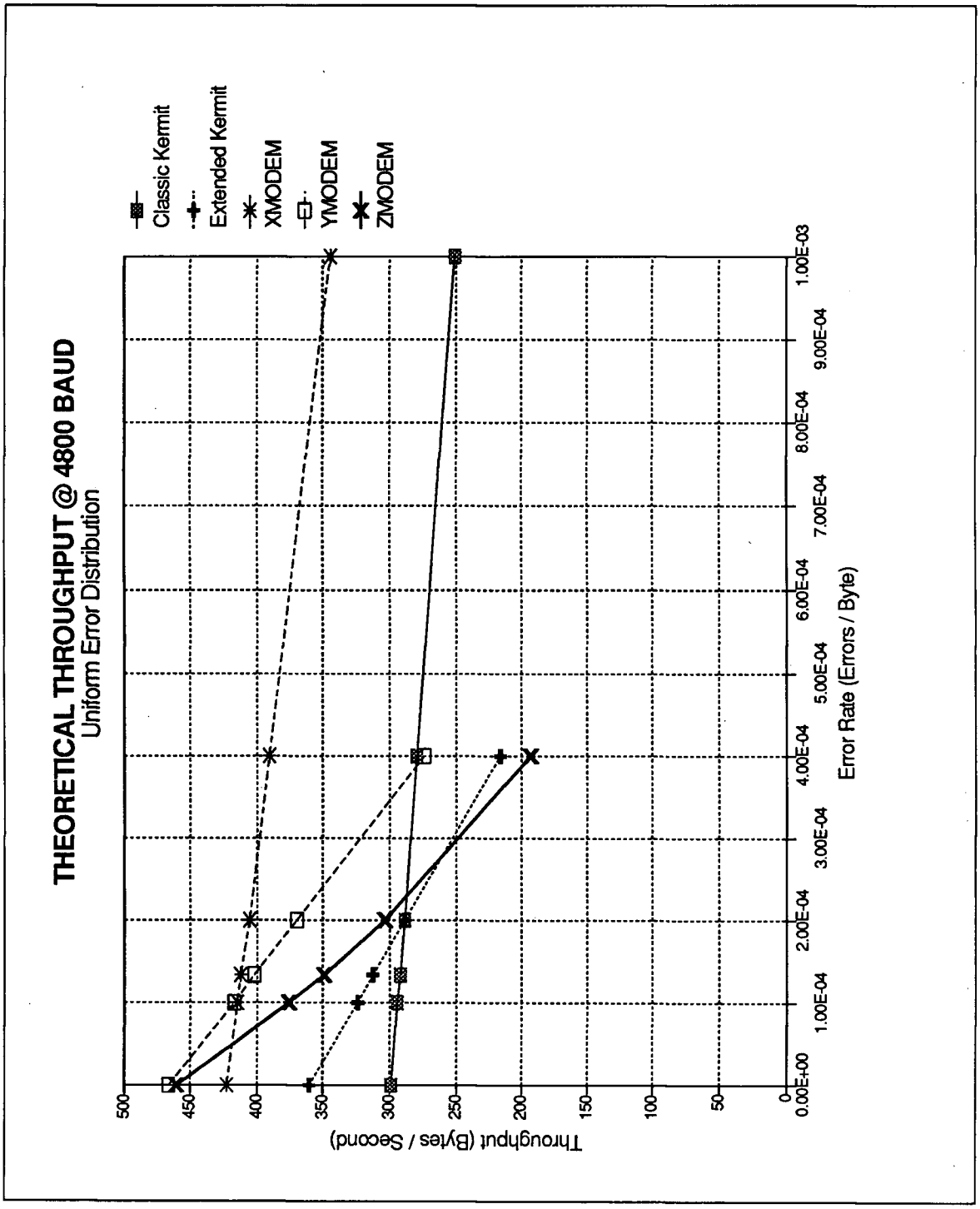
APPENDIX 2. Theoretical Throughput Graphs, Uniform Error Distribution

When examining the graphs in this appendix, please keep the following in mind:

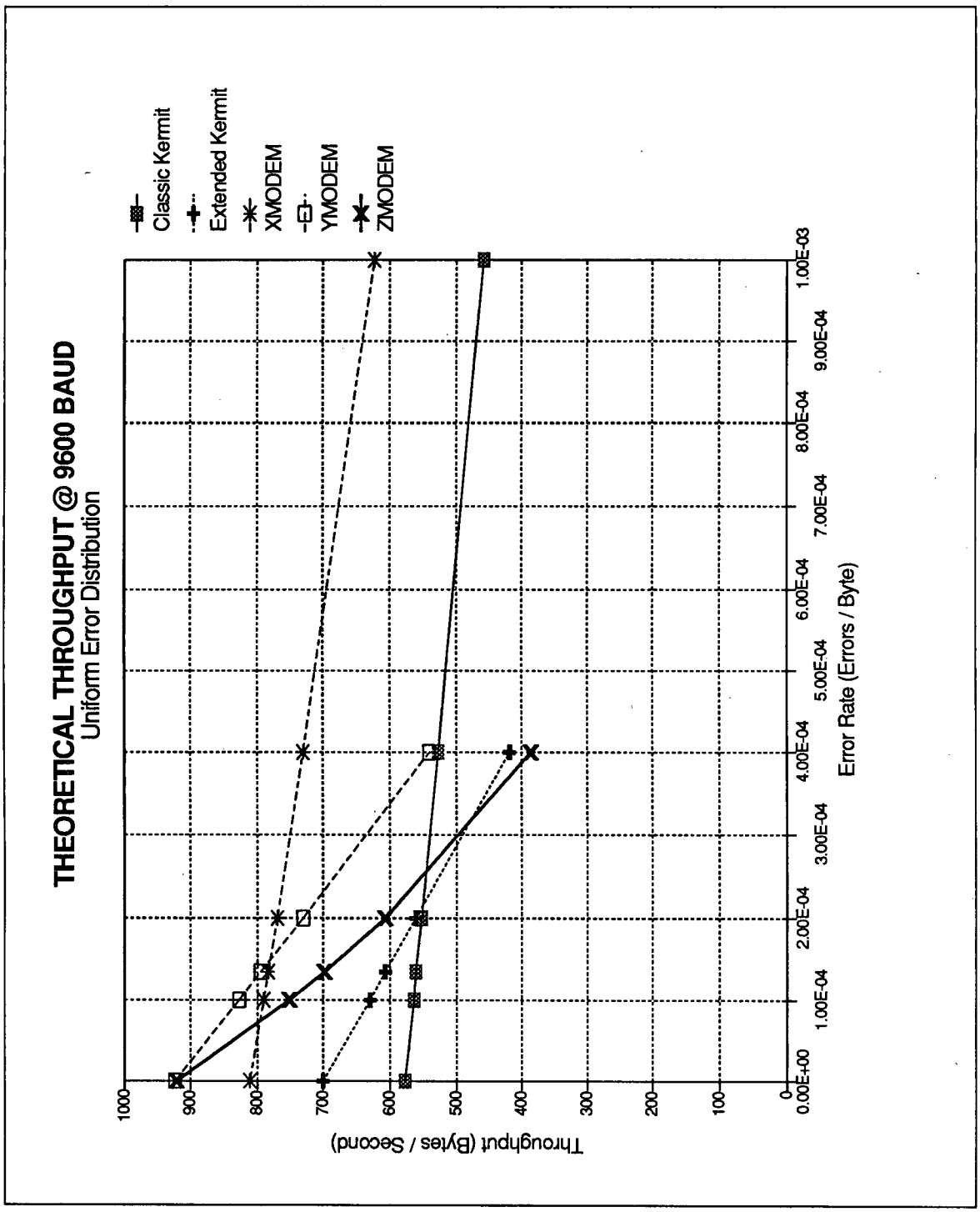
PROTOCOL	DATA PACKET SIZE	REPLY PACKET SIZE
Classic Kermit	97	8
Extended Kermit	1033	8
XMODEM-CRC	133	1
YMODEM	1029	1
ZMODEM	1027	21



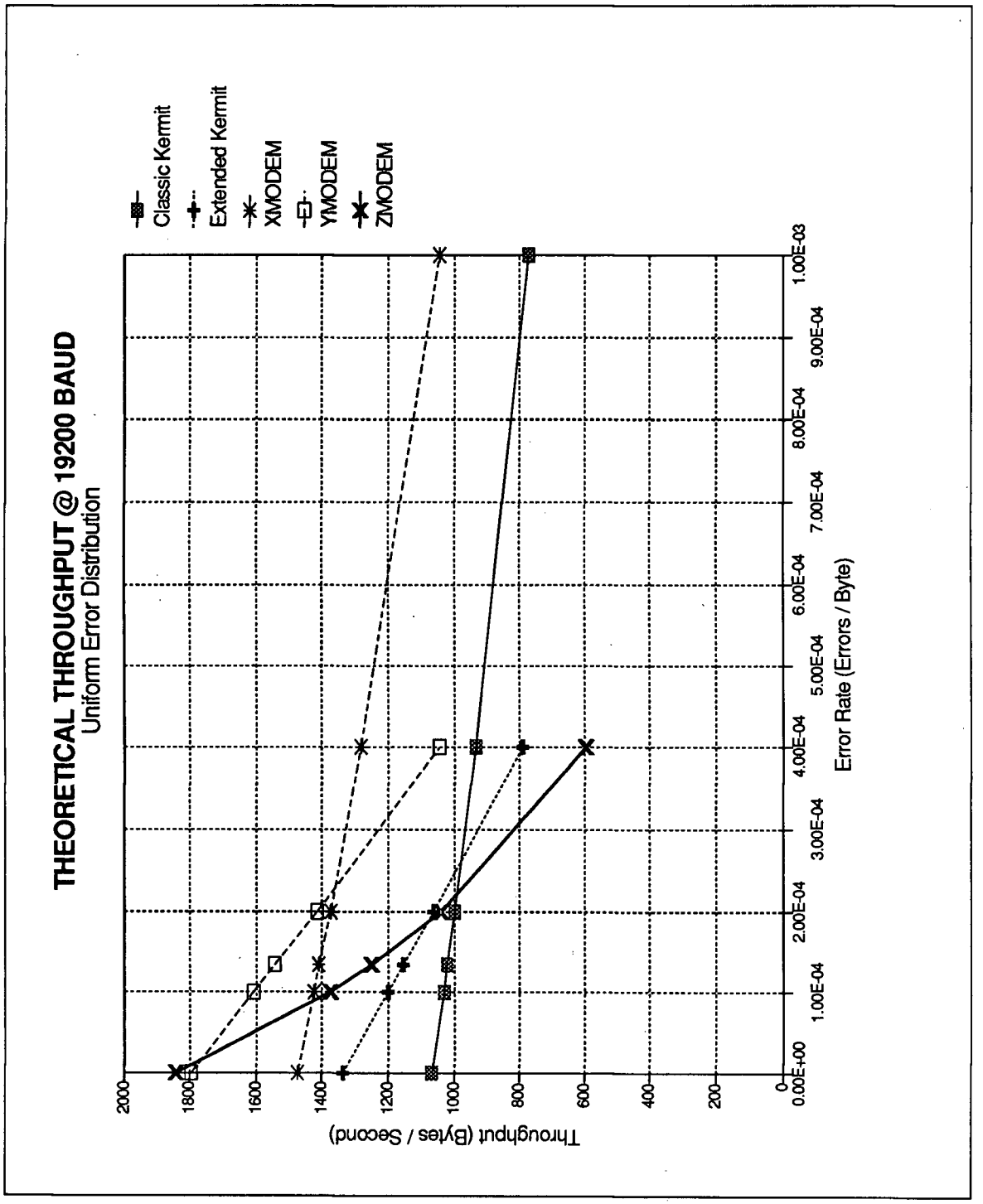
Graph 5. Theoretical Throughput At 2400 Baud And Uniform Error Distribution



Graph 6. Theoretical Throughput At 4800 Baud And Uniform Error Distribution



Graph 7. Theoretical Throughput At 9600 Baud And Uniform Error Distribution

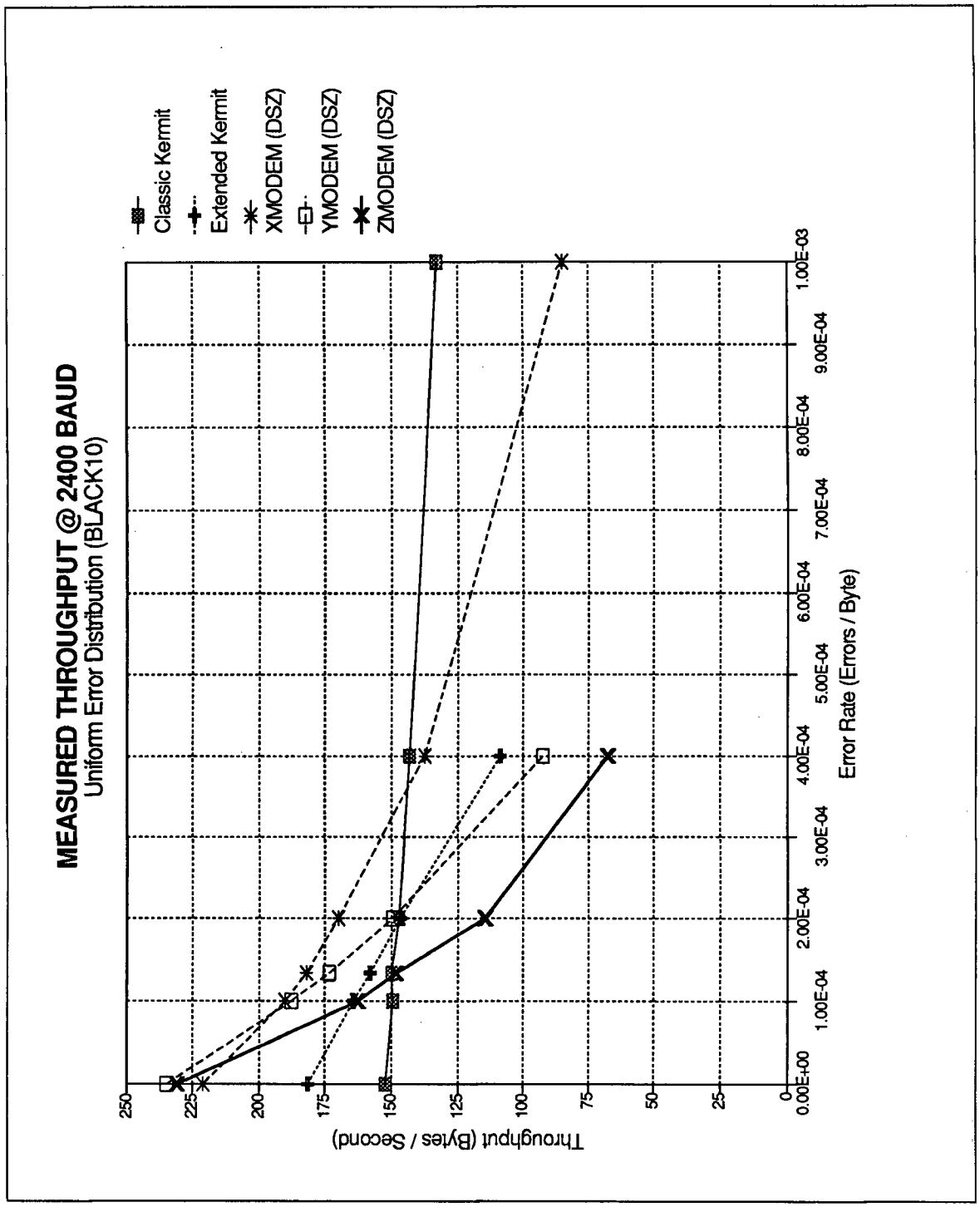


Graph 8. Theoretical Throughput At 19200 Baud And Uniform Error Distribution

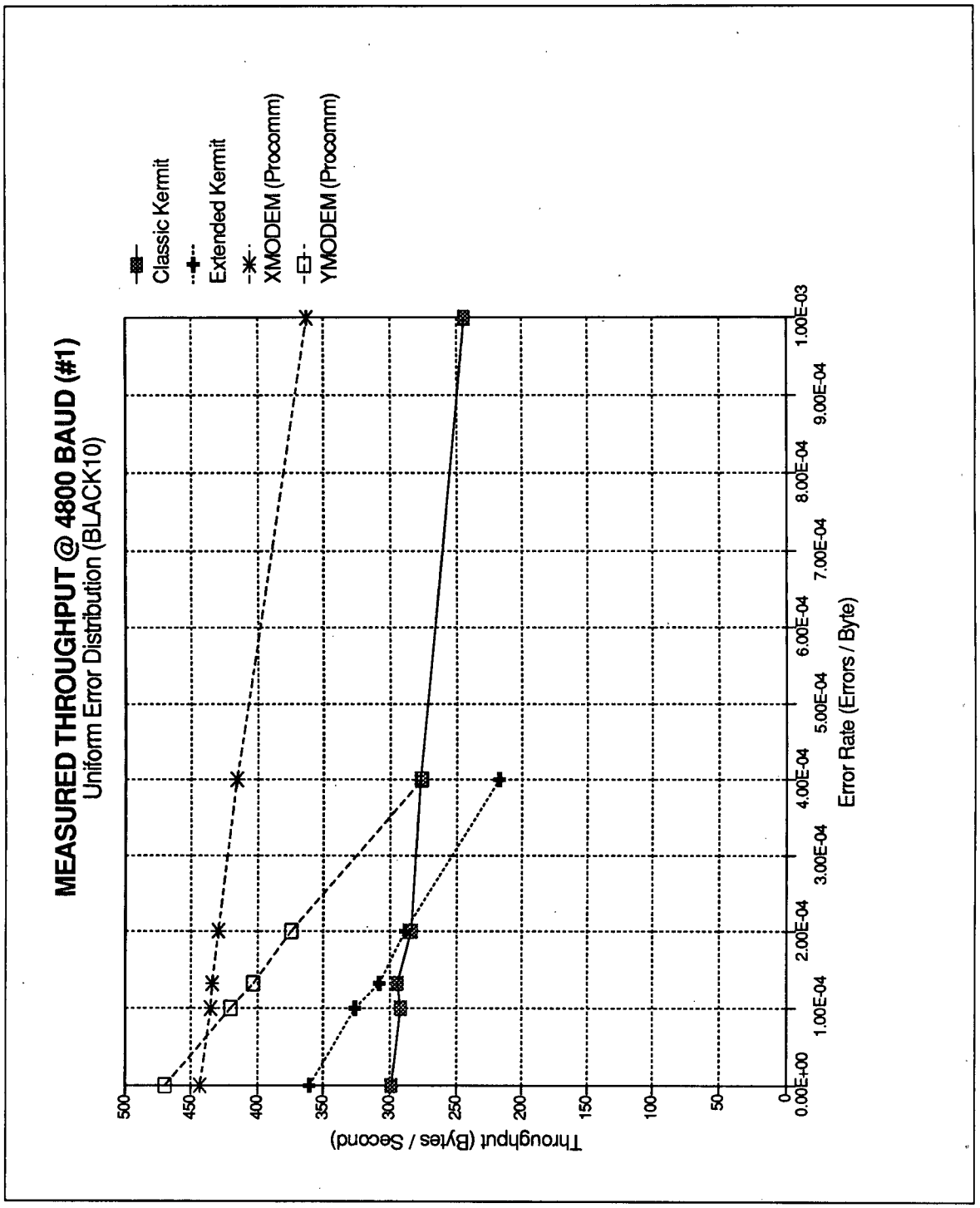
APPENDIX 3. Measured Throughput Graphs, Uniform Error Distribution

When examining the graphs in this appendix, please keep the following in mind:

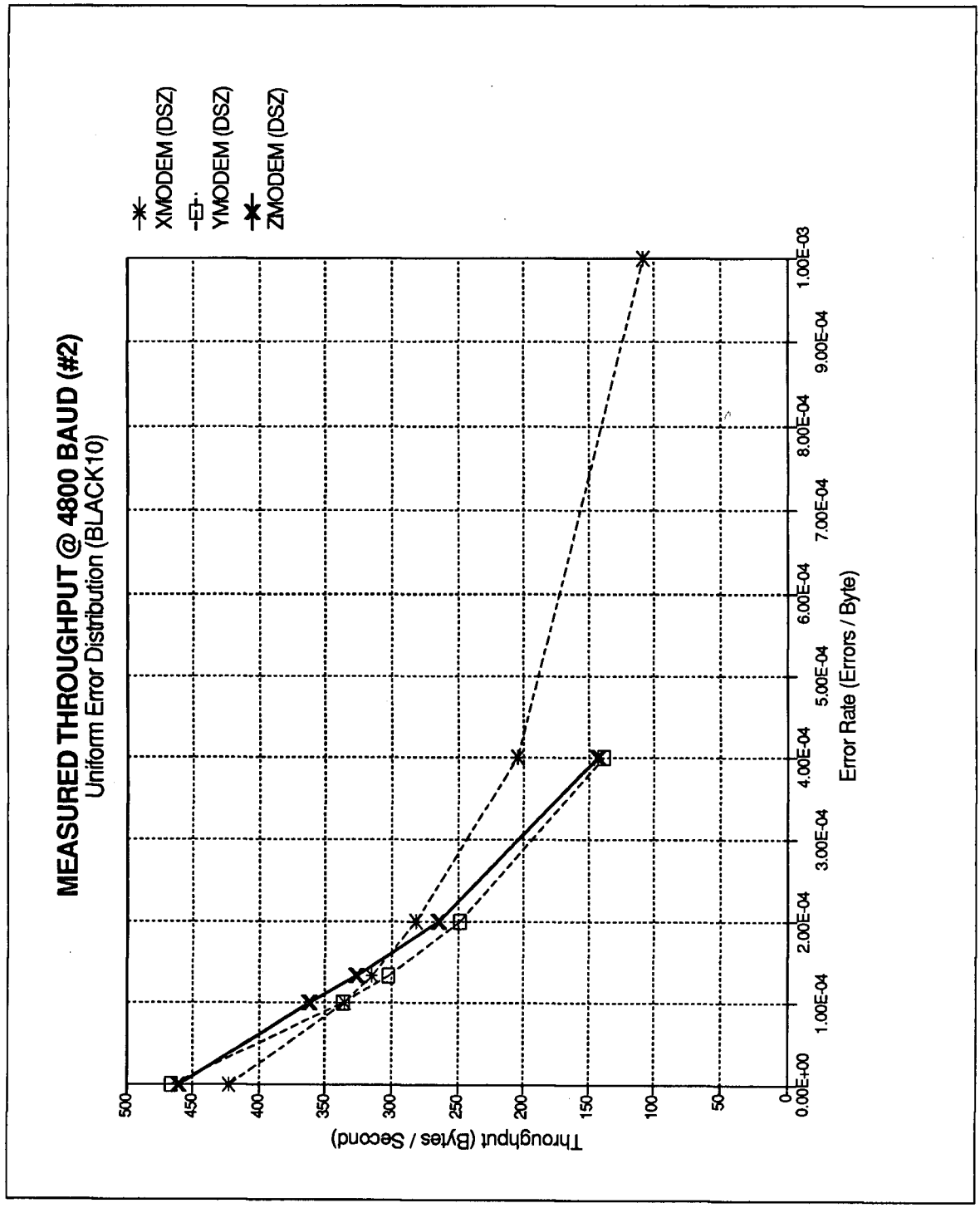
PROTOCOL	DATA PACKET SIZE	REPLY PACKET SIZE
Classic Kermit	97	8
Extended Kermit	1033	8
XMODEM-CRC	133	1
YMODEM	1029	1
ZMODEM	1027	21



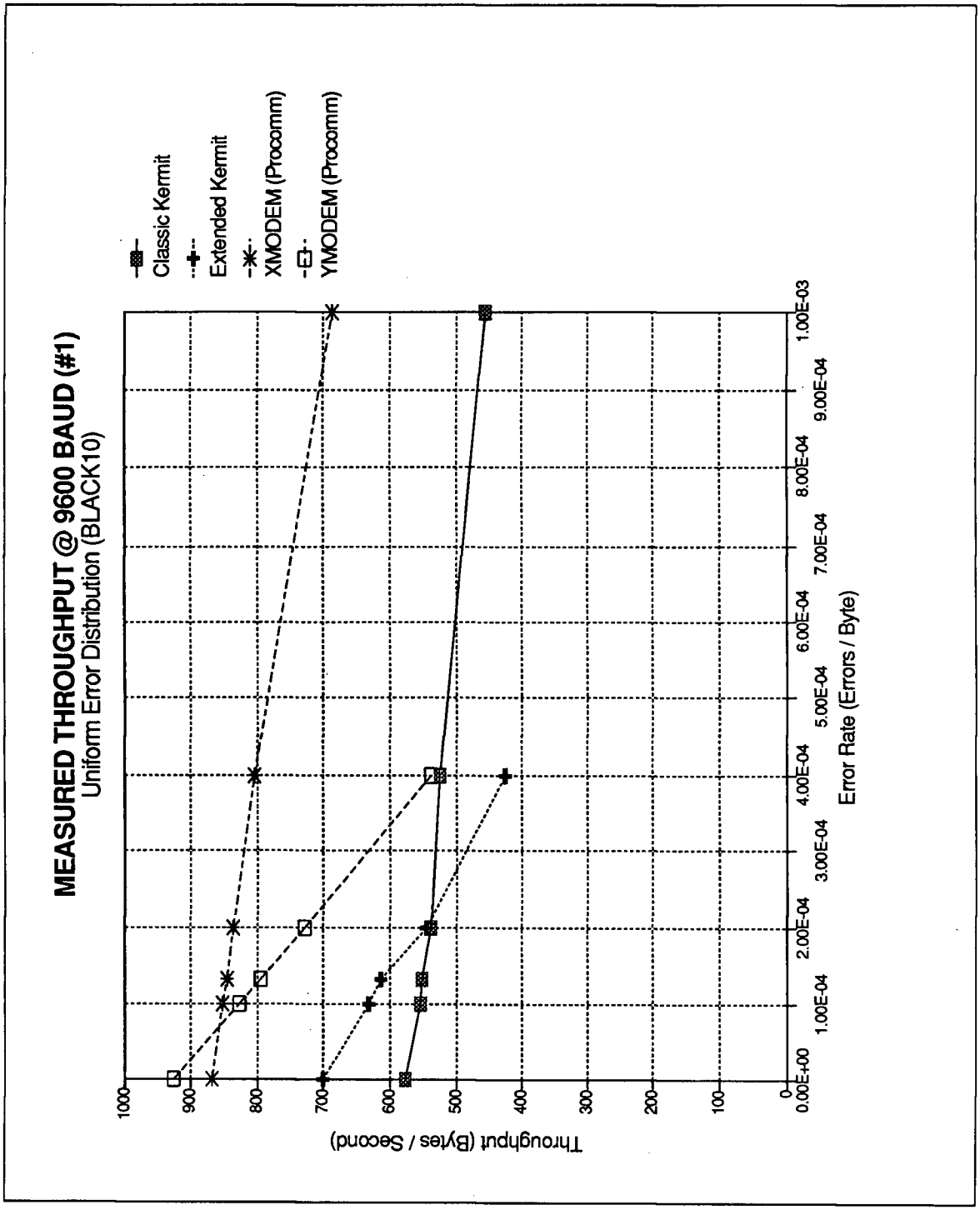
Graph 9. Measured Throughput At 2400 Baud And Uniform Error Distribution



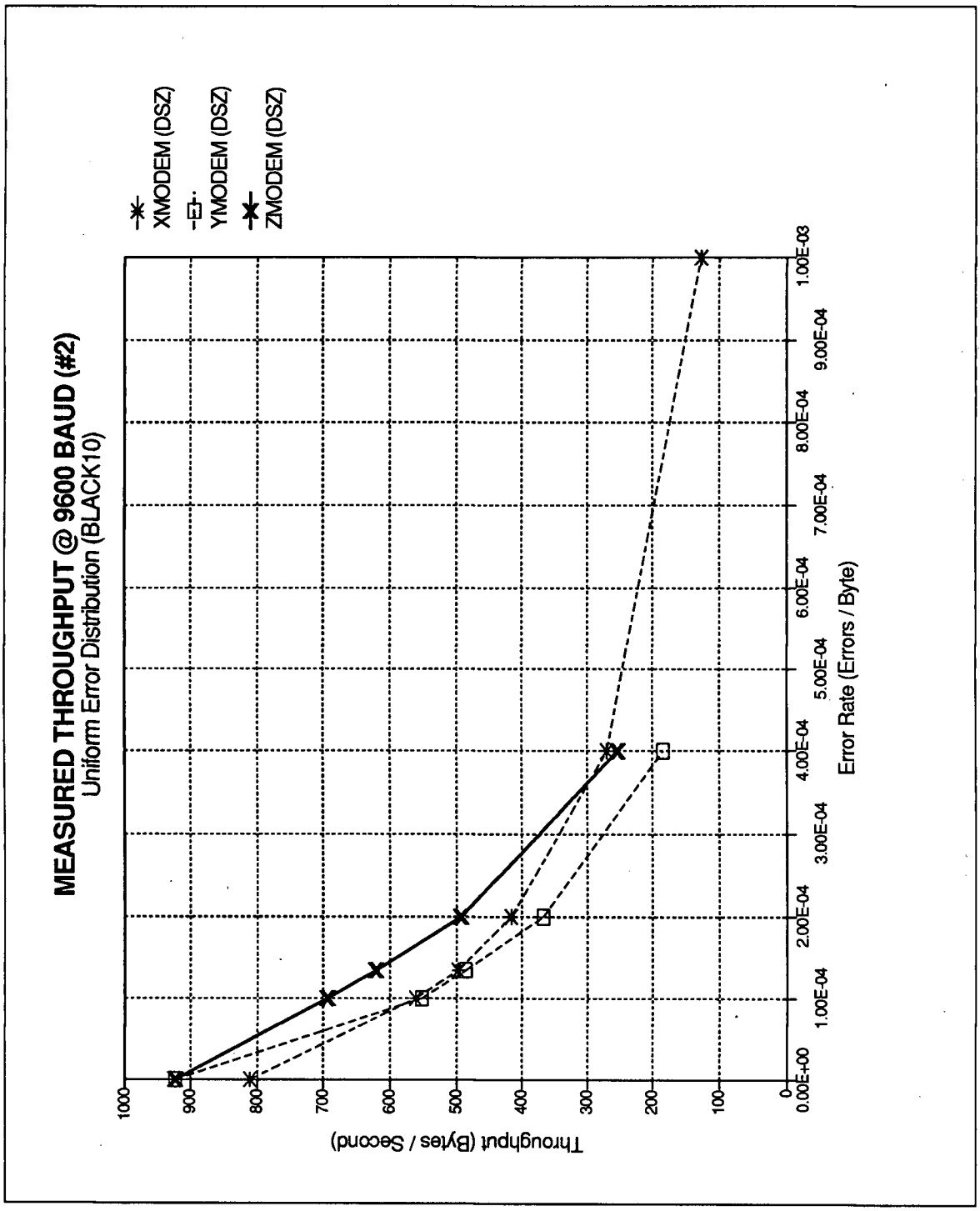
Graph 10. Measured Throughput At 4800 Baud (#1) Uniform Error Distribution



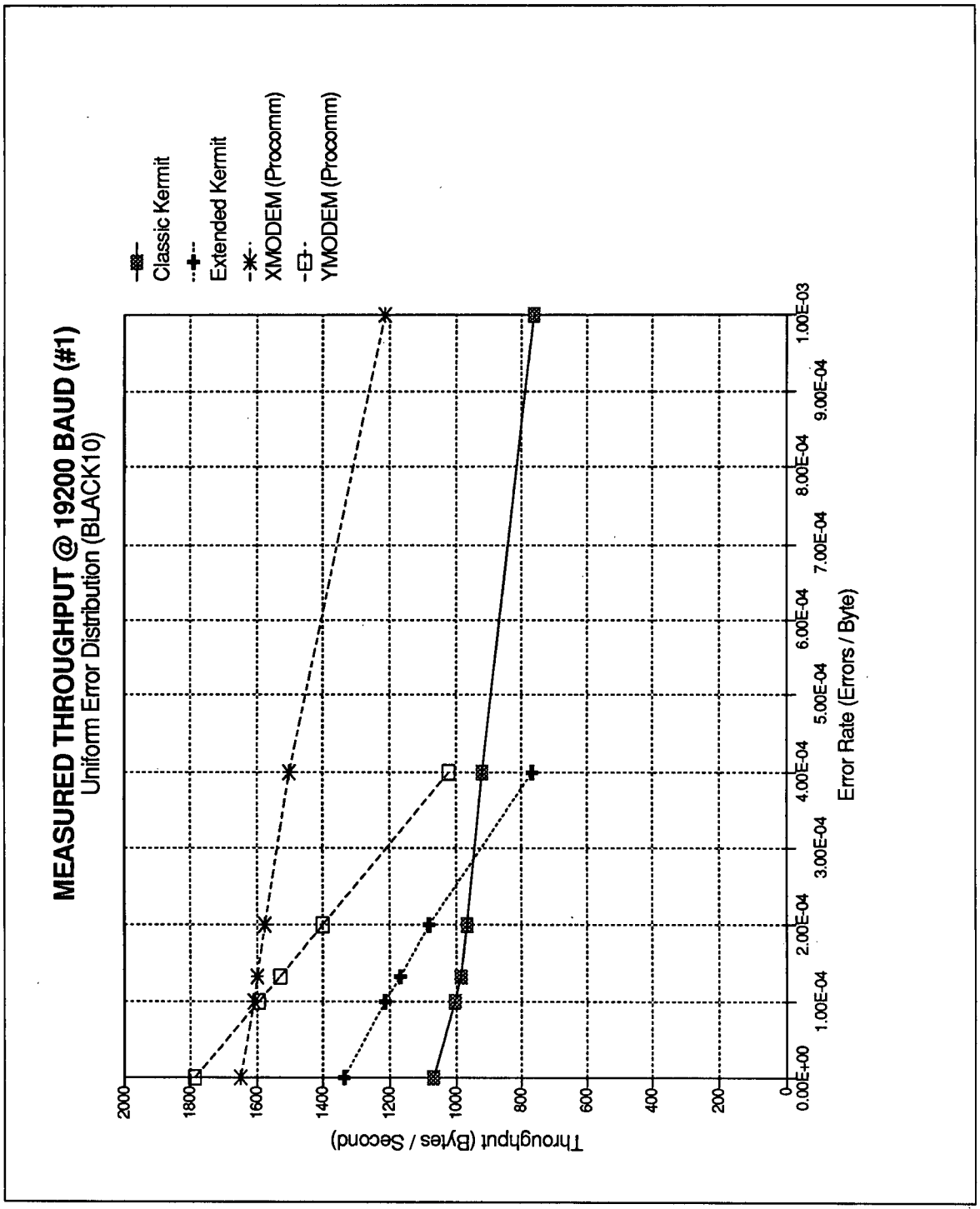
Graph 11. Measured Throughput At 4800 Baud (#2) Uniform Error Distribution



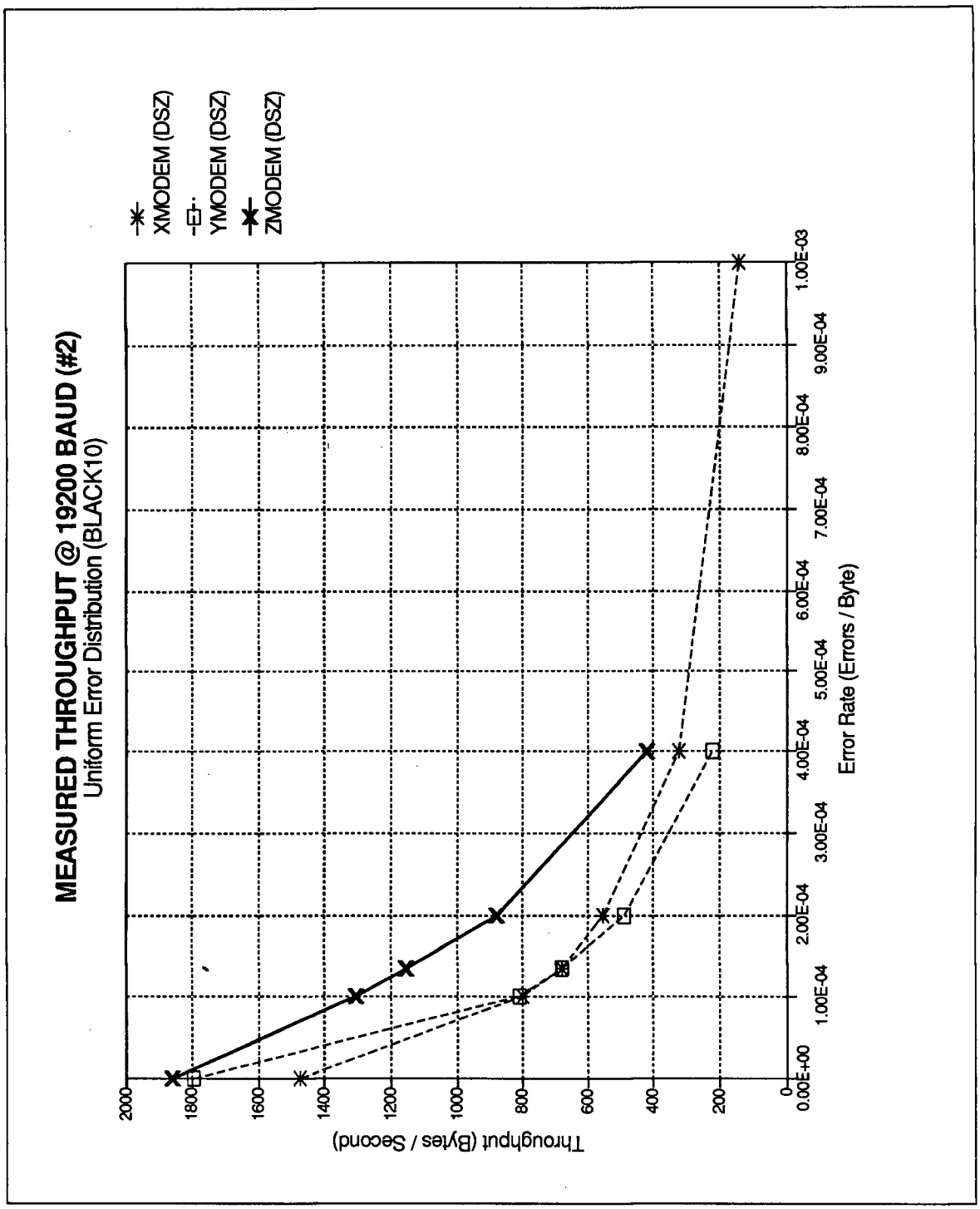
Graph 12. Measured Throughput At 9600 Baud (#1) Uniform Error Distribution



Graph 13. Measured Throughput At 9600 Baud (#2) Uniform Error Distribution



Graph 14. Measured Throughput At 19200 Baud (#1) Uniform Error Distribution

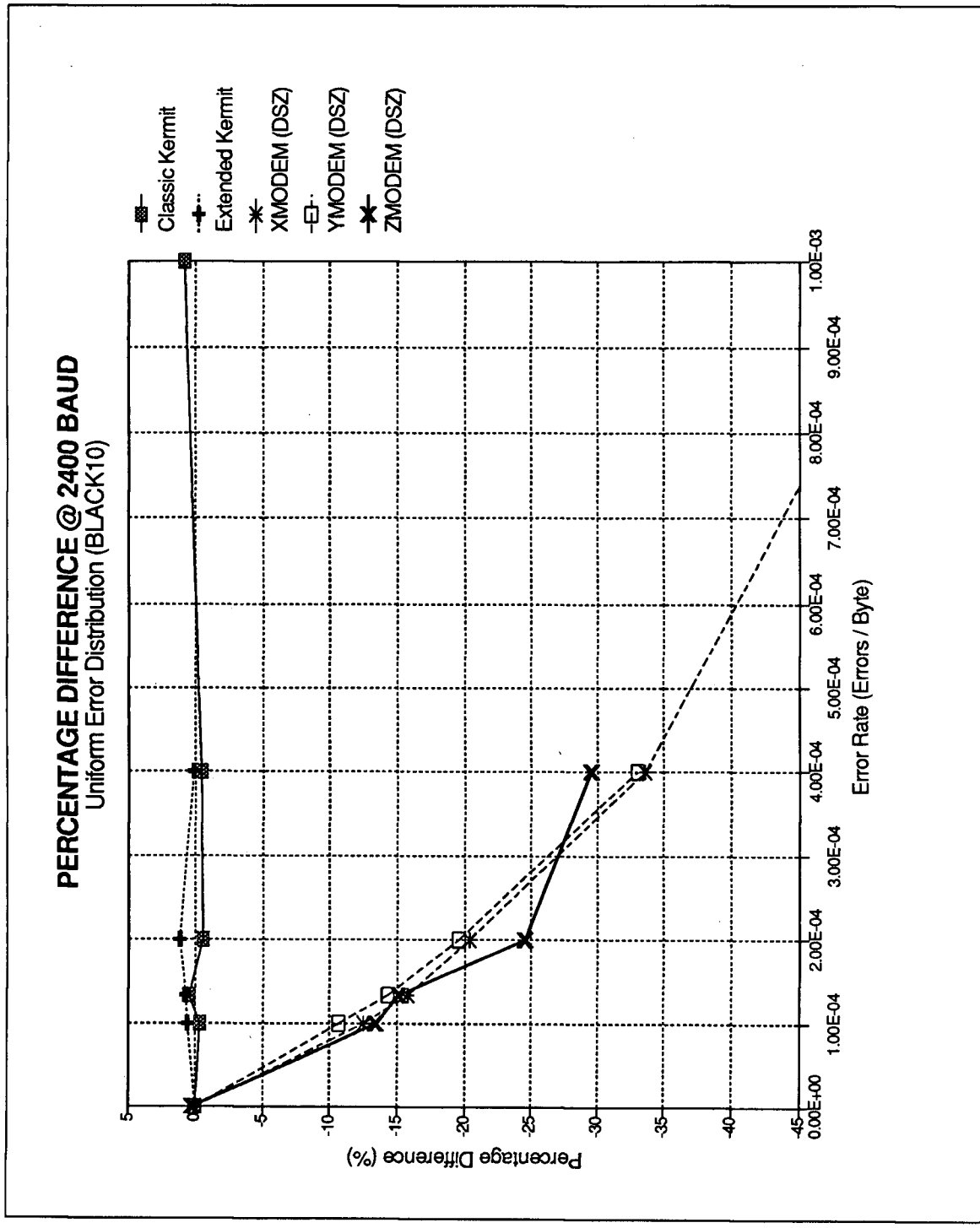


Graph 15. Measured Throughput At 19200 Baud (#2) Uniform Error Distribution

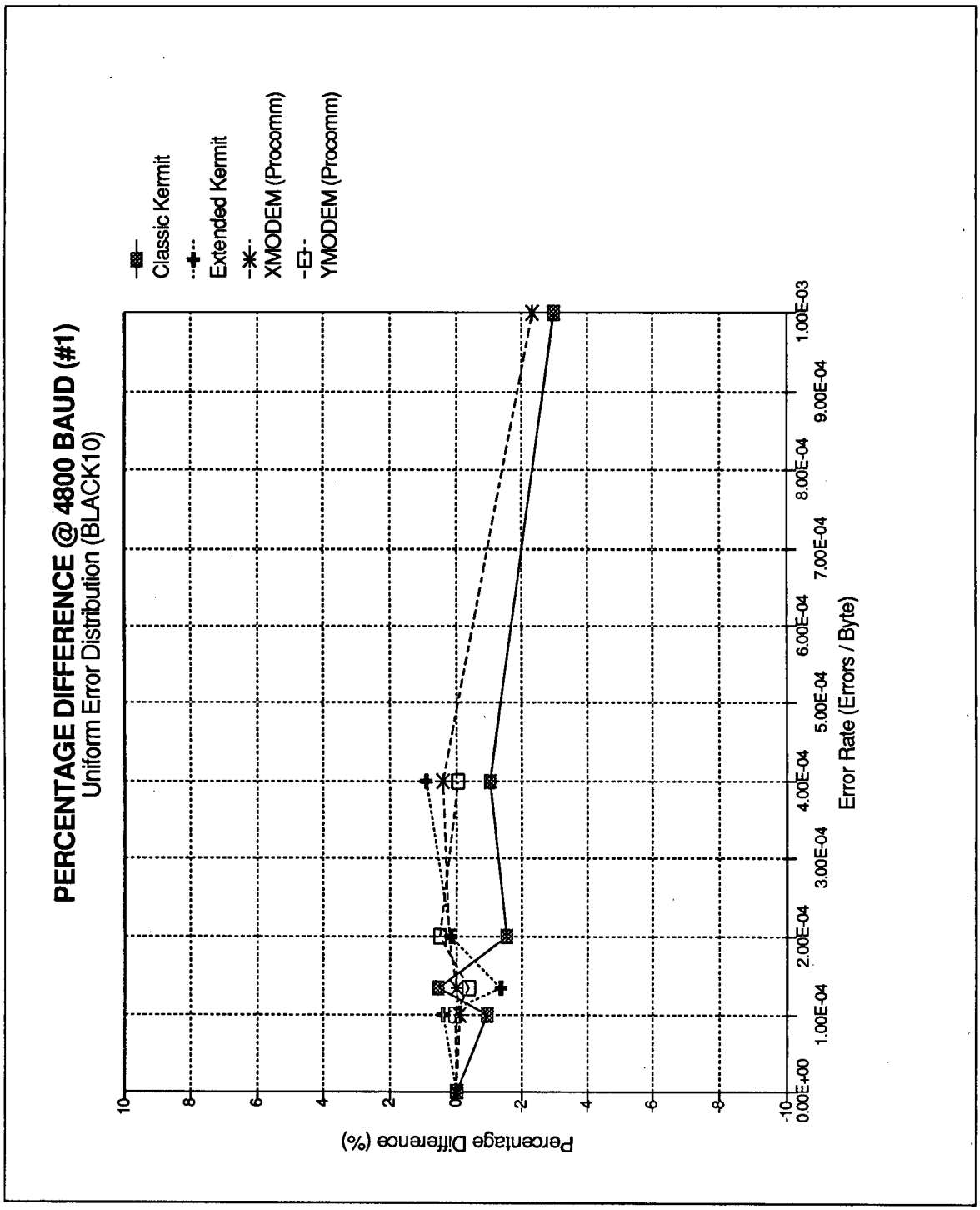
APPENDIX 4. Percentage Difference Graphs, Uniform Error Distribution

When examining the graphs in this appendix, please keep the following in mind:

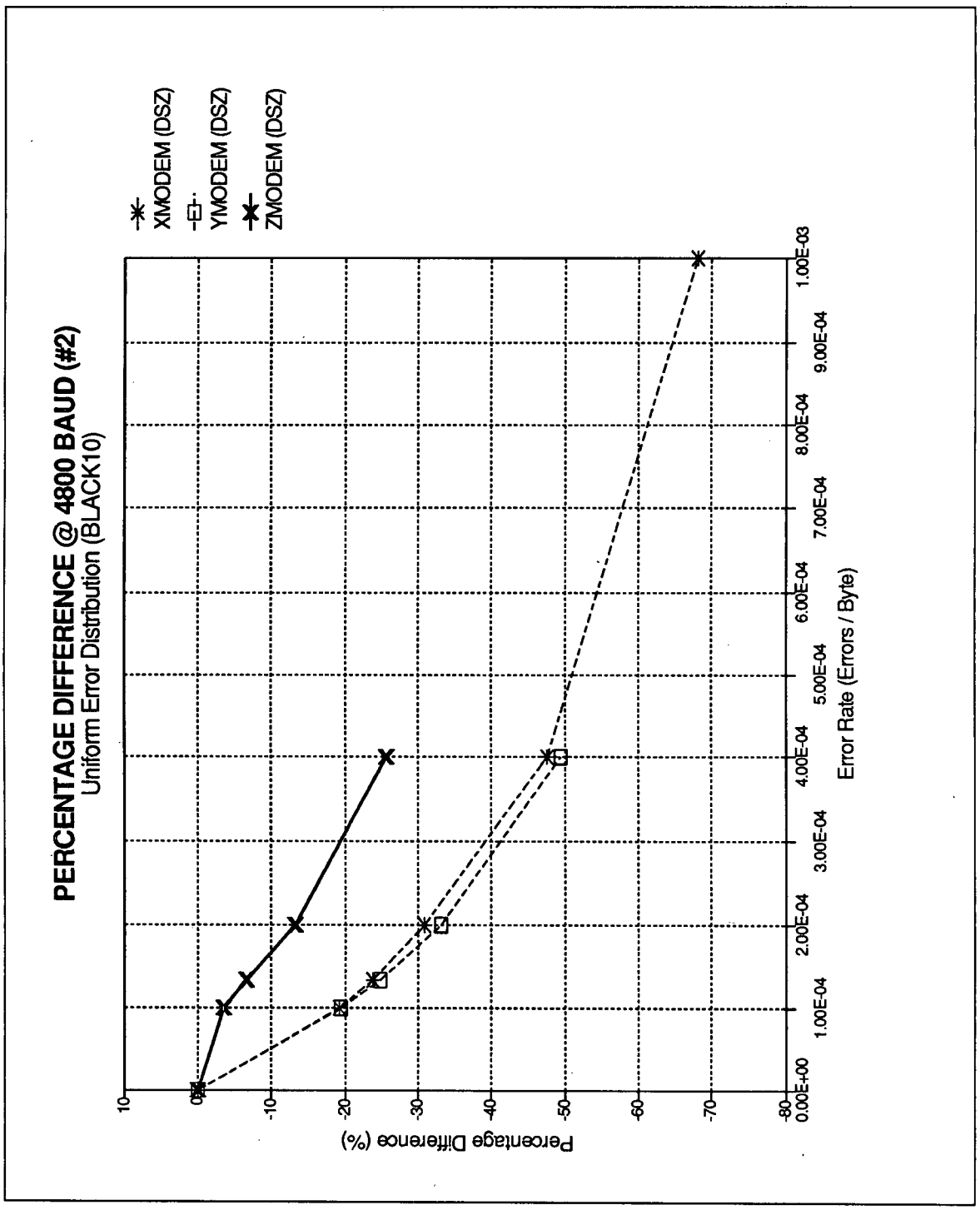
PROTOCOL	DATA PACKET SIZE	REPLY PACKET SIZE
Classic Kermit	97	8
Extended Kermit	1033	8
XMODEM-CRC	133	1
YMODEM	1029	1
ZMODEM	1027	21



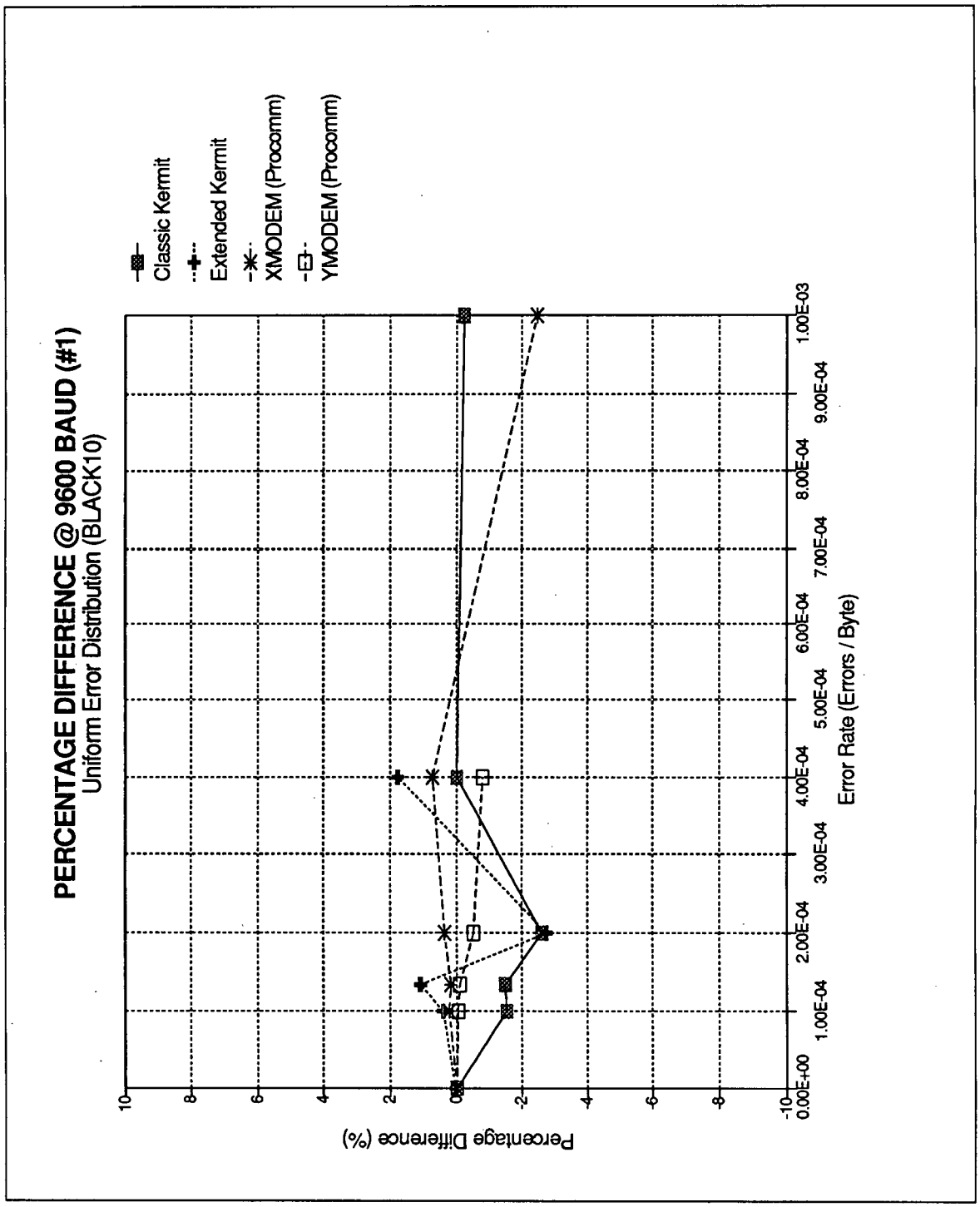
Graph 16. Percentage Difference At 2400 Baud Uniform Error Distribution



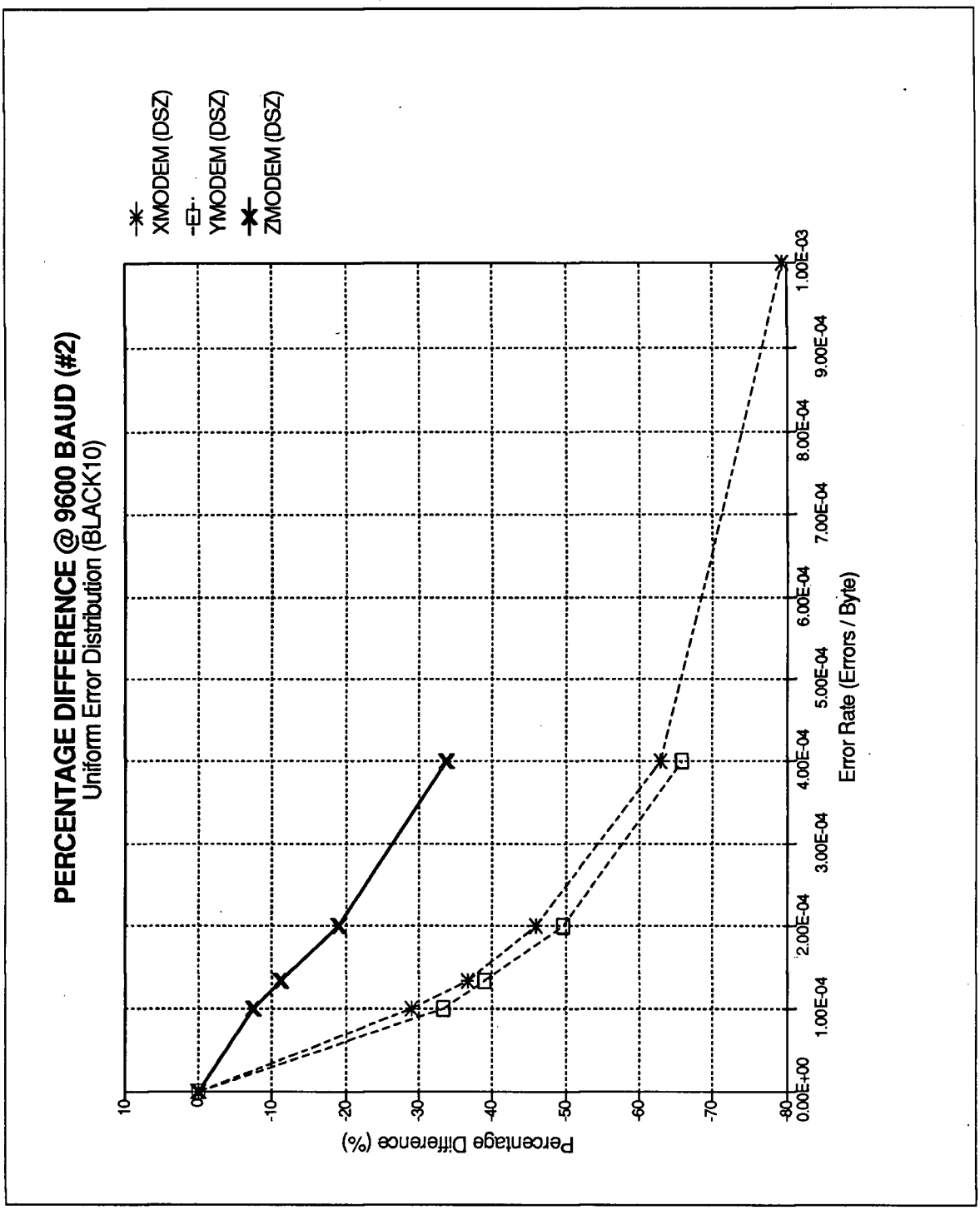
Graph 17. Percentage Difference At 4800 Baud (#1) Uniform Error Distribution



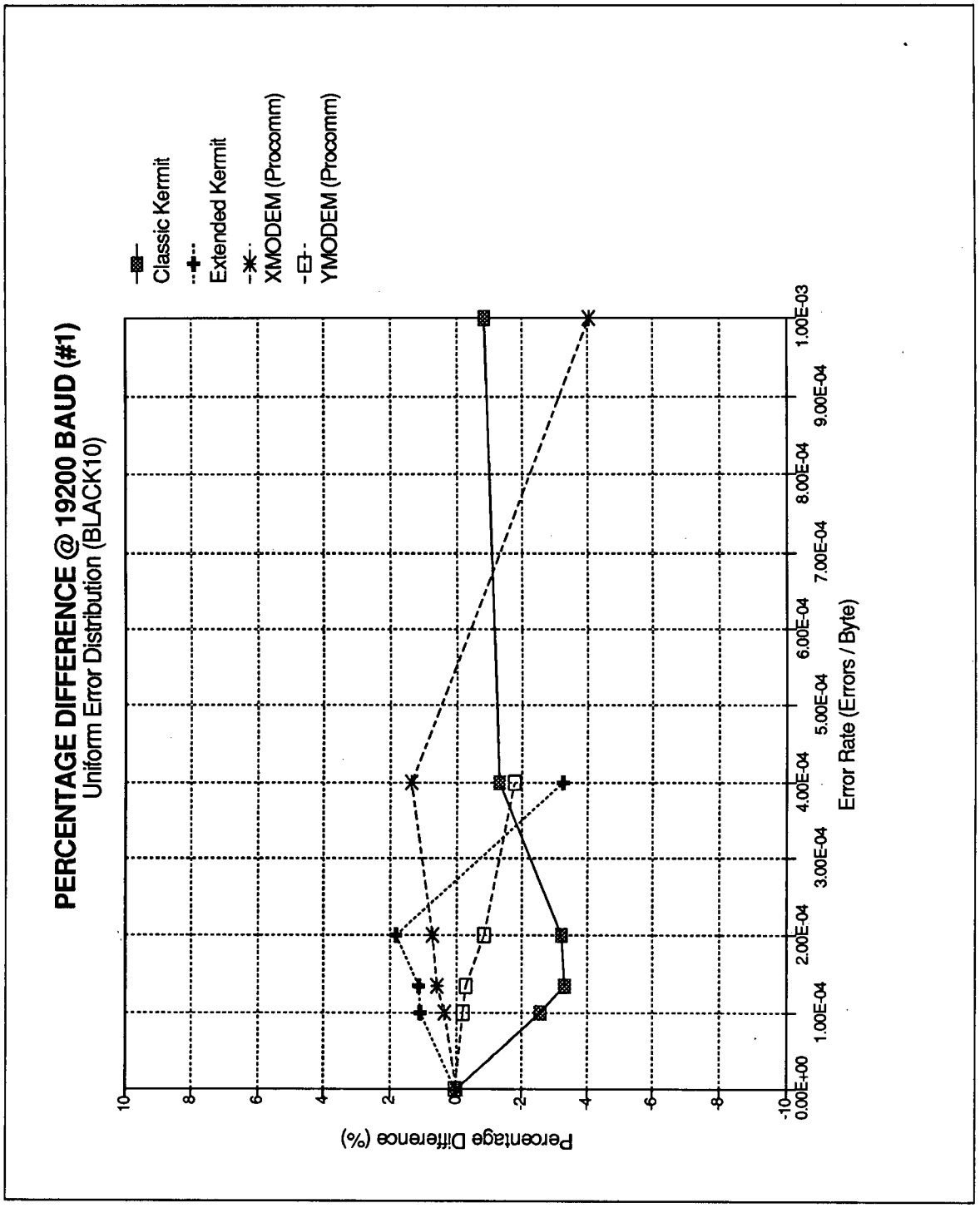
Graph 18. Percentage Difference At 4800 Baud (#2) Uniform Error Distribution



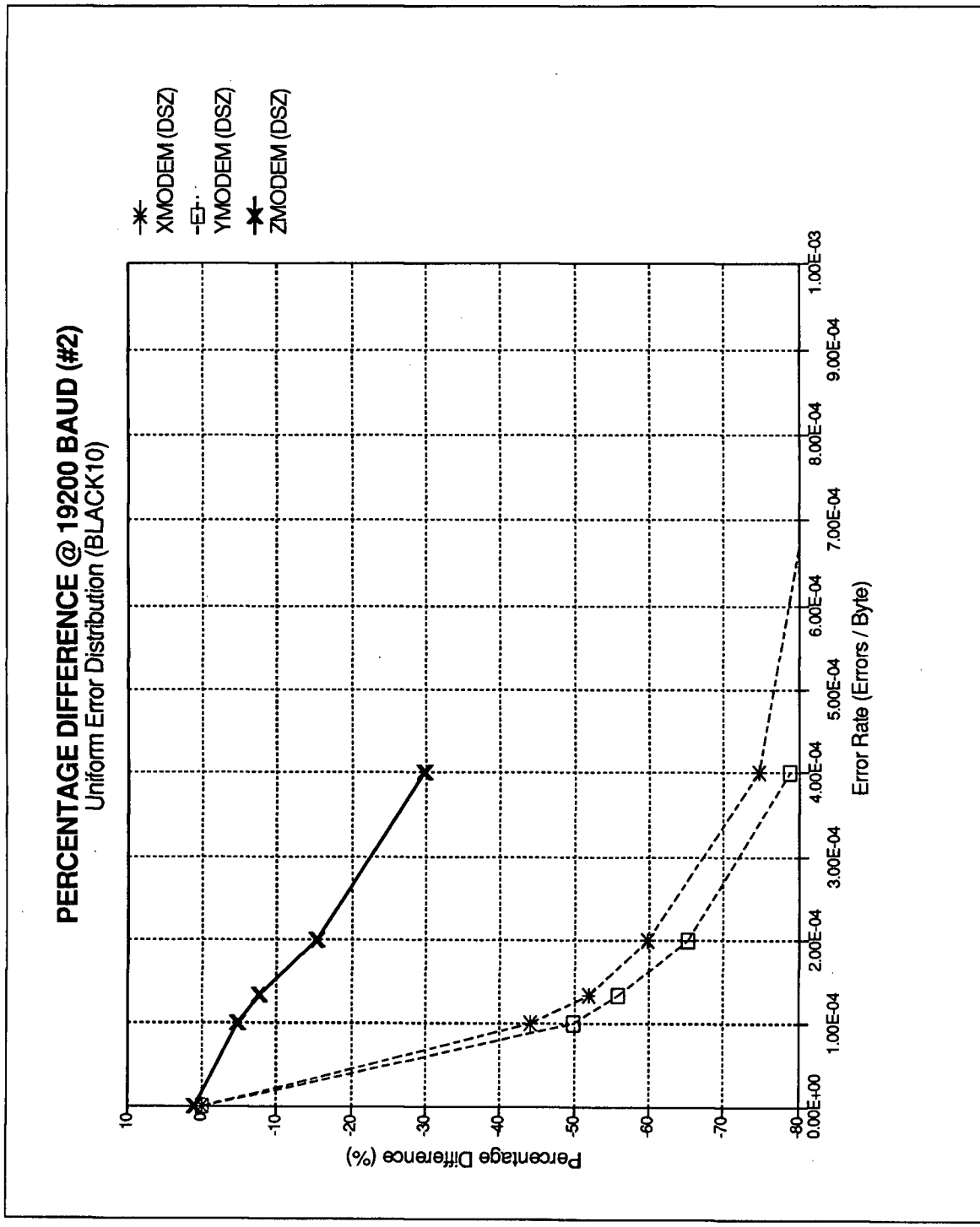
Graph 19. Percentage Difference At 9600 Baud (#1) Uniform Error Distribution



Graph 20. Percentage Difference At 9600 Baud (#2) Uniform Error Distribution



Graph 21. Percentage Difference At 19200 Baud (#1) Uniform Error Distribution

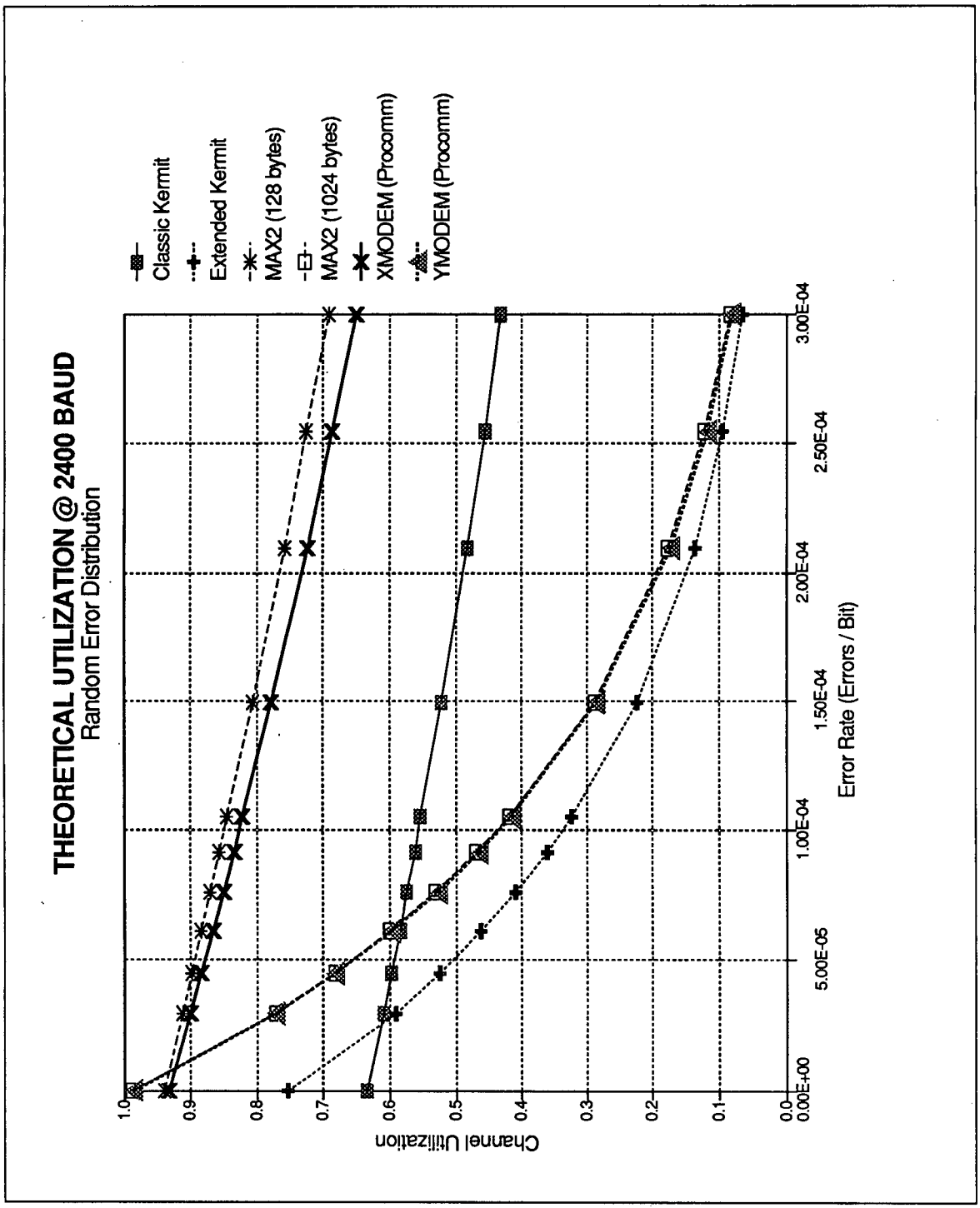


Graph 22. Percentage Difference At 19200 Baud (#2) Uniform Error Distribution

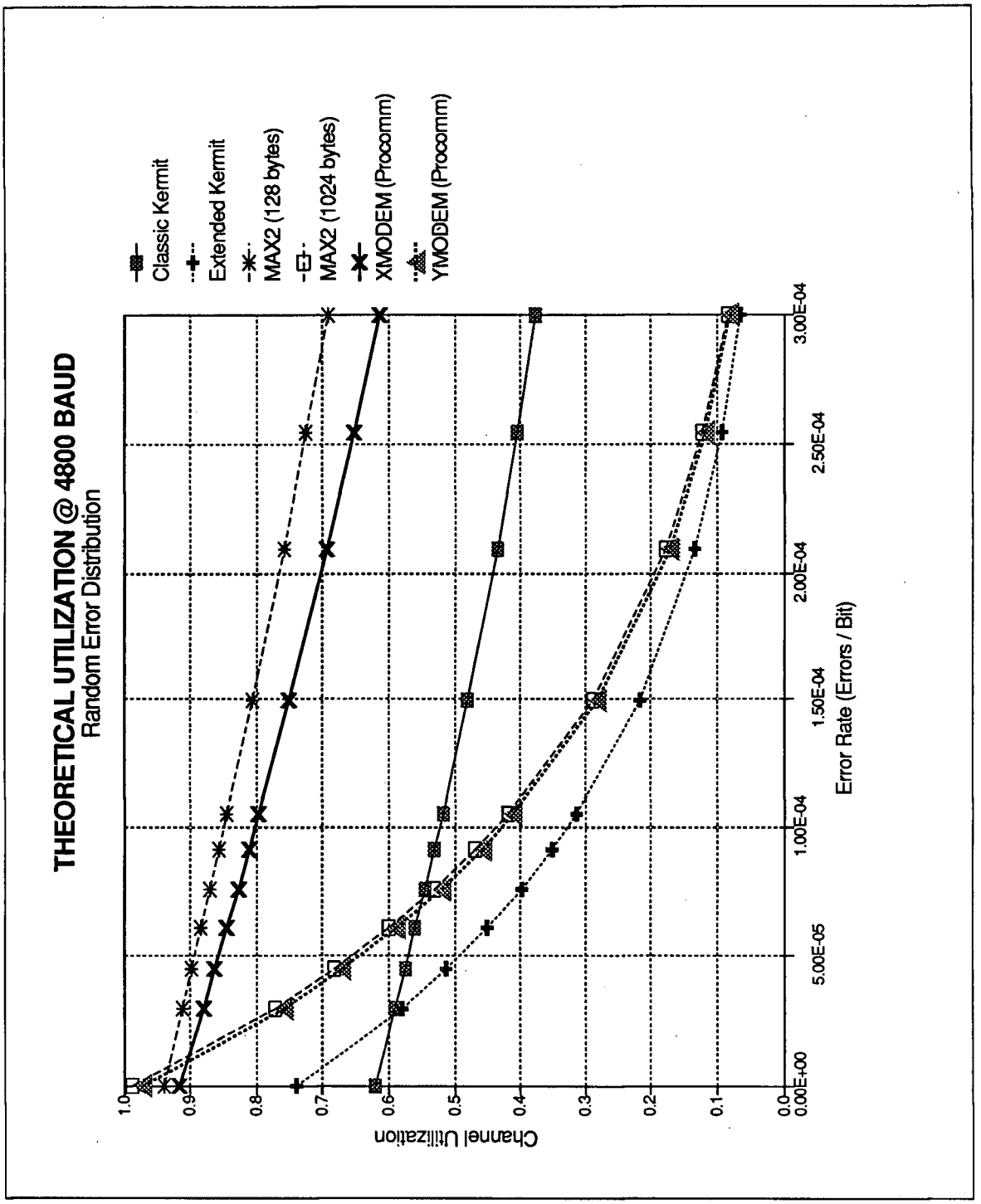
APPENDIX 5. Theoretical Utilization Graphs, Random Error Distribution

When examining the graphs in this appendix, please keep the following in mind:

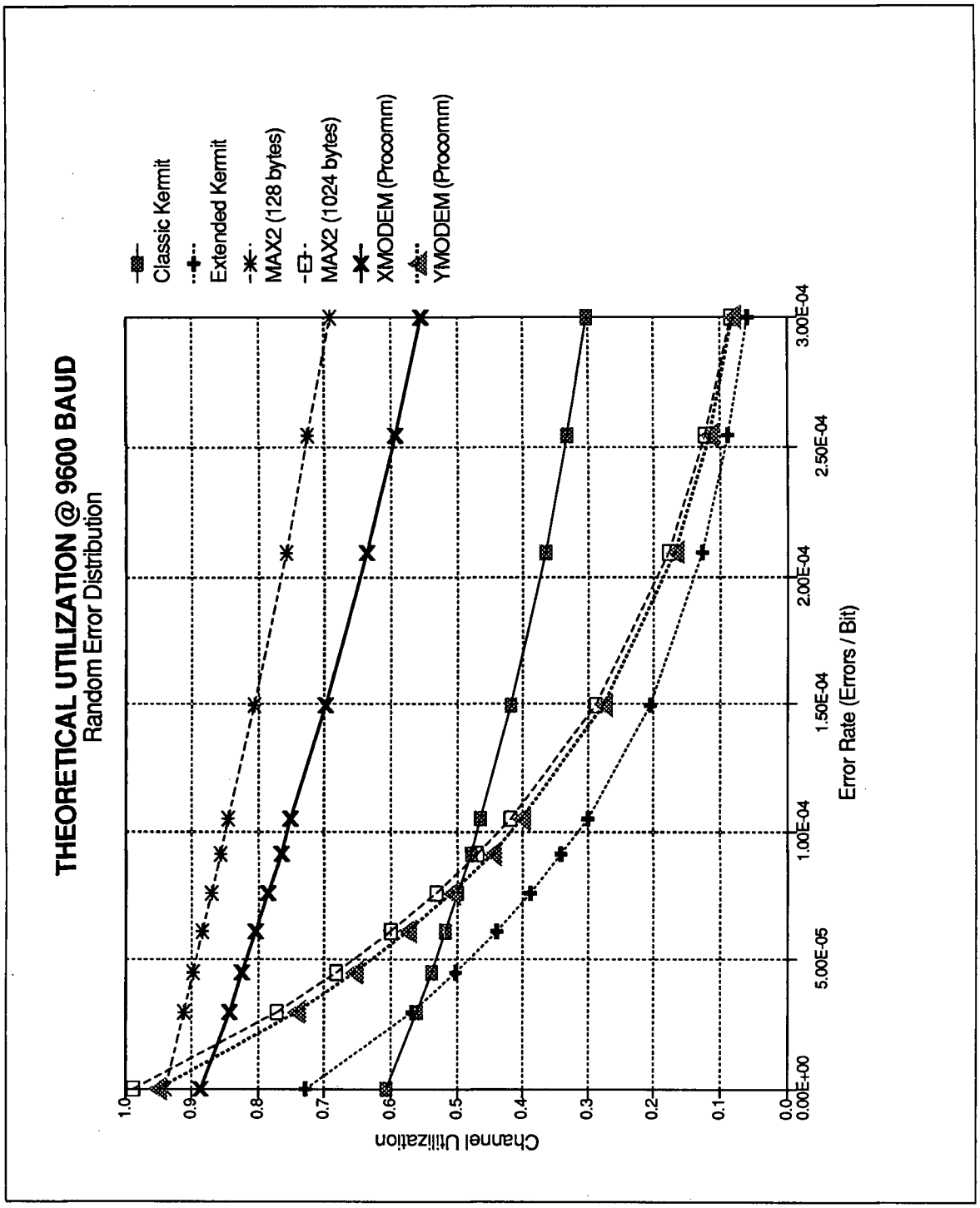
PROTOCOL	DATA PACKET SIZE	REPLY PACKET SIZE
Classic Kermit	97	8
Extended Kermit	1033	8
XMODEM-CRC	133	1
YMODEM	1029	1
ZMODEM	1027	21
MAX2 (1024)	1024	8
MAX2 (512)	512	8
MAX2 (256)	256	7
MAX2 (128)	128	7



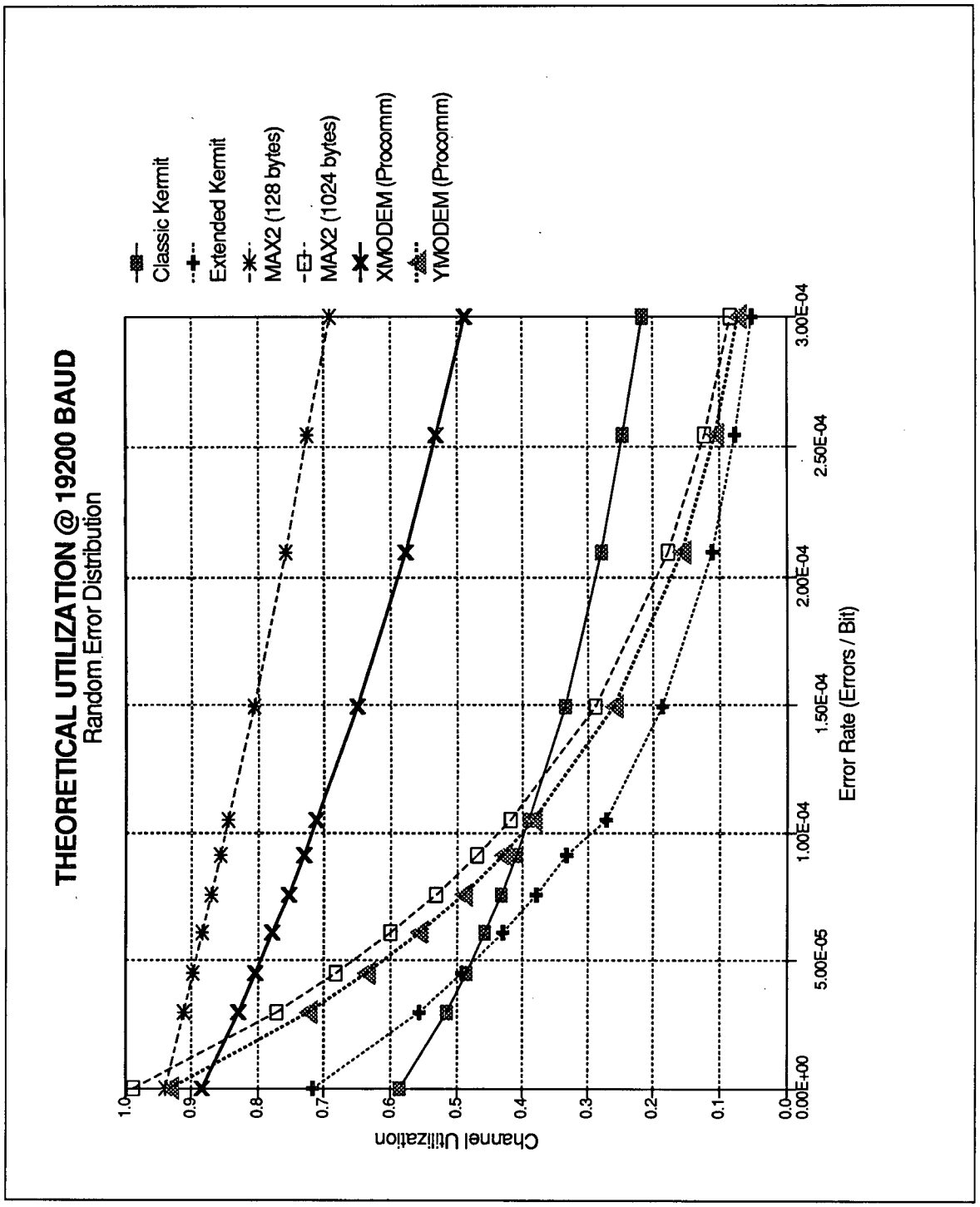
Graph 23. Theoretical Utilization At 2400 Baud And Random Error Distribution



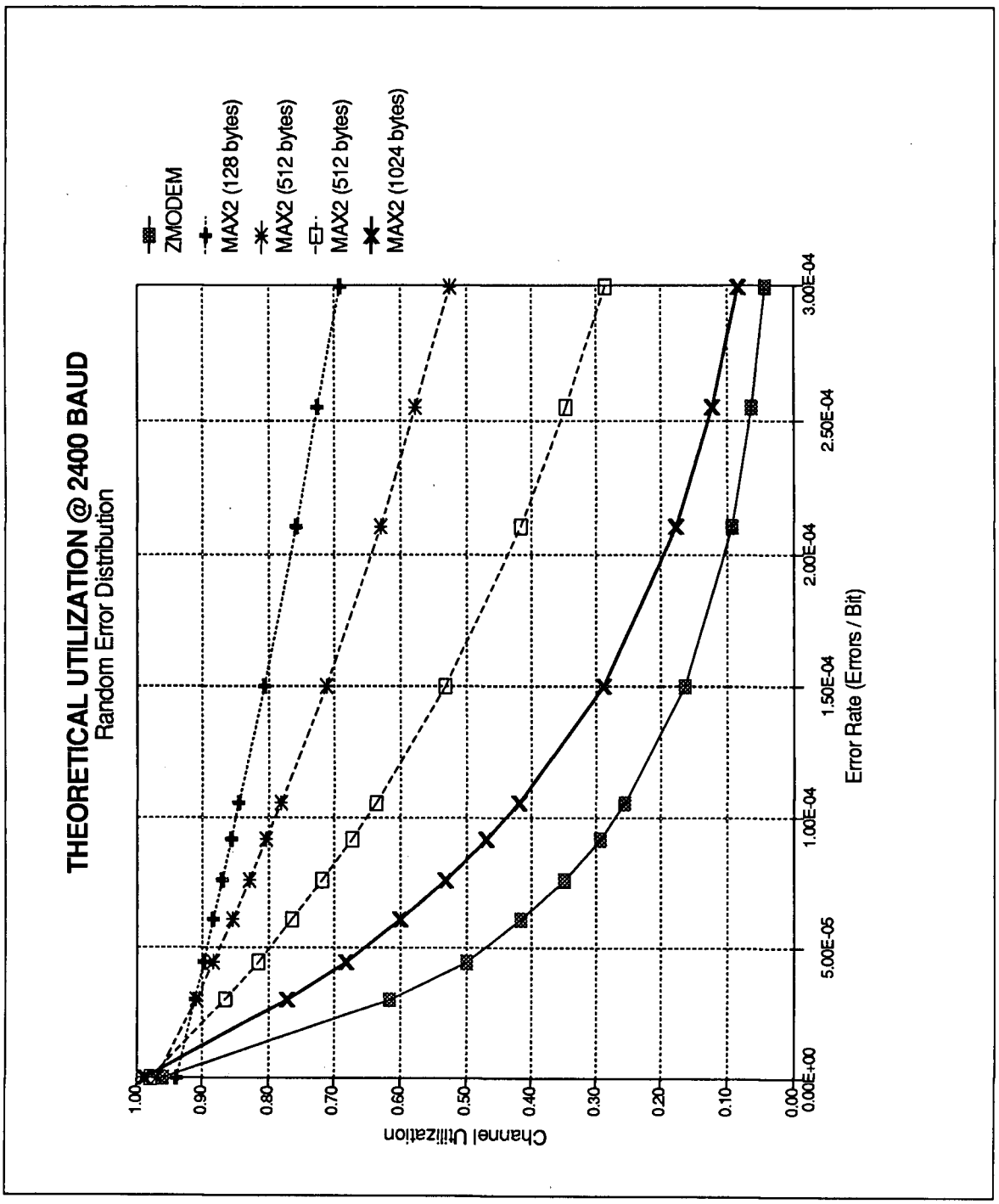
Graph 24. Theoretical Utilization At 4800 Baud And Random Error Distribution



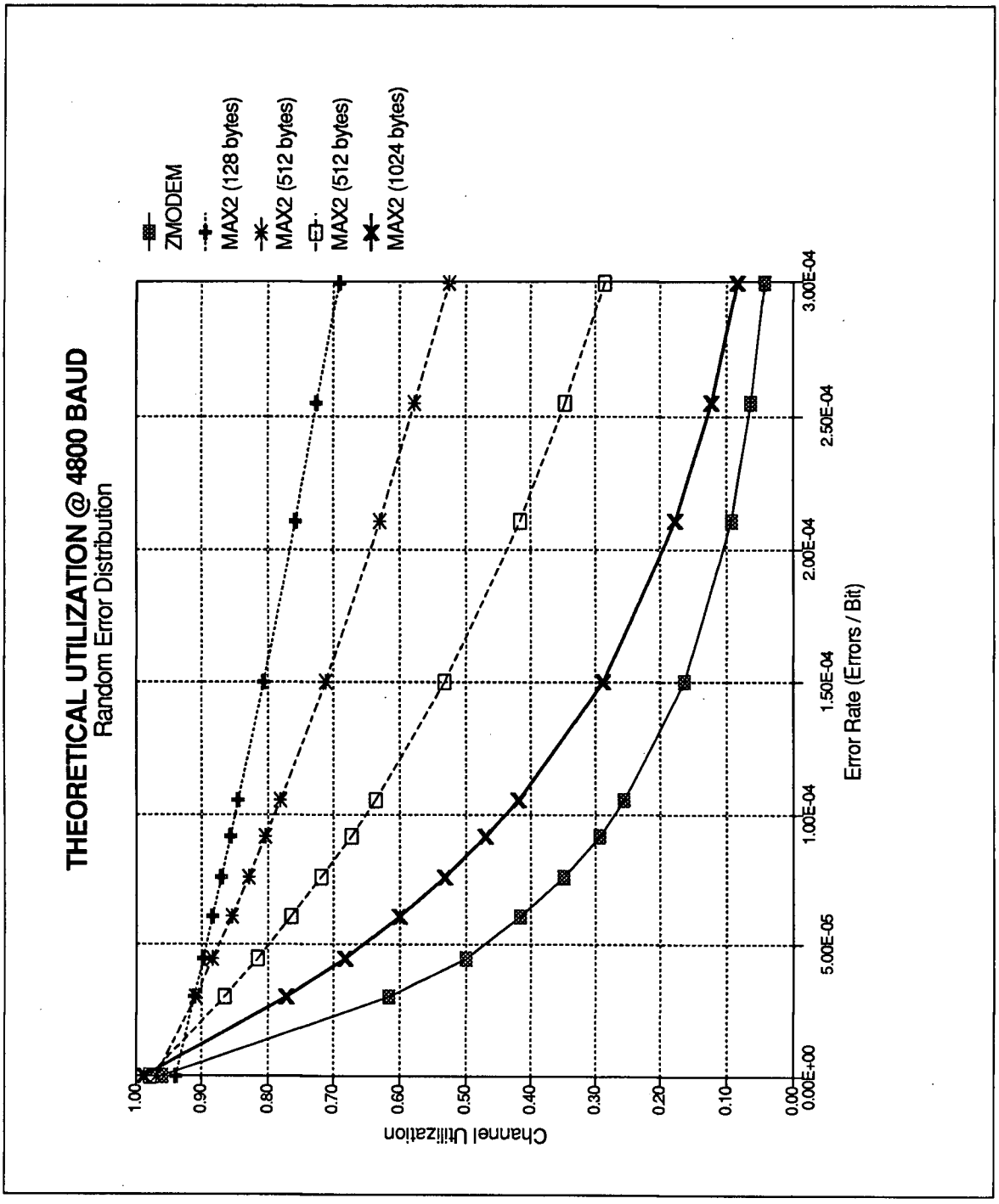
Graph 25. Theoretical Utilization At 9600 Baud And Random Error Distribution



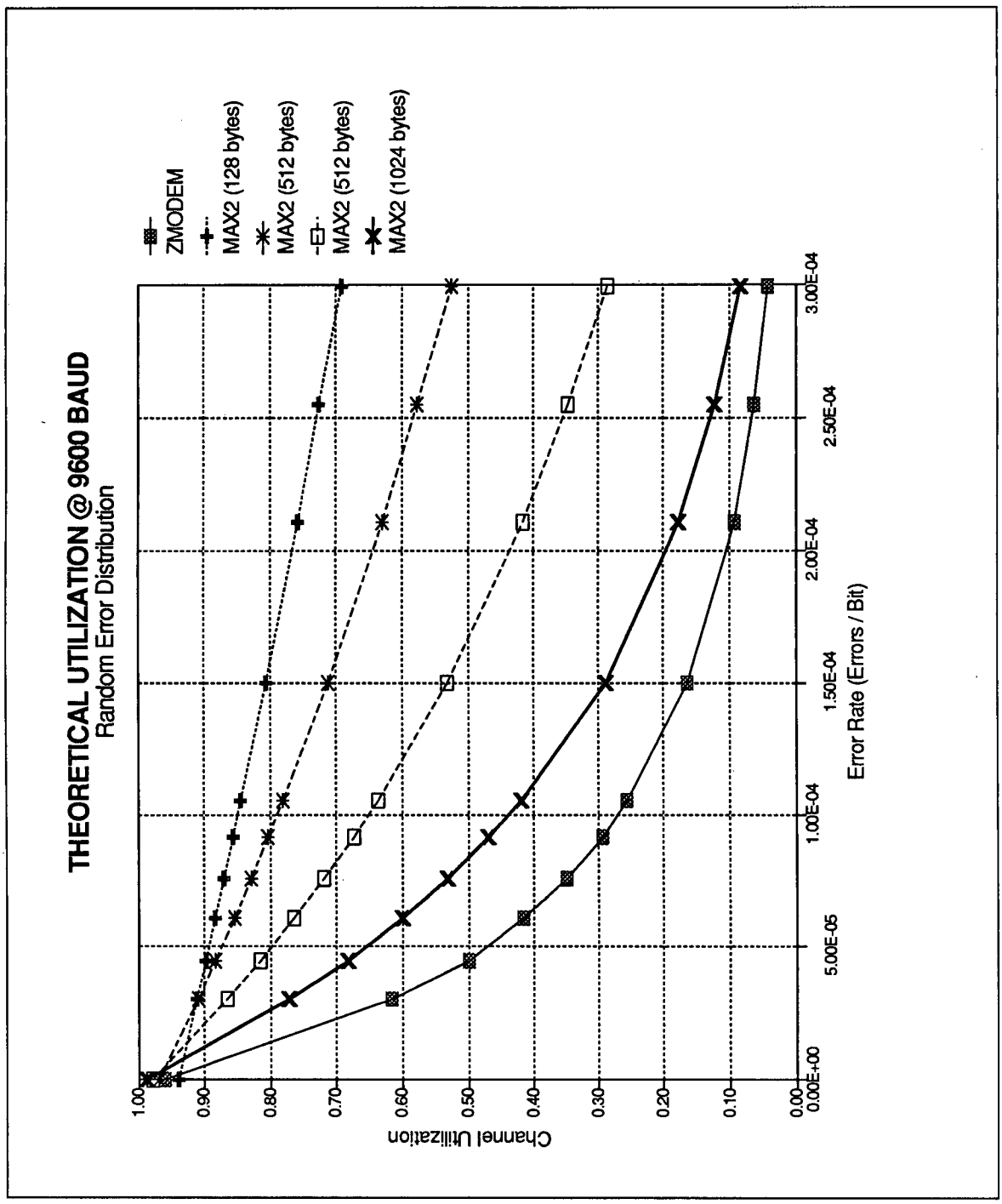
Graph 26. Theoretical Utilization At 19200 Baud And Random Error Distribution



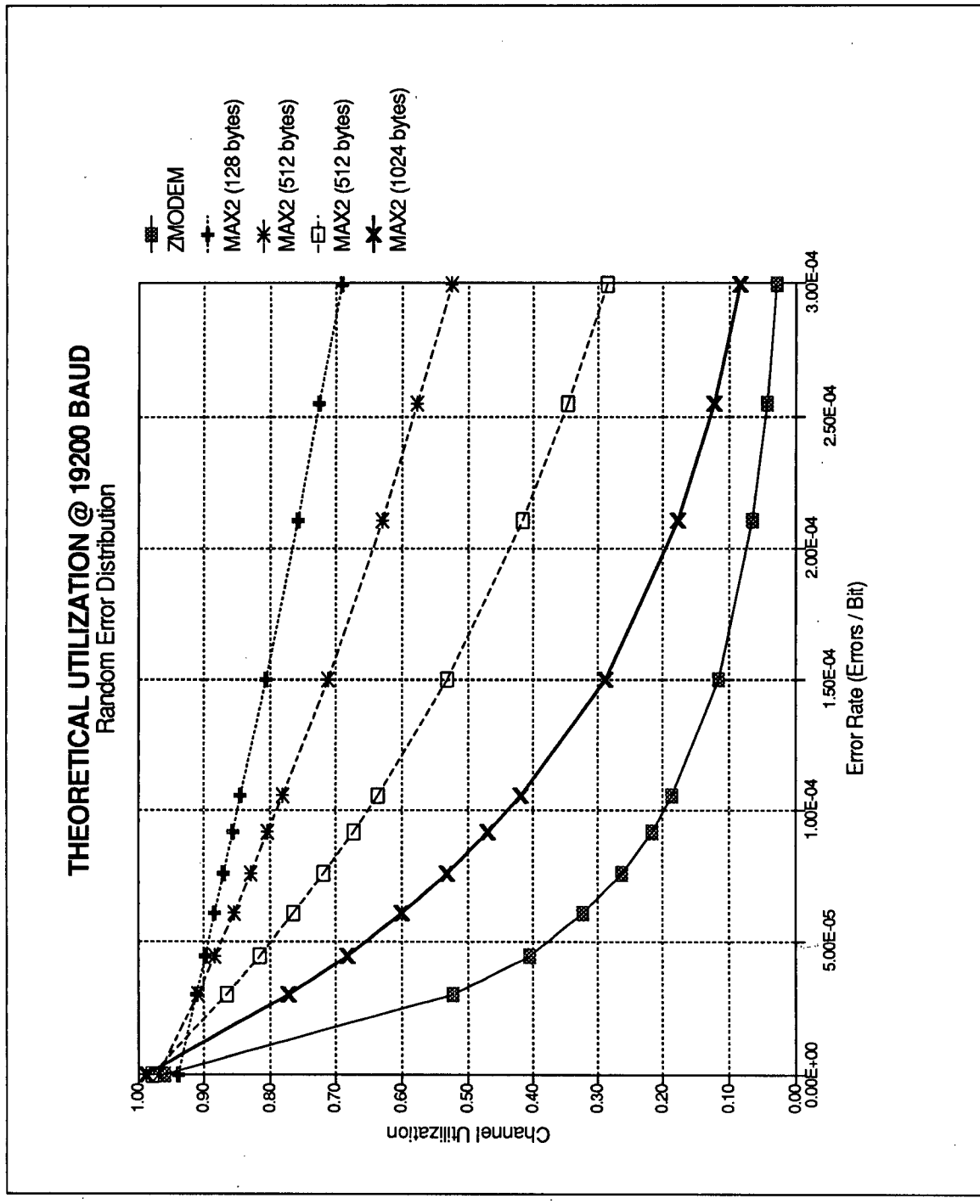
Graph 27. Theoretical Utilization (Streaming Protocols) At 2400 Baud And Random Error Distribution



Graph 28. Theoretical Utilization (Streaming Protocols) At 4800 Baud And Random Error Distribution



Graph 29. Theoretical Utilization (Streaming Protocols) At 9600 Baud And Random Error Distribution

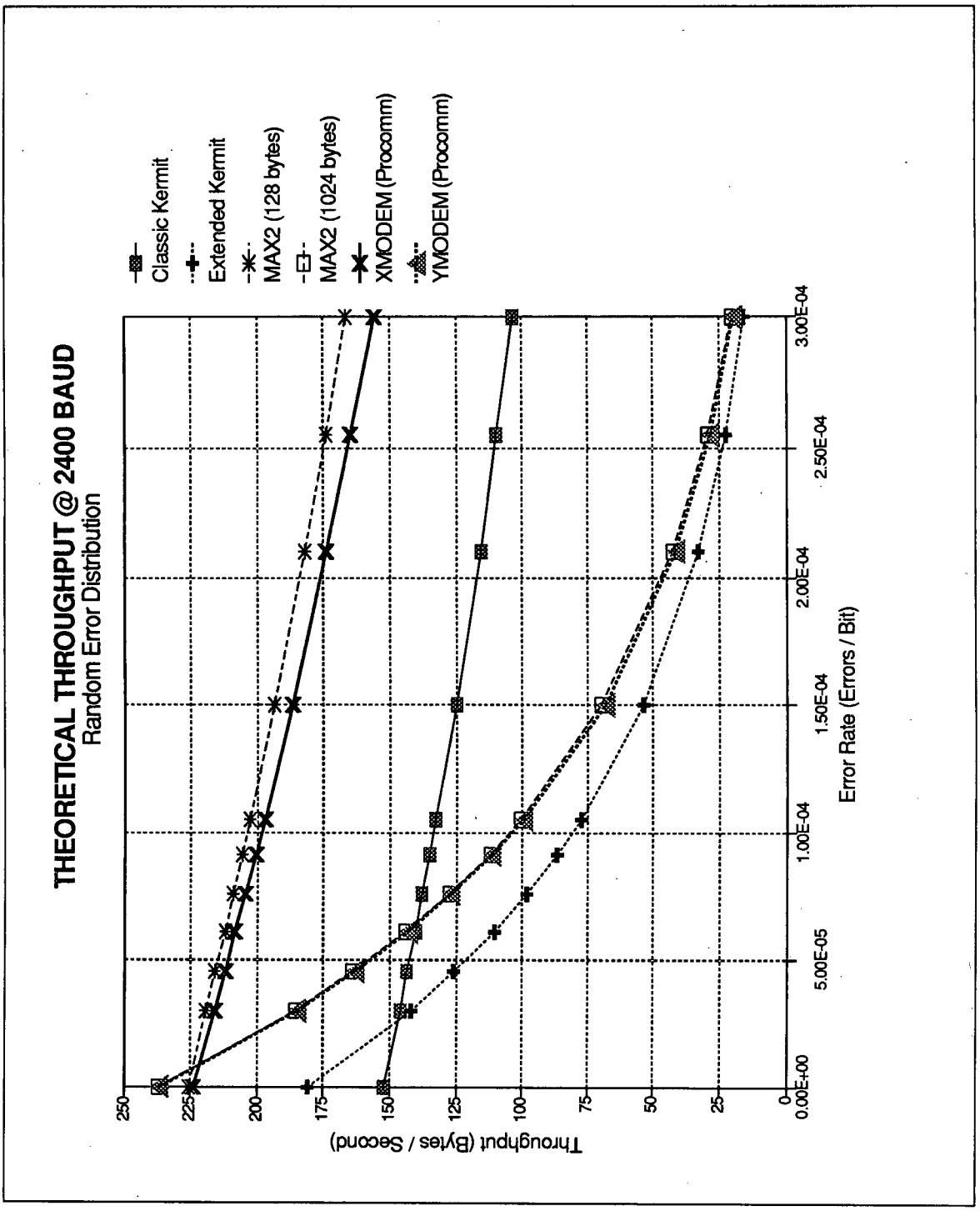


Graph 30. Theoretical Utilization (Streaming Protocols) At 19200 Baud And Random Error Distribution

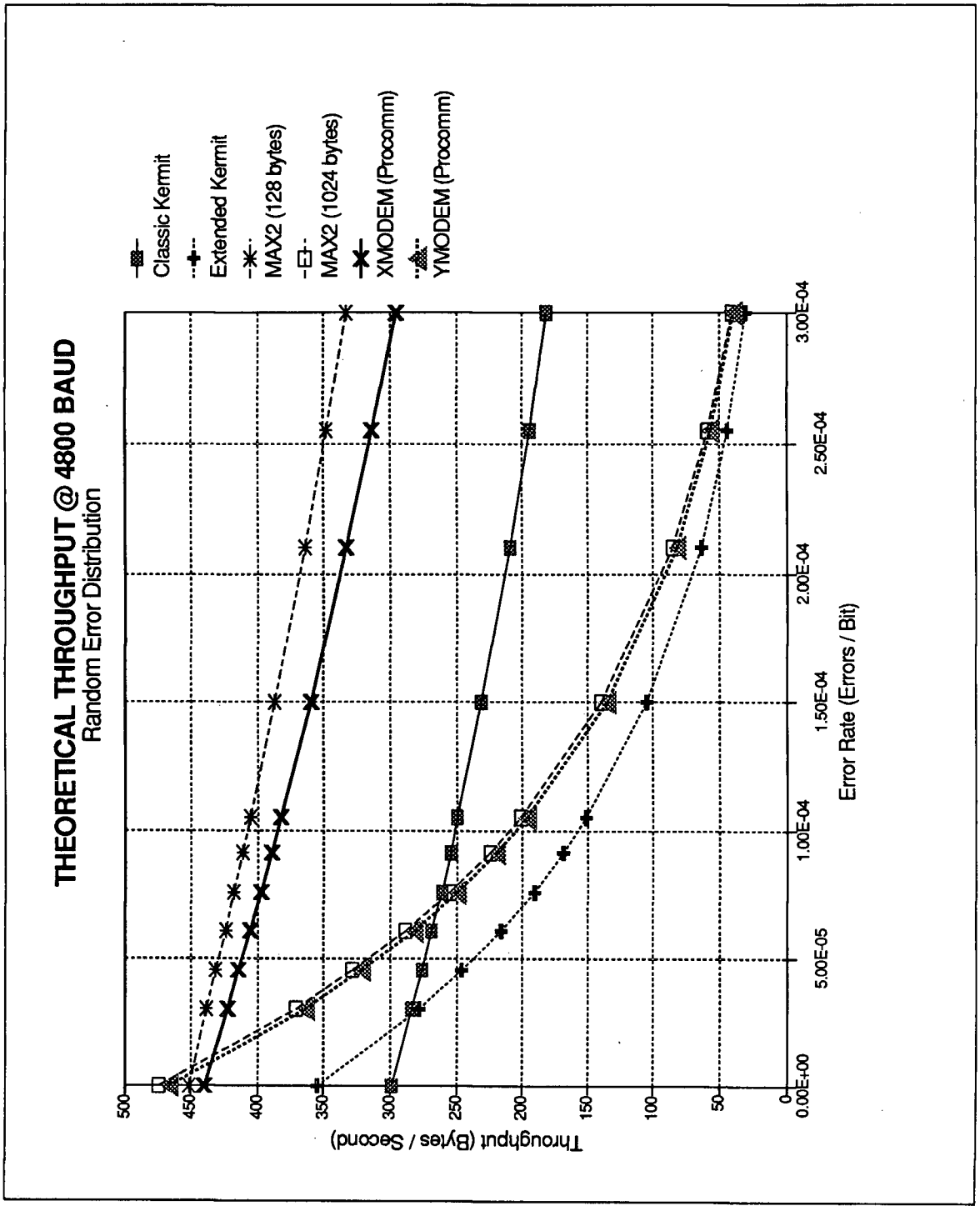
APPENDIX 6. Theoretical Throughput Graphs, Random Error Distribution

When examining the graphs in this appendix, please keep the following in mind:

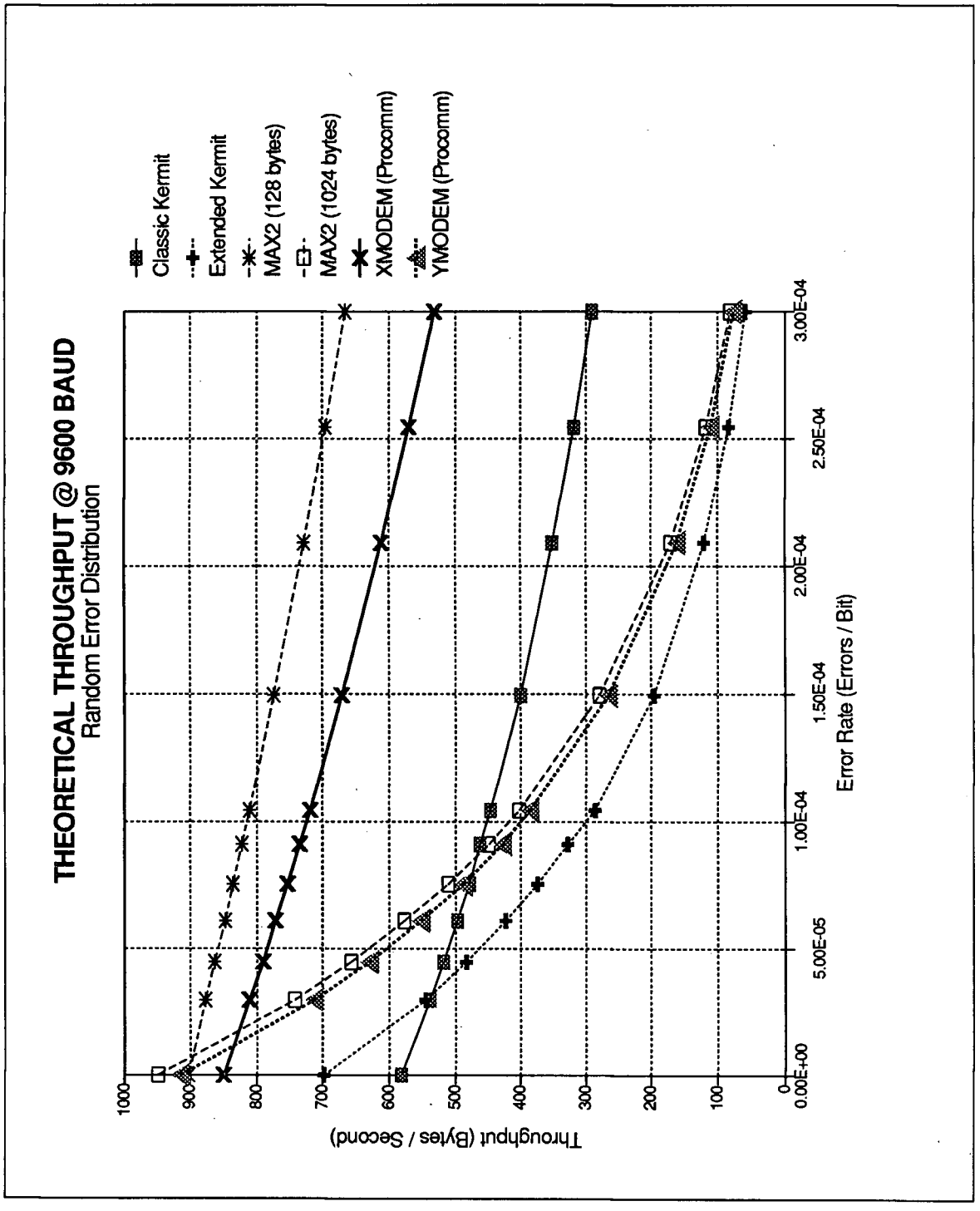
PROTOCOL	DATA PACKET SIZE	REPLY PACKET SIZE
Classic Kermit	97	8
Extended Kermit	1033	8
XMODEM-CRC	133	1
YMODEM	1029	1
ZMODEM	1027	21
MAX2 (1024)	1024	8
MAX2 (512)	512	8
MAX2 (256)	256	7
MAX2 (128)	128	7



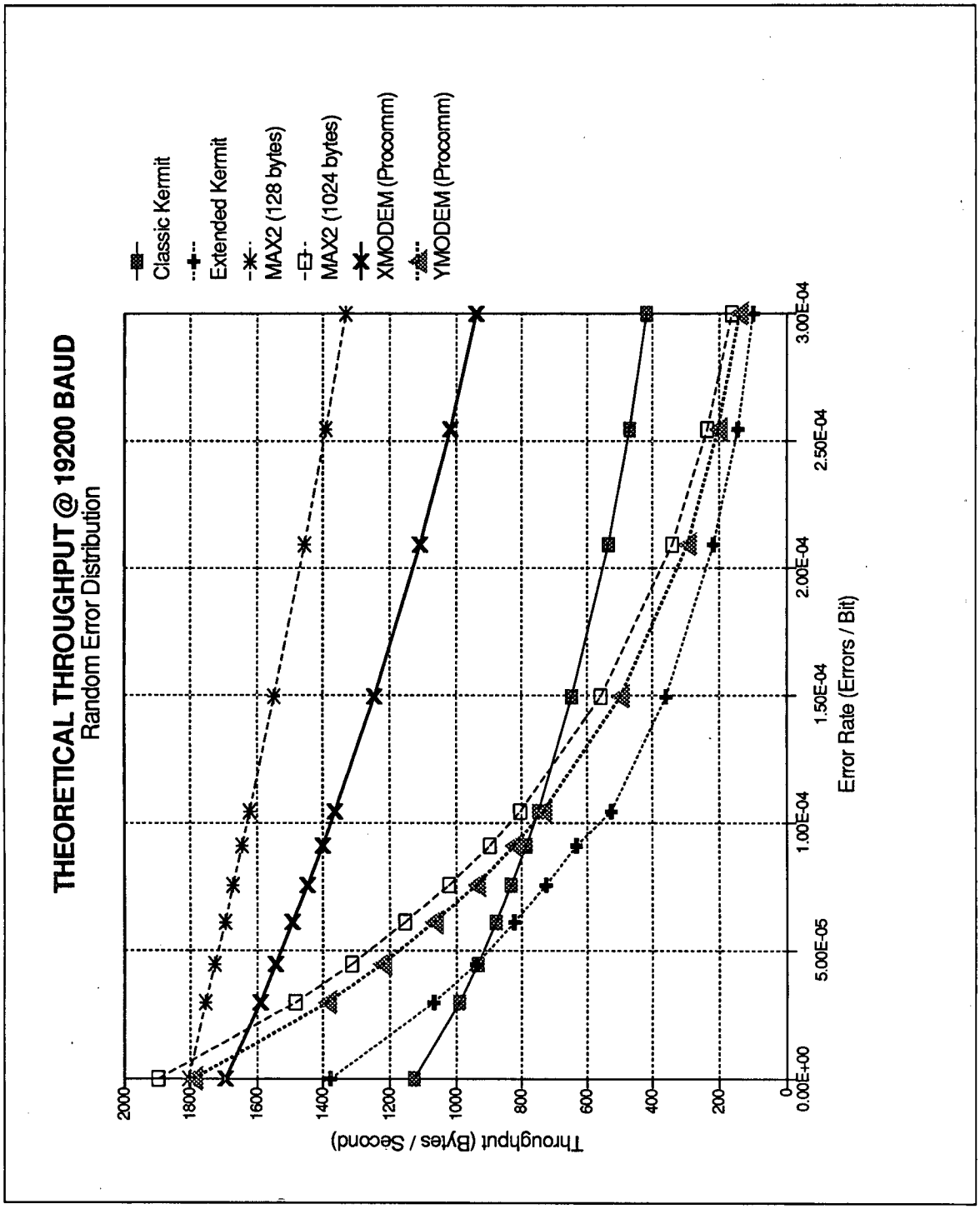
Graph 31. Theoretical Throughput At 2400 Baud And Random Error Distribution



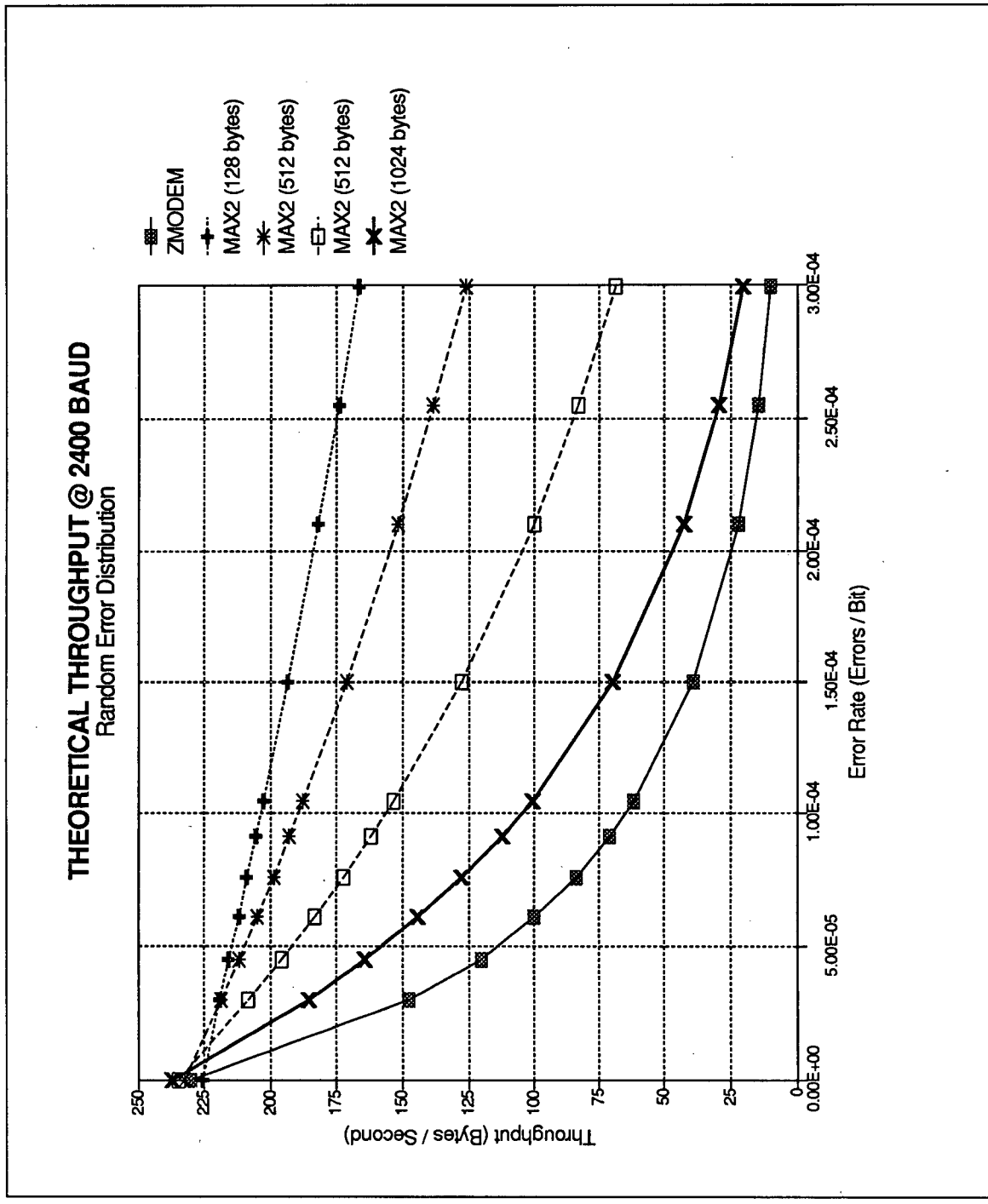
Graph 32. Theoretical Throughput At 4800 Baud And Random Error Distribution



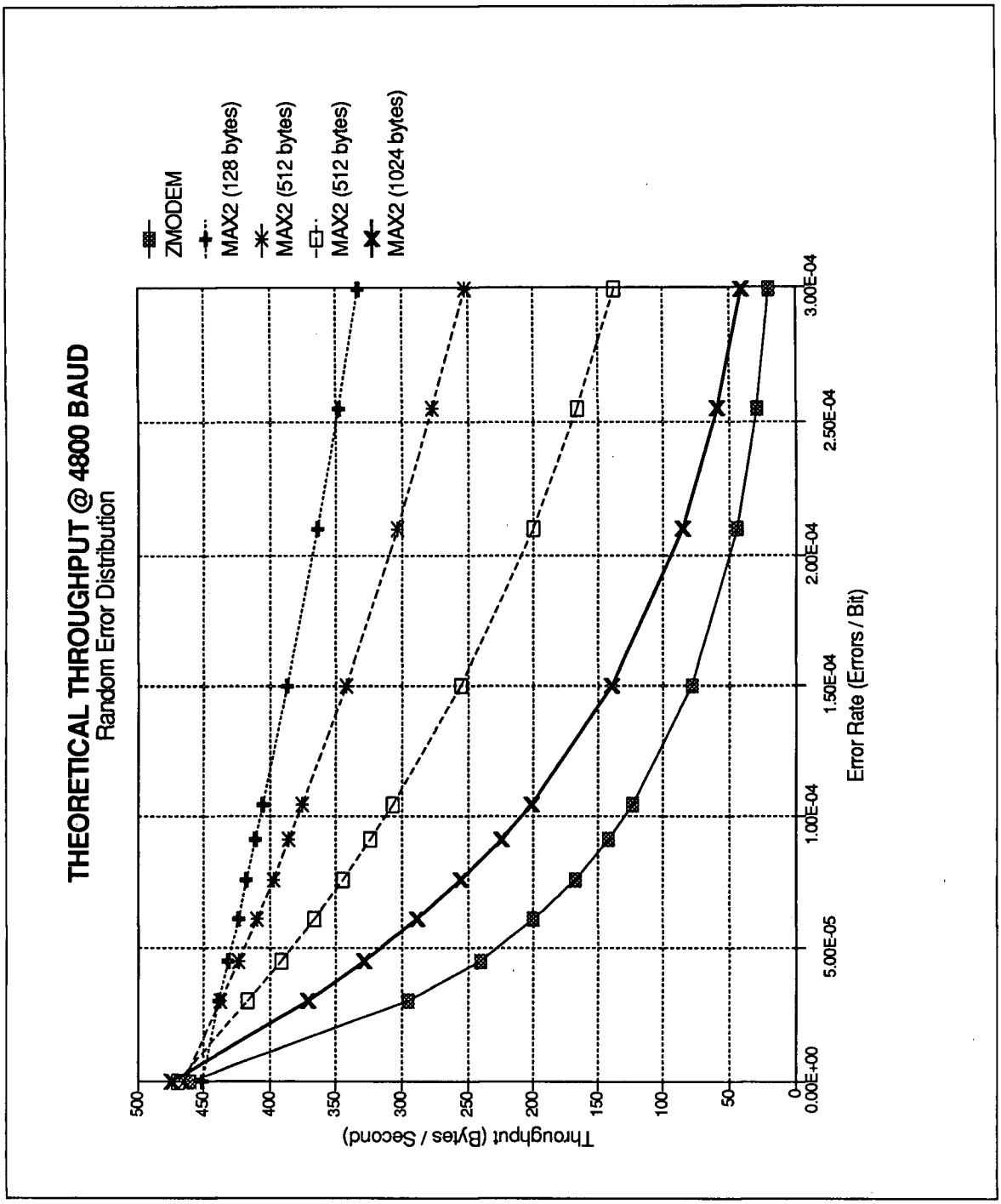
Graph 33. Theoretical Throughput At 9600 Baud And Random Error Distribution



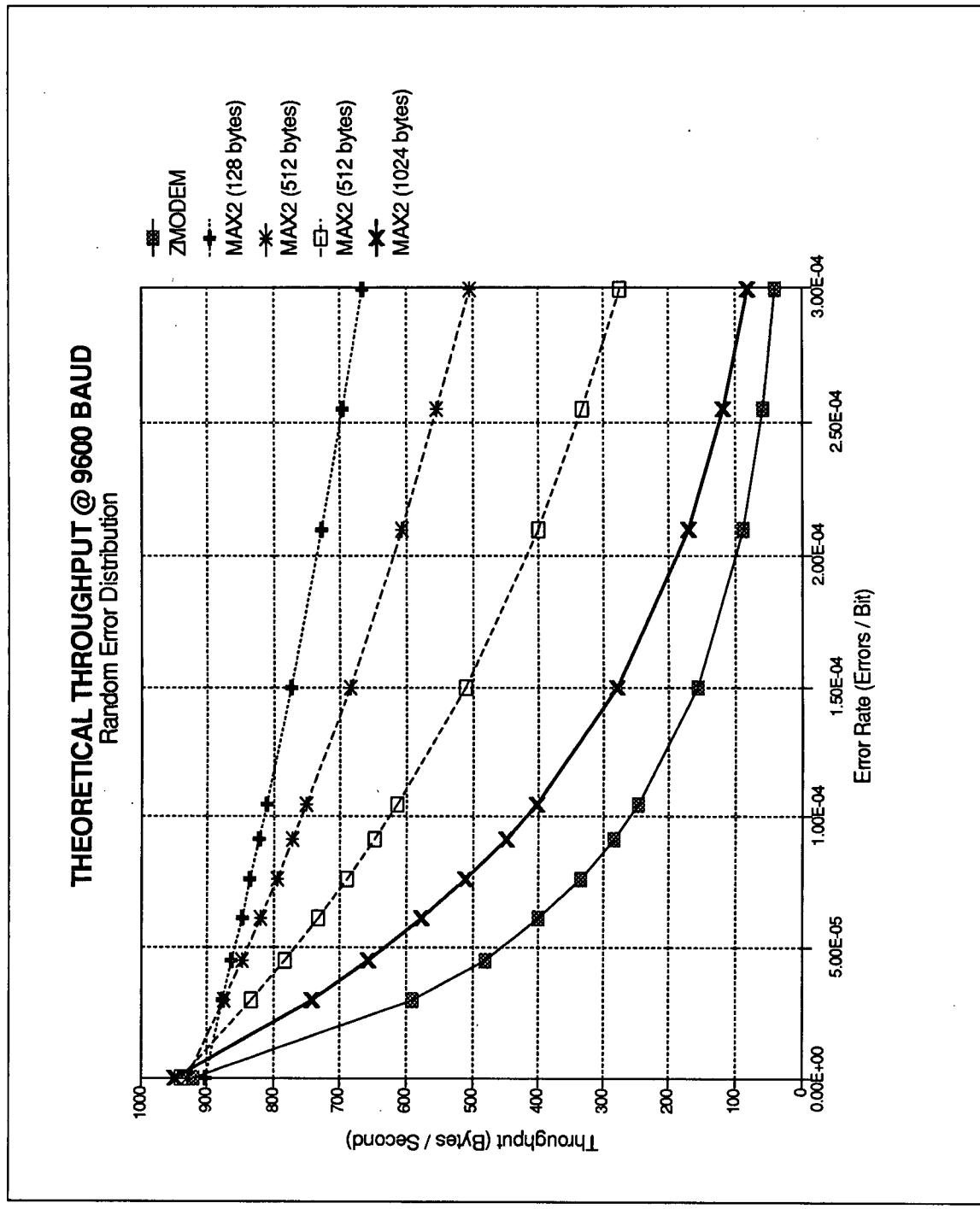
Graph 34. Theoretical Throughput At 19200 Baud And Random Error Distribution



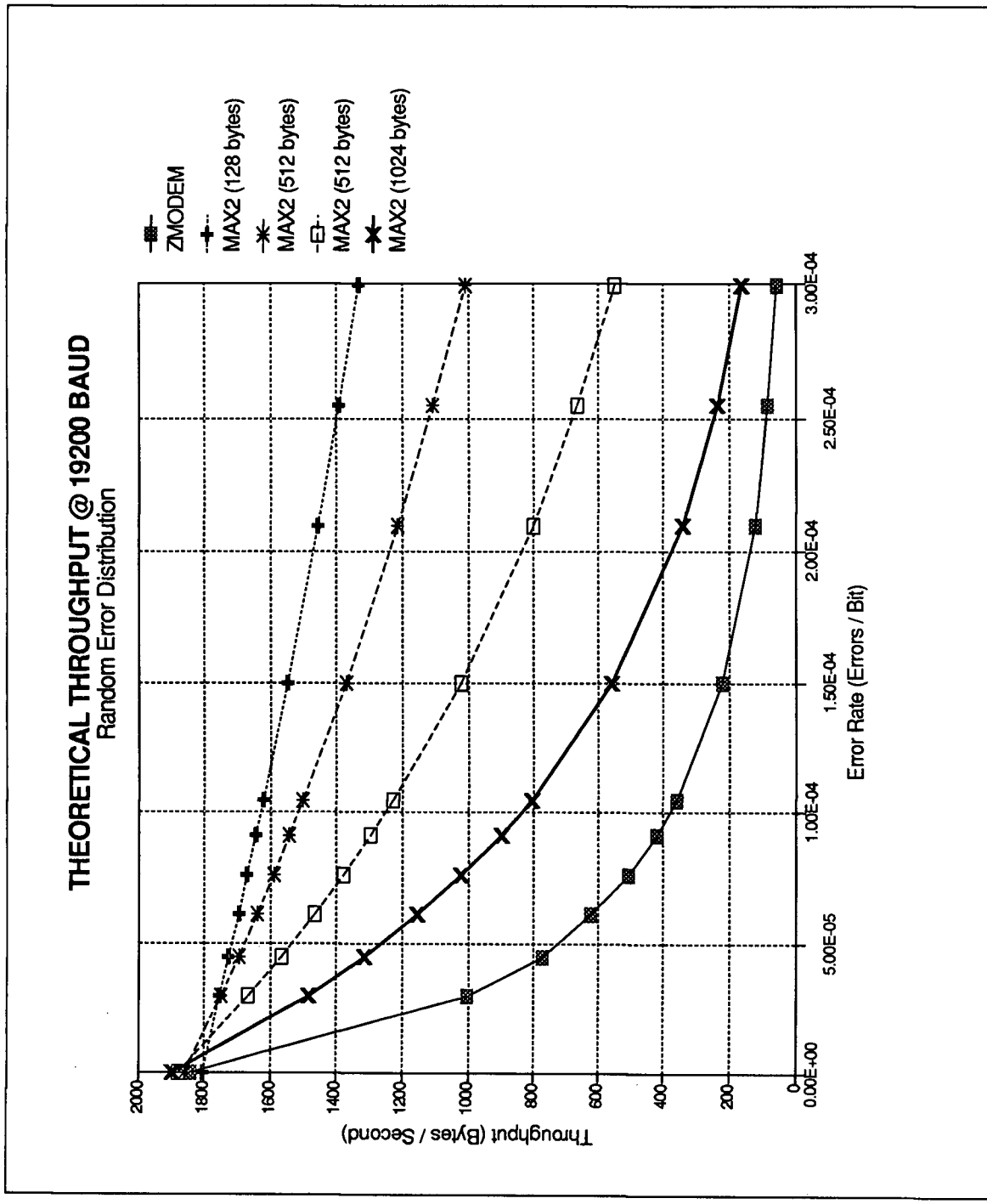
Graph 35. Theoretical Throughput (Streaming Protocols) At 2400 Baud And Random Error Distribution



Graph 36. Theoretical Throughput (Streaming Protocols) At 4800 Baud And Random Error Distribution



Graph 37. Theoretical Throughput (Streaming Protocols) At 9600 Baud And Random Error Distribution

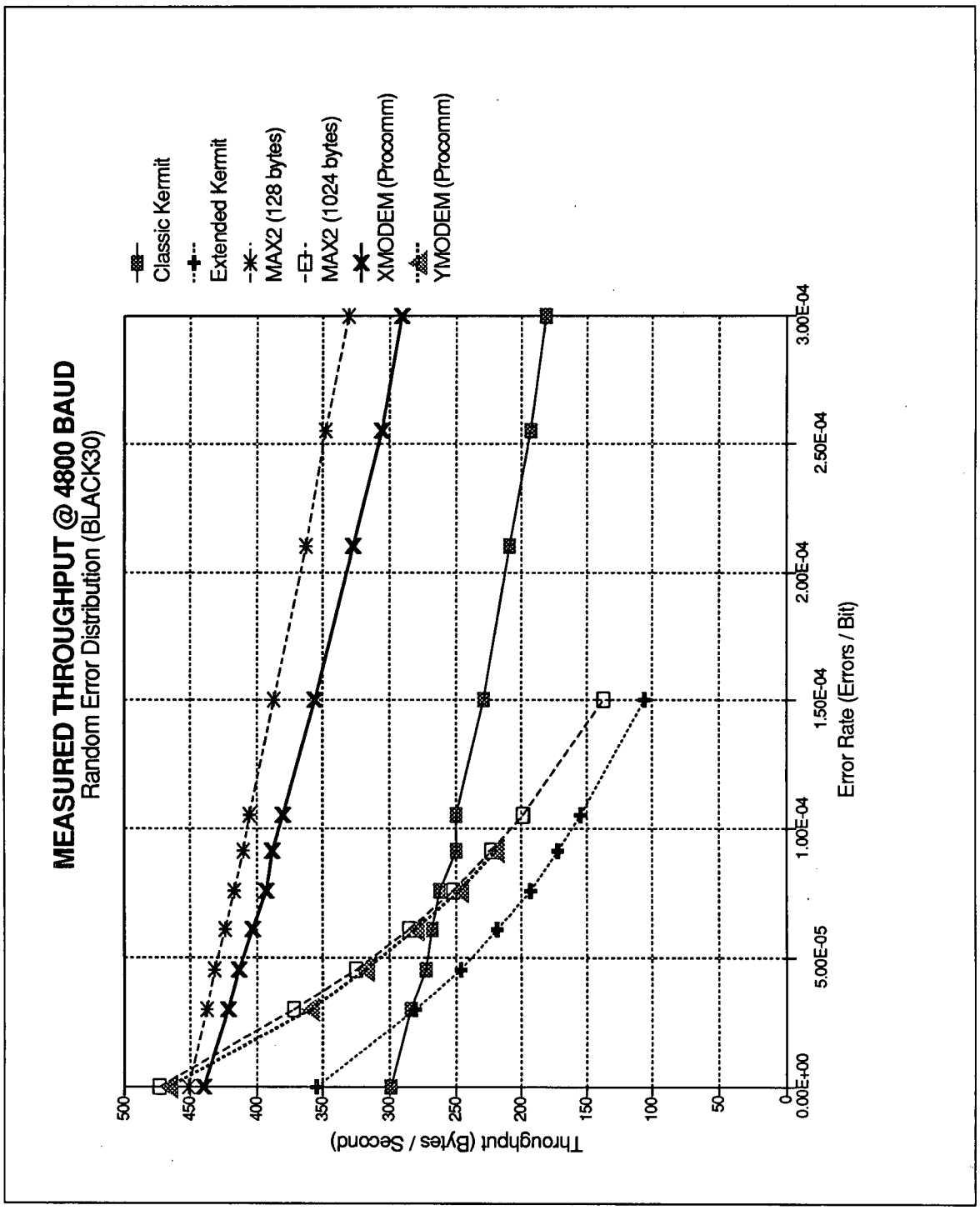


Graph 38. Theoretical Throughput (Streaming Protocols) At 19200 Baud And Random Error Distribution

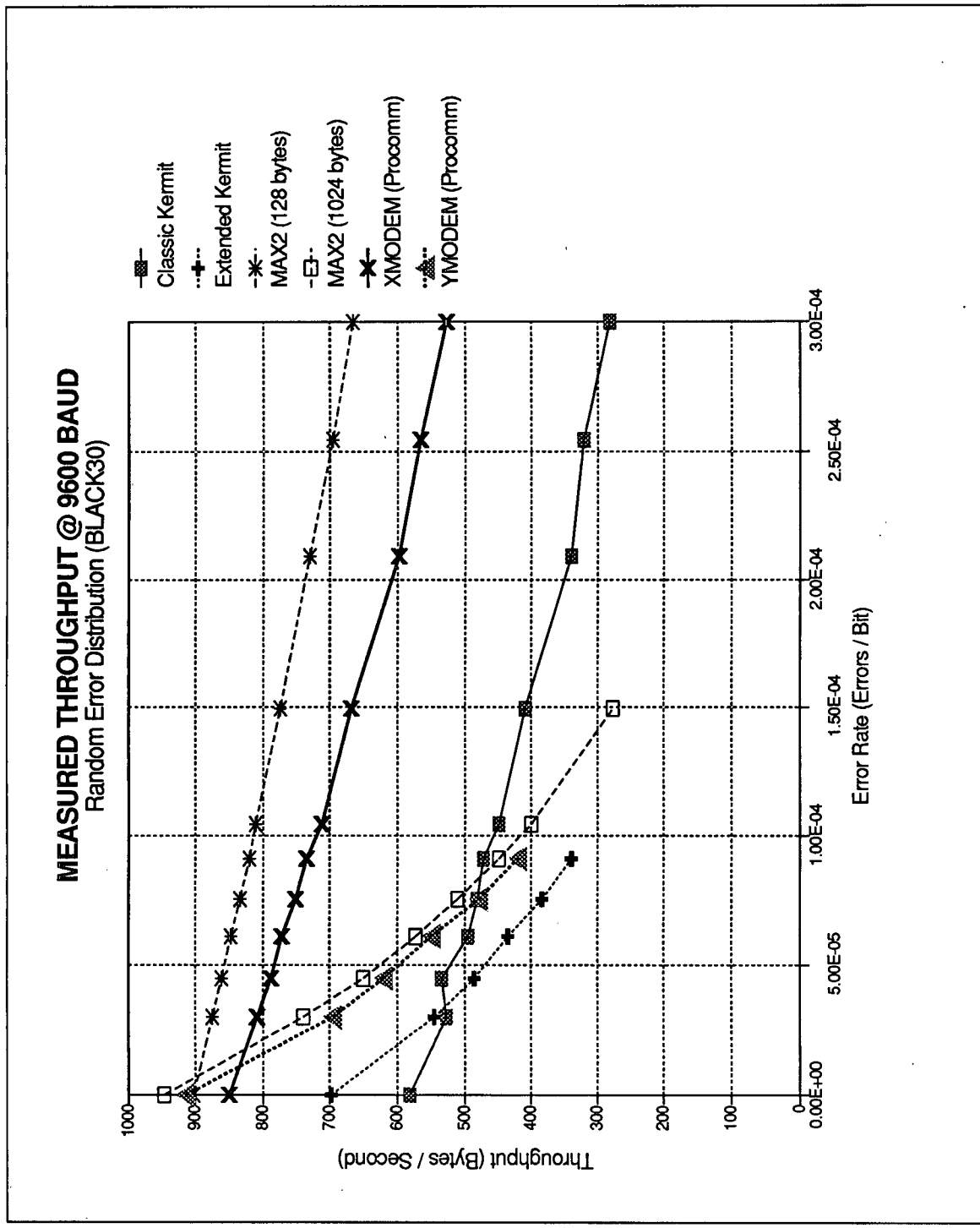
APPENDIX 7. Measured Throughput Graphs, Random Error Distribution

When examining the graphs in this appendix, please keep the following in mind:

PROTOCOL	DATA PACKET SIZE	REPLY PACKET SIZE
Classic Kermit	97	8
Extended Kermit	1033	8
XMODEM-CRC	133	1
YMODEM	1029	1
MAX2 (128)	128	7
MAX2 (1024)	1024	8



Graph 39. Measured Throughput At 4800 Baud And Random Error Distribution

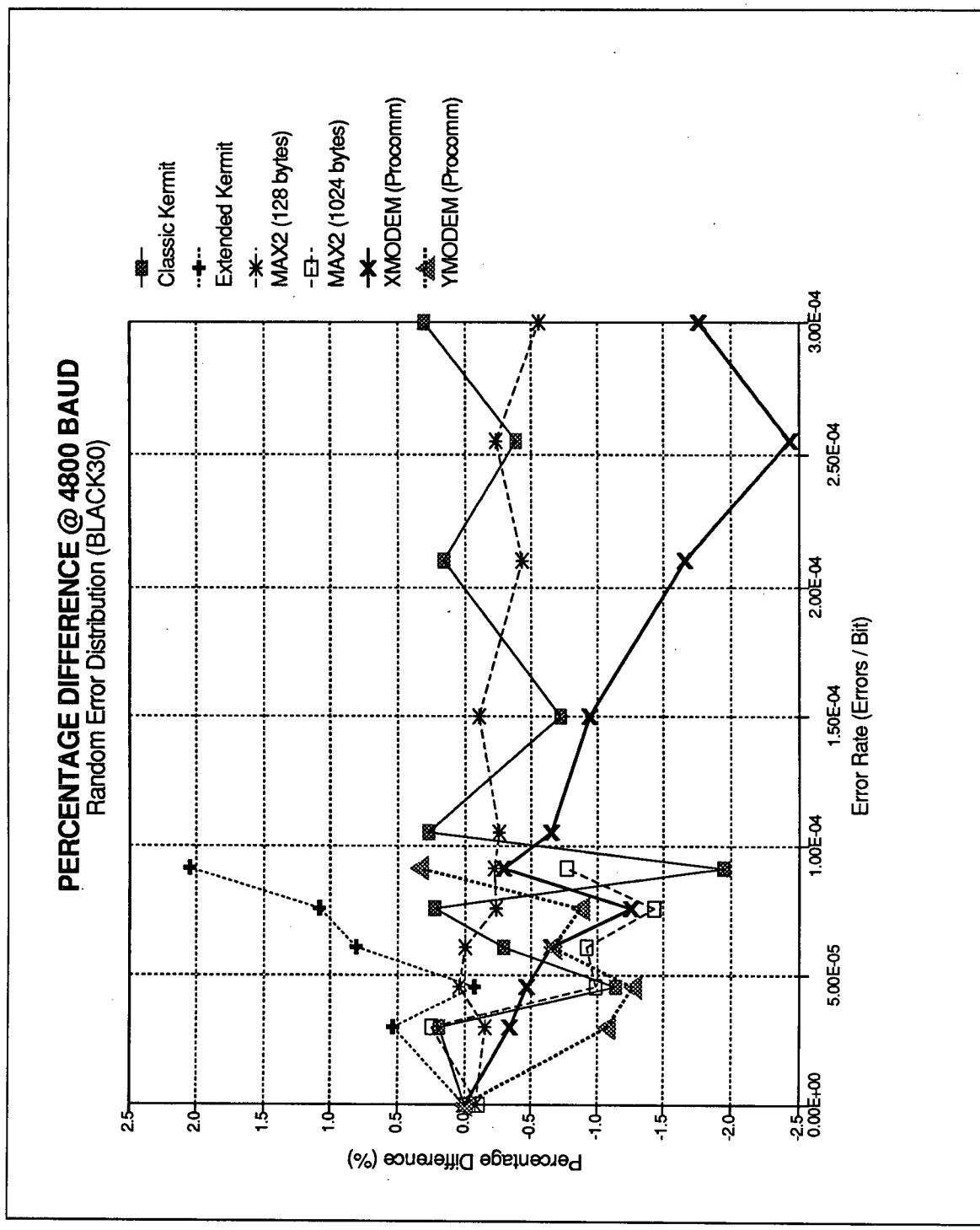


Graph 40. Measured Throughput At 9600 Baud And Random Error Distribution

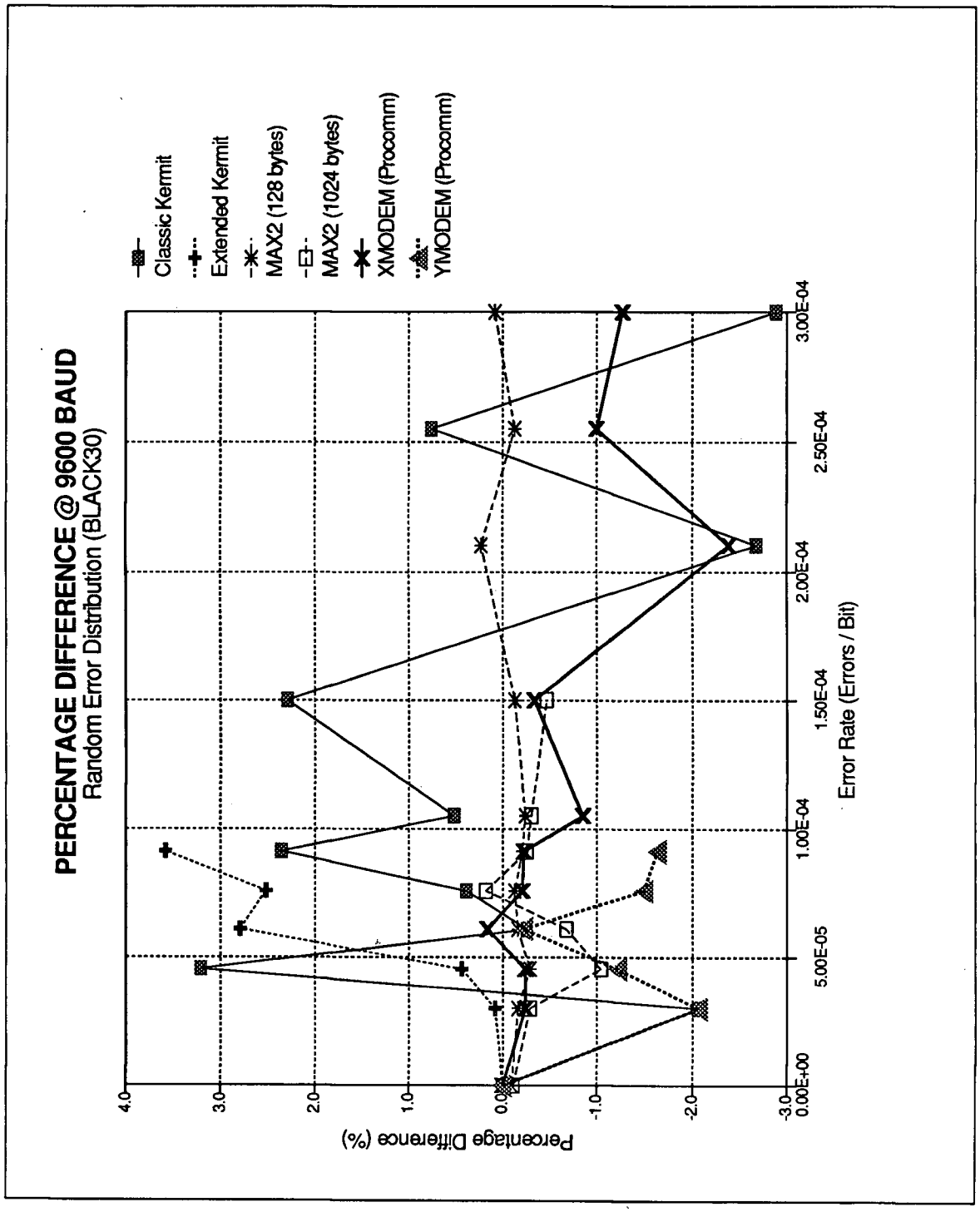
APPENDIX 8. Percentage Difference Graphs, Random Error Distribution

When examining the graphs in this appendix, please keep the following in mind:

PROTOCOL	DATA PACKET SIZE	REPLY PACKET SIZE
Classic Kermit	97	8
Extended Kermit	1033	8
XMODEM-CRC	133	1
YMODEM	1029	1
MAX2 (128)	128	7
MAX2 (1024)	1024	8



Graph 41. Percentage Difference At 4800 Baud And Random Error Distribution



Graph 42. Percentage Difference At 9600 Baud And Random Error Distribution