170
122

EVALUATION OF ANSI COMPRESSION IN A BULK DATA FILE TRANSFER SYSTEM

by

Douglas Gary Chaulklin

Project report submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Systems Engineering

APPROVED:

Ben S Blanchard

Benjamin S. Blanchard, Chairman

L. K Harmon

L. Kenneth Harmon, Jr.

C. Patrick Koelling

May 1991
Blacksburg, Virginia

EVALUATION OF ANSI COMPRESSION IN A BULK DATA FILE TRANSFER SYSTEM

by

Douglas Gary Chaulklin

Committee Chairman: Benjamin S. Blanchard

Department of Industrial and Systems Engineering

(ABSTRACT)

This report evaluates the use of a newly proposed American National Standard Institute (ANSI) standard for data compression in a bulk data transmission system. An overview of the transmission system, the current compression method, and the ANSI algorithm are presented. A dynamic systems model is used to analyze the benefits and impacts of various alternatives to addressing the needs of the system. A decision model is built to summarize the alternatives based on the perceived problem contexts.

ACKNOWLEDGEMENTS

During the time this paper was written, our nation was involved in a war in the Persian Gulf. Our president has asked the people to pray for our nation. We should also acknowledge not only His help in large matters like the Persian Gulf, but also small matters like the writing of this paper. I would also like to thank Liza, my loving and supportive wife whose work at home allowed this paper to be completed.

TABLE OF CONTENTS

ABSTRACT ii

ACKNOWLEDGEMENTS iii

CHAPTER 1 - INTRODUCTION 1
Project Introduction 1
Terminology 3
The Federal Reserve 4
Types of Payments 5
Financial Institution Connectivity 5
The BulkData System 6
Data Compression 9

CHAPTER 2 - SCOPE 14

CHAPTER 3 - NEEDS ANALYSIS 16

CHAPTER 4 - BULKDATA FILE TRANSFER SYSTEM MODEL 18
Purpose of the model 18
Problem with the System 20
Model Conceptualization 21
Model Construction 24
Model Evaluation 24
System Model Output 26

CHAPTER 5 - EVALUATION OF ALTERNATIVES 40
System Operational Requirements 40
System Maintenance Concept 41
CPU Usage Evaluation 41
Cost Allocation Comparison 43
Dial Line Costs 45
Relative Line Costs 45
X9.DC Implementation Costs 46
Criteria for Decision Making 46
Context for Decision Making 47

Contextual Decision Model 48

CHAPTER 6 - OPTIMIZATION MODEL 51
ANSI X9.DC Algorithm Tutorial 51
Test Methodology 52
ANSI X9.DC Parameter Optimization 53

CHAPTER 7 - MANAGEMENT RECOMMENDATIONS 57
Conclusions 57
Maintenance Approach 57
Development Approach 58
Proposed Project Schedule 58

CHAPTER 8 - PROJECT BENEFITS 60
Operating Costs 60
Quality of Service 61
Promoting the Standard 61
BulkData Computer Model 61
BulkData Preprocessor Problem 62

REFERENCE LIST 63

APPENDIX A - DRAFT STANDARD X9.DC 65

APPENDIX B - MODEL LISTING 84

APPENDIX C - COMPRESSION TEST ROUTINE 113

CHAPTER 1 - INTRODUCTION

Project Introduction

This project evaluates the feasibility of replacing the existing data compression programming in a bulk data file transmission system called BulkData with an algorithm proposed in a draft ANSI standard. Specific problems with the current BulkData system are defined and analyzed that might be alleviated by implementing a better data compression algorithm.

A prototype ANSI compression computer program compatible with BulkData was written to establish a basis for comparing the ANSI data compression algorithm with the existing BulkData algorithms. This prototype computer program was used as a point of reference in developing and validating a computer model that simulates the operation of BulkData with the new data compression algorithm.

First, a computer model was built that could accurately predict the current operation of BulkData. This computer model was then expanded to predict impacts and benefits of implementing the ANSI data compression algorithm.

The project provides an evaluation of alternatives with an emphasis on weighing all relevant decision factors and integrating these factors in a decision table. Typical quantitative decision factors such as cost are rated and combined with more qualitative measures such as ease of operation.

Based on the conclusion that the ANSI data compression algorithm should be implemented, a method was developed to optimize the parameters of the algorithm. The prototype computer program was used to find the optimum values for each parameter.

A strategy and approximate schedule for implementing the ANSI algorithm are provided. The schedule must be modified based on resource availability and software release dates.

Terminology

The following lists several terms that are used throughout this paper. These terms are defined in the context of this particular project.

bit

The smallest unit of information stored in digital, binary computer systems. Bits are binary codes that can only take on the values zero (0) or one (1). Generally, eight of these units or bits are required to represent a single uncompressed character. When bits are grouped together in bytes, each bit is numbered from zero to seven. The value of each bit then gets multiplied by $2^0, 2^1, \ldots, 2^7$ in the very same fashion applied to decimal numbers.

byte

Characters in digital computers are represented in codes made up of a constant number (usually eight) of bits. Each possible character of information (such as the letter "a" or the symbol "&") in the particular codeset used by the computer system is assigned to one of these bytes.

codeword

Data compression algorithms encode one or more bytes of data into a single unit of information called a codeword. Each codeword consists of a number of bits that is greater than the number of bits in a single character or byte.

Systems Network Architecture

A data communications architecture created by IBM that provides a set of protocols and formats that enables a single physical data communications network to be shared by many independent applications.

T1

A digital telecommunications service that provides a communications path with an aggregate line speed of 1.544 million bits per second.

The Federal Reserve

The Federal Reserve System serves as the central bank for the United States. Like many central banks around the world, the Federal Reserve System has the largest electronic payment system in its country. There are about 7,000 financial institutions that are connected to the Federal Reserve System's electronic payment network.

The Federal Reserve System is divided into twelve district banks, each responsible for the financial institutions in their geographic area. Most district banks have branches and regional check processing centers. All of these Federal Reserve offices are connected by a single 56 KBPS (56,000 bit per second) high speed communications network called FRCS-80 (Federal Reserve Communication System of the 1980s). This network will be replaced in about three years with a T1 (1.544 million bit per second) network called Fednet.

Additional sites serve as backup for contingency operation of a district's computer facilities. Districts transmit all critical transactions and data bases to their contingency sites in case of a prolonged district computer outage.

Types of Payments

From a computer systems perspective there are two types of electronic payments. One is the familiar funds transfer which has the distinguishing characteristics of immediate payment (a minute or two) and a large dollar value per transaction (current average is over a million dollars).

The second type of payment is known as a "batch" payment. Batch payments are a group of smaller payments sent to financial institutions that eventually cause many bank accounts to be updated with the payment values specified in the batch. Several associations in the United States participate in the standardization of these payments known as ACH (for Automated Clearing House) payments.

Financial Institution Connectivity

Financial institutions have different types of connections to their local district office based on the volume of data they send and receive. Small volume financial institutions use a standard PC software package called Fedline and a dial (switched) line. Medium volume financial institutions also use Fedline software and a PC, but are connected to their district office by a multidropped lease line. (A multidropped line is shared by several financial institutions in the same geographic area.) The highest volume

financial institutions are connected by dedicated (not shared) lines.

The fixed monthly cost to the financial institution for electronic payment services provided by the Federal Reserve System is based on connection type and line speed. Variable costs are based on the financial institution's volume and the payment type (for example, ACH payment or funds transfer).

The BulkData System

BulkData is a standard application developed by the Federal Reserve System for transferring files (ACH files for example) between offices within a district, between district head offices, and between a Federal Reserve office and a financial institution. BulkData implements IBM's System Network Architecture (SNA) protocols to communicate with a variety of endpoints (financial institutions, government, and other Federal Reserve offices).

From a processing perspective BulkData is divided into three major functions to accomplish file preparation before sending, communications, and processing received files as shown in Figure 1. These three processing steps are abbreviated as BDPRE, BDUTIL, and BDPOST, respectively. A summary of these three functions is given here as an

introduction to the more comprehensive discussion needed for developing a current system model.

The BulkData preprocessor, BDPRE, receives files from hundreds of different applications for transmission. These applications and files are called "user" application and files to distinguish them from the BulkData application and internal files required to manage the transmission of these user files. BDPRE reformats files into compressed segments of a common size suitable for transmission. The reformatted file is queued for transmission to a specific destination by the BulkData transmission utility, BDUTIL.

The BulkData transmission utility, BDUTIL, is an SNA application that has two primary functions related to sending and receiving files transmitted over the data communication network. For files on the outbound queue, BDUTIL establishes a connection with the specified destination and sends the file in segments. A checkpoint process is used to be able to restart the transmission of any file from the last segment that was acknowledged by the destination.

The other primary function receives incoming files from the data communication network. An acknowledgement is given for each segment after recording its receipt in a checkpoint

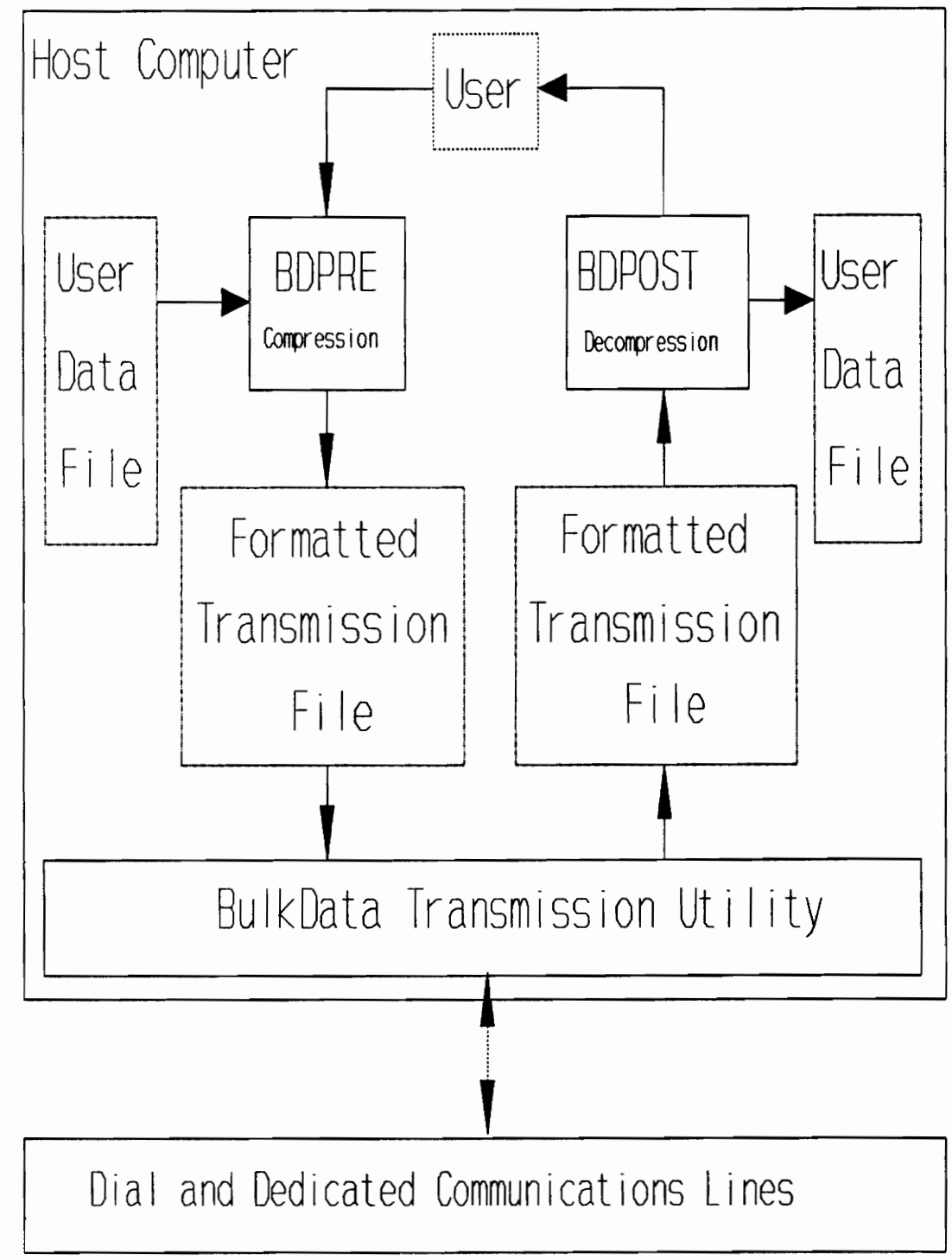

Figure 1. BulkData System Overview

file. When the entire file has been received, BDUTIL causes a job to be queued that will process the incoming data. This job will invoke the BulkData post-processing subsystem, BDPOST.

The BulkData post-processor, BDPOST, reconstructs the original file from compressed segments received by BDUTIL. After the entire file has been reconstructed, the appropriate application responsible for processing this data is invoked.

Data Compression

As used in this project the term data compression should be thought of in the context of the compression of data being transmitted in the data communications environment. Different authors propose different definitions and different ways of categorizing the various methods of data compression. Data compression is the "technique that provides for the transmission of fewer data bits without the loss of information. The receiving location expands the received data bits into the original sequence." [1] Algorithms that intentionally lose some of what they compress support the transmission of digitized images (e.g. FAX) and sounds (e.g. voice).

There are two types of data compression. Most of the older algorithms are based on the frequency of individual characters occurring in the data. That is, they are based on the "information content" of the data. Information content is called the entropy of the original data. This type of algorithm is called character compression.

If we consider a stream of N characters each having a probability of occurrence, P_i, then the entropy [2] or average number of bits needed to represent each character is given by:

$$H = -\sum_{i=1}^{N} (P_i)(\mathrm{Log}_2 P_i) \qquad (1.1)$$

The entropy, H, of "normal English" is about 0.5. [3] Compression algorithms operating on a given set of data will produce a compressed data stream with an average encoded character length of L bits. The efficiency of the algorithm is given by:

$$E = (H/L)*100 \qquad (1.2)$$

Also useful in comparing the various data compression techniques is the term "Compression Ratio" (CR).

$$CR = \frac{\text{Compressed data stream length}}{\text{Original data stream length}} \qquad (1.3)$$

Character compression algorithms take a single byte consisting of eight bits and convert the byte into a variable number of bits based on the frequency of each byte in the original data stream. Most of these algorithms also encode repeated characters (up to a constant limit) with a multiplier code that indicates how many repetitions followed by the encoding of the repeated character.

The above equations are useful for character compression algorithms that operate on each character (or repeating character) based on probability of occurrence. The current algorithms used by BulkData are character compression algorithms. Both are limited in their efficiency, E, as given above.

The first data compression algorithm implemented in BulkData is called C64. The C64 algorithm, as stated above, is a character compression algorithm. C64 is based on the fact that most financial data consists of a subset of the

total character set. The algorithm is able to compress this type of data by about 50 percent.

The second data compression algorithm was implemented for exchanging files with PCs. This character compression algorithm known in the industry as LU1 compression is defined by IBM's System Network Architecture. Like C64, LU1 compresses most financial data by about 50 percent. LU1 compression, however, is not dependant on the selection of a subset of the total character subset and is able to compress data within the limits of data entropy as defined above.

The other major type of data compression algorithm operates on strings (one or more characters) of data and is dependant on the frequency of repeating strings in the data. Most of the current advances in data compression are being made using string compression as the basic approach.

String compression takes a variable number of characters (a string) and converts them to a fixed codeword. One codeword can represent any number of bytes up to the maximum allowable string depth specified in the algorithm. The codeword size is more than eight bits and less than some predetermined maximum also specified in the algorithm.

In January of 1990, a string compression algorithm was selected by CCITT (International Telegraph and Telephone Consultative Committee) as the most effective algorithm for use in a data communications modem environment. This algorithm is defined in CCITT Recommendation V.42*bis* [4].

American National Standards Institute (ANSI) is adapting this algorithm for use with financial applications. A copy of the current ANSI draft standard is included as an appendix to this paper. For reference this draft ANSI document has been labelled ANSI X9.DC [5]. This algorithm compresses financial by 65% to 88% depending on the type of data.

CHAPTER 2 - SCOPE

The scope of this project is the conceptual phase in systems development. Figure 2 shows the whole systems engineering process in the context of this project. In this phase of systems development "the need for a system is defined and the technical basis for an acquisition program is established through feasibility studies. Basic operation and support concepts are identified, and early logistic support planning commences." [6]

The focus of this project is the feasibility of implementing alternative compression methods in BulkData. As shown in the conceptual design block of Figure 2, the operational requirements and maintenance concept are established as a basis for the feasibility analysis and to provide the foundation for the stages of the systems life cycle that follow the conceptual design phase.

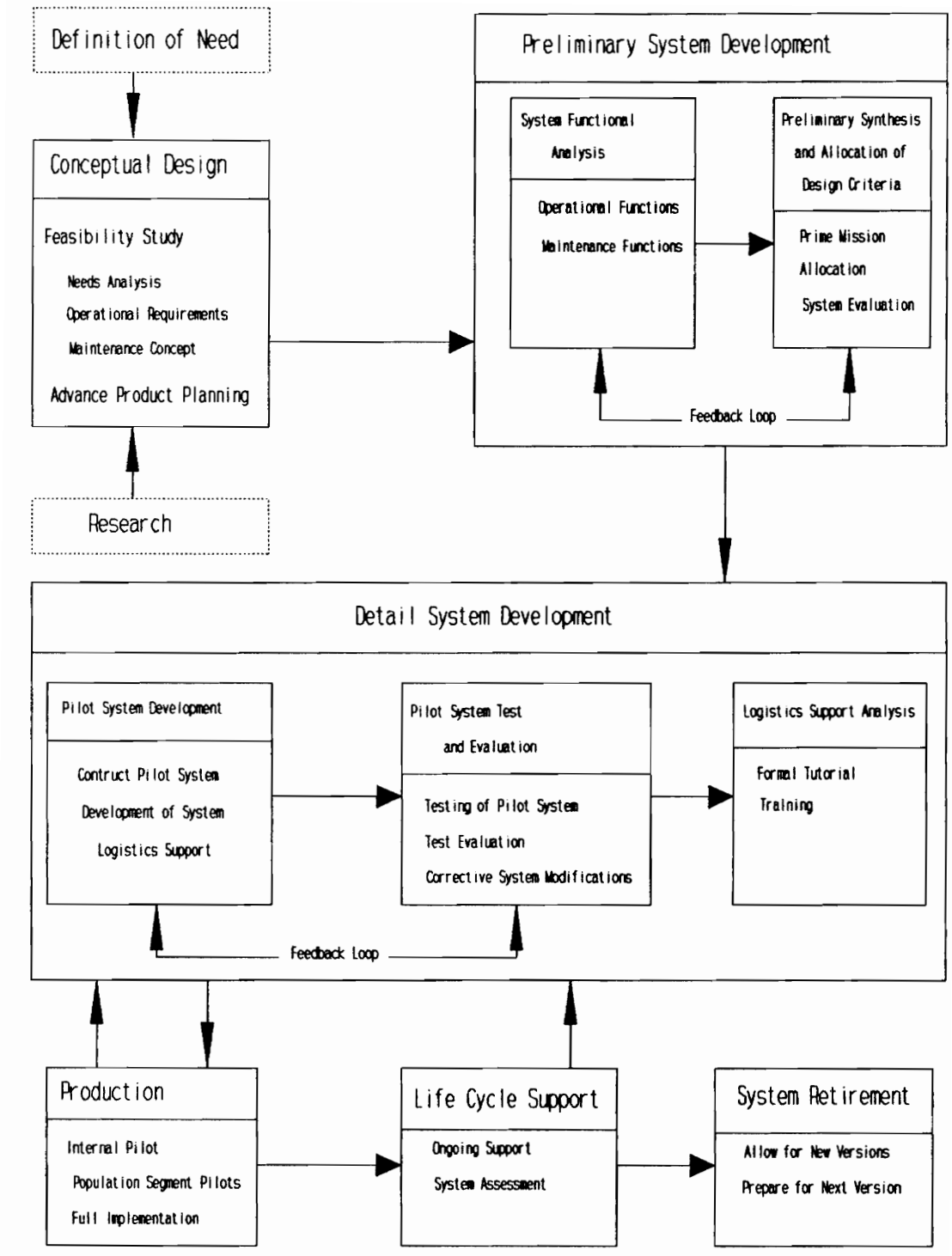

Figure 2. System Life Cycle

CHAPTER 3 - NEEDS ANALYSIS

There are three basic needs that have been identified as problems in the current operational environment:

1) A time critical service has recently been implemented that sends up to four million bytes of data to a single financial institution. This service involves sending MICR (Magnetic Ink Character Recognition) codes for checks that the financial institution will be receiving. The financial institution sends this data to its customers which take advantage of this advance knowledge of these payments.

   This data can not be released by the Federal Reserve until 9:30 A.M. The financial institutions have advertised delivery of this service to their customers by 10:00 A.M. The delivery times and processing times during this half-hour window do not provide adequate margins for errors.

   In some cases the delivery of this data is late, which limits the usefulness of this service to the financial institution and its corporate customer who is the ultimate recipient of this data. This paper will refer to this service as MICR.

2) The PC implementation of BulkData compression uses a different compression algorithm, LU1 compression. This causes operational problems when files must be

rerouted to PCs in contingency situations. These files must be reprocessed for delivery to the PC. Files that were originally compressed using C64 compression can't be sent to Fedline PCs. (All compression on Fedline PCs uses LU1.)

3) The current compression algorithms actually expand many types of data. This is an operational problem that requires files to be sent in compressed mode or uncompressed mode based on the type of data being sent.

CHAPTER 4 - BULKDATA FILE TRANSFER SYSTEM MODEL

General guidelines used in the problem definition and conceptualization phase of model building are:

1. Have a clear purpose for the model.
2. Focus on the problem with the system, rather than the system. [7]

Purpose of the model

The purpose of the model is to aid management in developing policies on a technical level in support of the business requirements satisfied by the BulkData system. Therefore, the model is used as a tool to improve the delivery of electronic services to the end user.

Reduction in file delivery time, minimizing the end users costs, and minimizing the Fed's costs to provide this service are the principle improvements to consider in model development. The project will concentrate on the effects of data compression within the system. This model allows us to

evaluate all benefits (for example reduced line speed requirement) and costs (such as increased CPU utilization and memory allocation) of data compression.

Construction of the model may be outlined as follows:

1. Problem definition. The problems with the system must be clearly defined in the context of relevant business requirements. A set of objectives are developed based on the problems with the current system and the business goals for the organization that apply to the system. Therefore, any relevant (i.e. significant problems that inhibit the goals of the business) problems must be clearly defined.

2. System conceptualization. A mental model of the system is built that clearly relates to the problems being solved. Key variables affecting system operation must be understood and ,if feasible, graphed over time to show the relationships between system variables. These relationships are explored to uncover the "reference behavior modes" [7] of the system. These modes show the patterns of system operation (for example, normal mode versus exception modes or periods of system degradation).

   The construction of the dynamic model involves diagramming these key variables in a causal diagram that focuses on the feedback loop concept. Variables are classified based on their use in equations. The fundamental variables are classified as levels and rates. A third variable known as the auxiliary variable helps in the construction of rate equations and represents information flows in the causal diagram.

   A level variable identifies the accumulation of some quantity over time. Rate variables show how levels change over time. For example, the amount of water in a storage tank is a level variable that varies based on the rates of flow into and out of the tank. The population served by the tank can be viewed as an auxiliary variable that affects the outflow rate.

   "... given a purpose, one should then define the boundary which encloses the smallest possible

number of components. One asks not if a component is merely present in the system. Instead, one asks if the behavior of interest will disappear or be improperly represented if the component is omitted." [8]

3. Model construction. A single, all-inclusive model would be difficult to construct to represent all of the variations required. "Thus, when developing models, the modular approach is preferable." [9]

4. Model evaluation. Output from the model should be able to reconstruct the reference modes shown in graphs of the system variables.

Problem with the System

If data can't be delivered within the constraints of the application, a single alternative is usually considered by management - increasing the aggregate line speed. In the national network, six additional 56 KBPS lines were added to deliver data needed for contingency operations. Volume increased on the national network in the last fifteen months from 1.6 billion to 3.6 billion characters per day.

Given the expense of increasing line speeds and adding additional lines, other alternatives should be explored. This model should provide a basis for exploring other alternatives based on a more complete understanding of the factors that contribute to file delivery time.

Model Conceptualization

Figure 3 shows a causal diagram used in construction of the models. The diagram illustrates all level and rate variable relationships. Key auxiliary variables are also diagrammed. Table I lists key variables and their meanings.

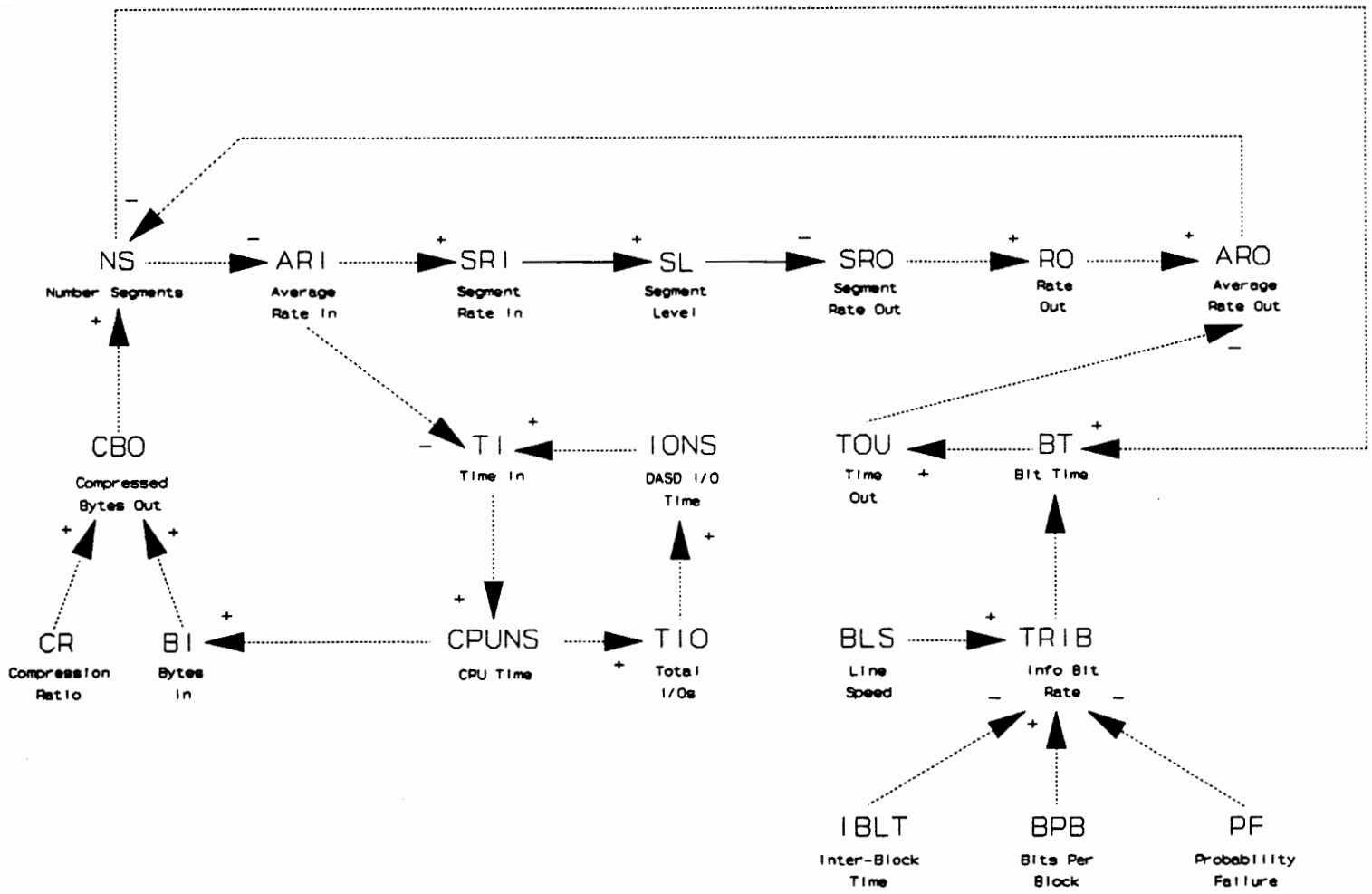

Figure 3. Causal Diagram for BulkData

Table I: List of Key Variables and Constants

LABEL	DESCRIPTION	TYPE/"UNITS"	
SL	The number or level of segments on queue, ready for delivery.	L	Segments
SRI	The rate of segments placed on queue for BulkData to deliver.	R	Seg/Sec
SRO	The rate of segments transmitted by BulkData.	R	Seg/Sec
CPUNS	CPU time for overhead and compression while preprocessing a file.	A	Sec
TI	Time spent preprocessing a file prior to transmission including the overhead and compression CPU time, and I/O (Input/Output) wait time.	A	Sec
TOU	Time spent transmitting a file.	A	Sec
TRIB	The Transmission Rate of Information Bits (TRIB). A measure of the effective rate of information transfer. [1]	A	Bits/Sec
BI	Total number of bytes in a particular uncompressed file to be transmitted	C	Bytes
BLS	The line speed in bits per second.	C	Bits/Sec
CR	The compression ratio. The ratio of the number of output bytes to the number of input bytes.	C	none
IBLT	Time spent between blocks on a transmission line waiting for a response, etc.	C	Sec
PF	Probability that a block of data will have to be resent due to a line error.	C	none

Model Construction

Separate models were constructed for the current environment with C64 compression and an alternative environment with ANSI X9.DC data compression. For these two scenarios, separate models were built to handle the differences between dedicated and switched data communications lines. The four DYNAMO models are listed in Appendix B.

Model Evaluation

The models of the current system were validated for each line type based on available statistics. The accuracy of the models were within a few percent for dedicated lines, but switched lines have a larger variation in connect time based on many non-deterministic factors. However, the model of switched line operation was within five or six percent of actual system operation for most transmissions and all deviations from projections made by the model were understandable in the context of the particular transmission (i.e. size of file, length of telephone call, etc.). Table II provides a characteristic sampling of actual transmission times and the transmission times projected by the model.

Table II: Model verification

COMPARISON OF ACTUAL AND PROJECTED TRANSMISSION TIMES					
Line type	**Line speed (bits per second)**	**Bytes Sent**	**Actual time in seconds**	**Model time in seconds**	**Absolute error (%)**
Dedicated	56,000	922,289	206	195	5.34
Dedicated	56,000	476,447	101	102	0.99
Dedicated	19,200	1,563,847	767	764	0.39
Dedicated	19,200	546,728	269	268	0.37
Dedicated	9,600	246,026	236	225	4.66
Dedicated	9,600	5,613,644	5,512	5,109	7.31
Sample average error for dedicated lines					**3.18**
Switched	9,600	197,694	492	521	5.89
Switched	9,600	95,147	255	266	4.31
Switched	2,400	27,996	169	175	3.55
Switched	2,400	8,266	117	73	37.61
Sample average error for switched lines					**12.84**

System Model Output

The following graphs from the model show the number of segments per second that are preprocessed and transmitted. The number of segments in the queue is also shown. The top legend of each graph provides the vertical scales for segment rate in, segment rate out, and segment queue level as follows:

[line of short dashes] Segment Rate In (Y1,Y2)
[line of long dashes] Segment Rate Out (Y1,Y2)
[solid line] Segment Queue Level (Y1,Y2)

where Y1 is the minimum vertical axis value and Y2 is the maximum vertical axis value for the particular quantity being graphed. The horizontal axis shows the time in seconds.

Figure 4 was generated by a model of the current system. The figure shows the level of segments in the BulkData system based on current input and output rates for a million character input file that was compressed to 0.55 of its original size by the current C64 compression algorithm. The figure represents a typical ACH transmission over a 56 KBPS dedicated line. Figure 5 shows the delivery of the same file over a 19.2 KBPS dedicated communications line.

The current system model was then modified to show the delivery of the same file with ANSI X9.DC compression used to

compress the same file to .347 of its original length. Figure 6 and Figure 7 show the delivery of these files over 56 KBPS and 19.2 KBPS dedicated communications lines, respectively.

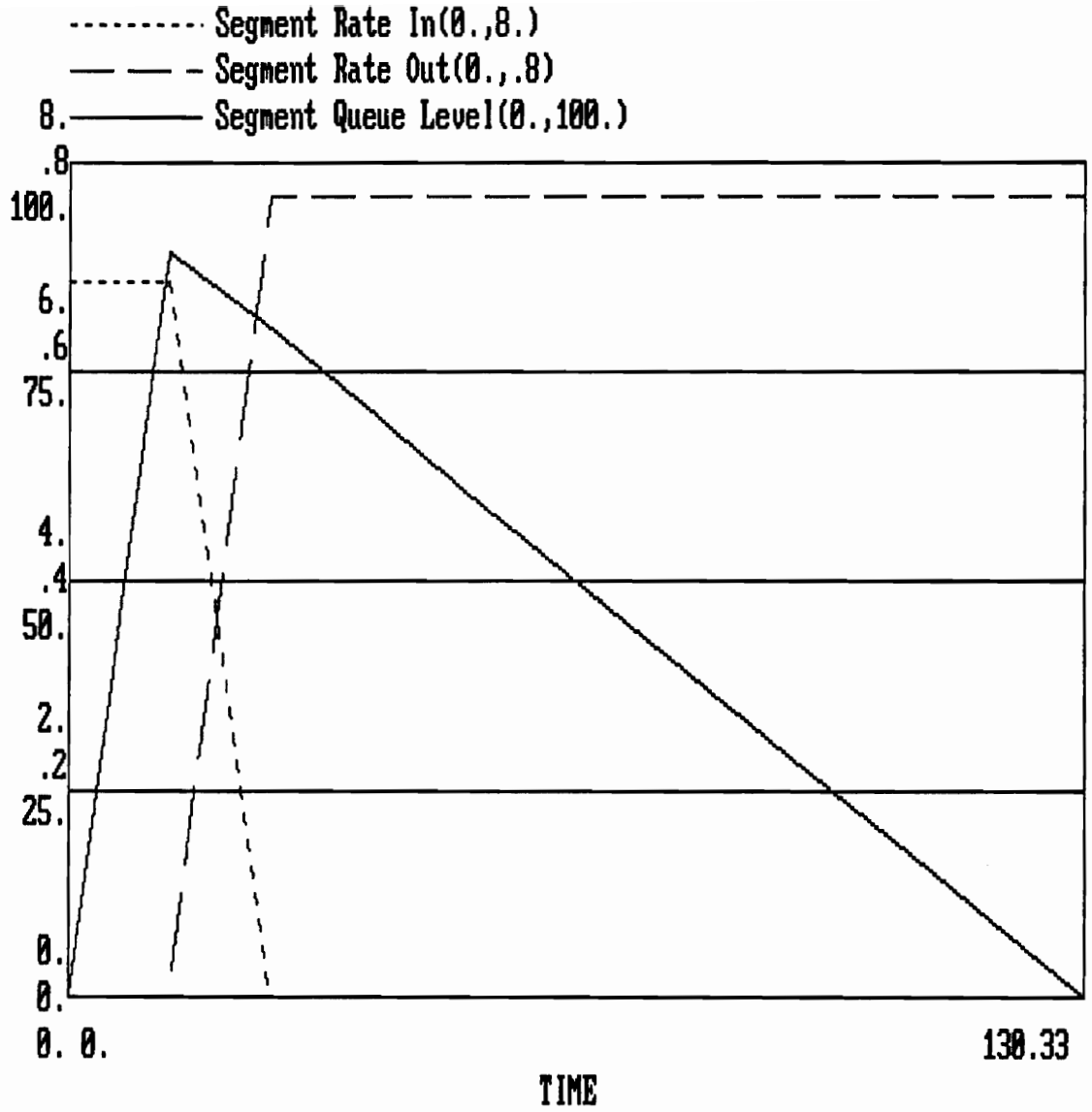

Figure 4. Current BulkData System: 56 KBPS Line

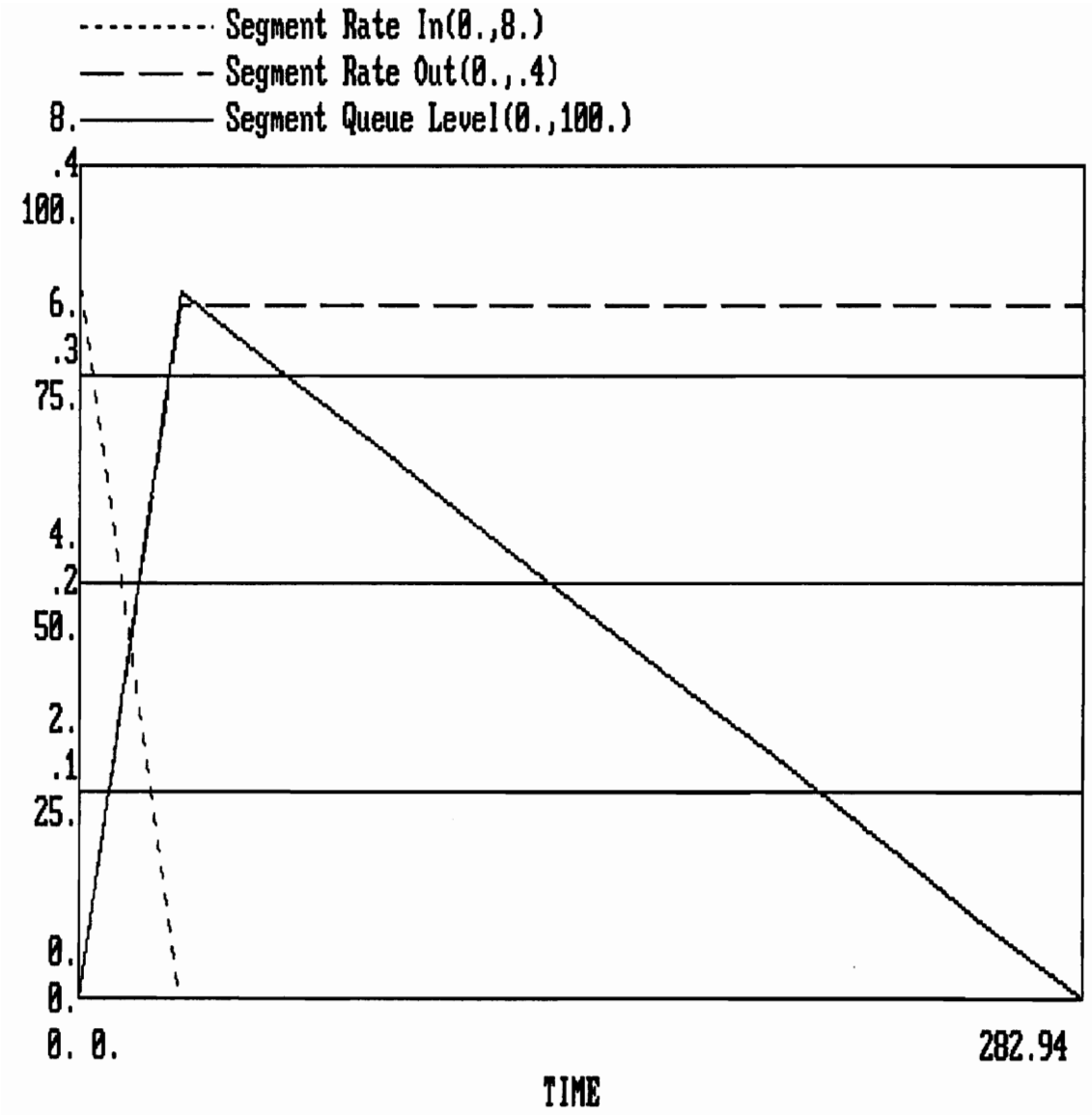

Figure 5. Current BulkData System: 19.2 KBPS Line

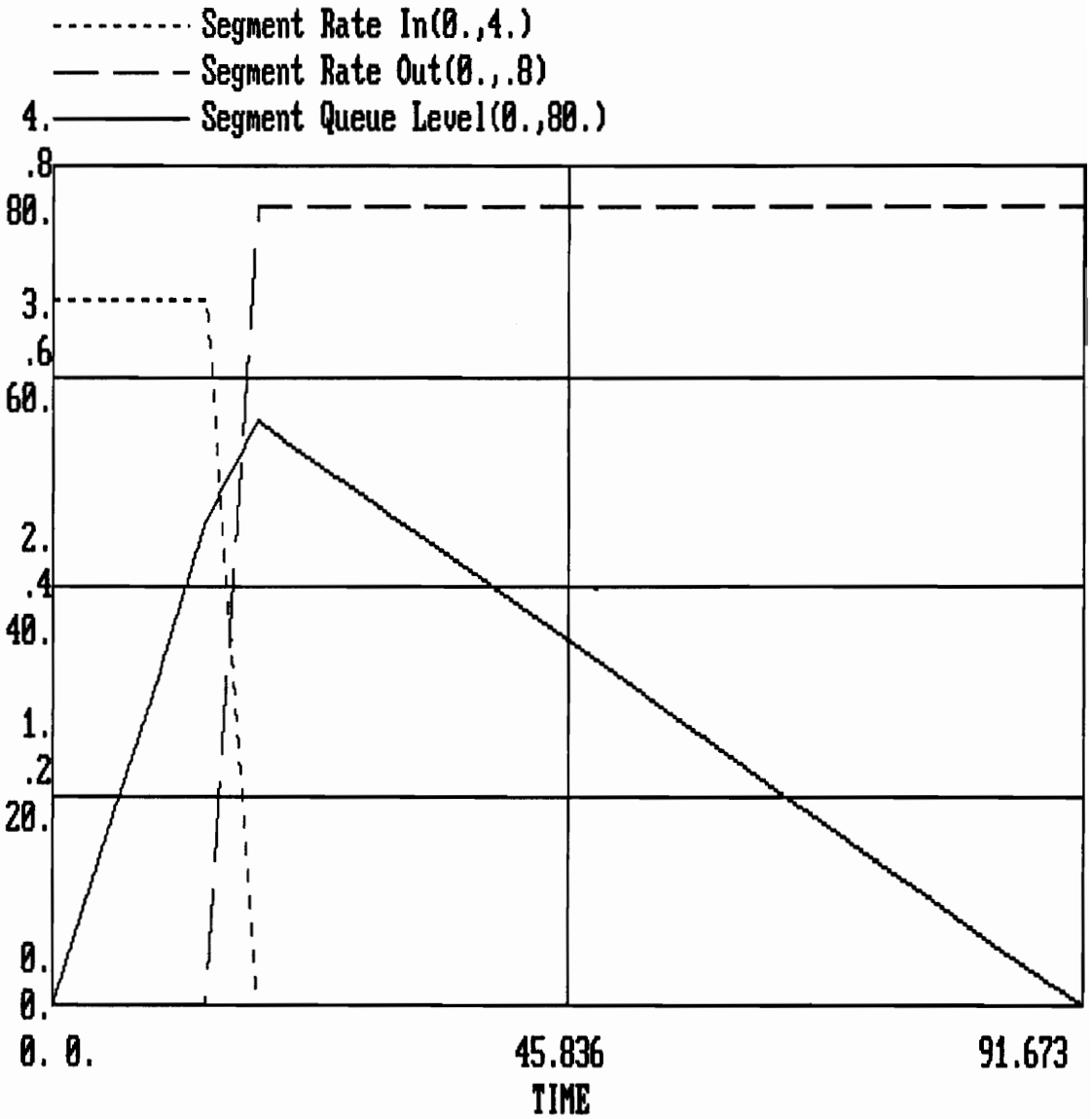

Figure 6. BulkData with X9.DC: 56 KBPS Line

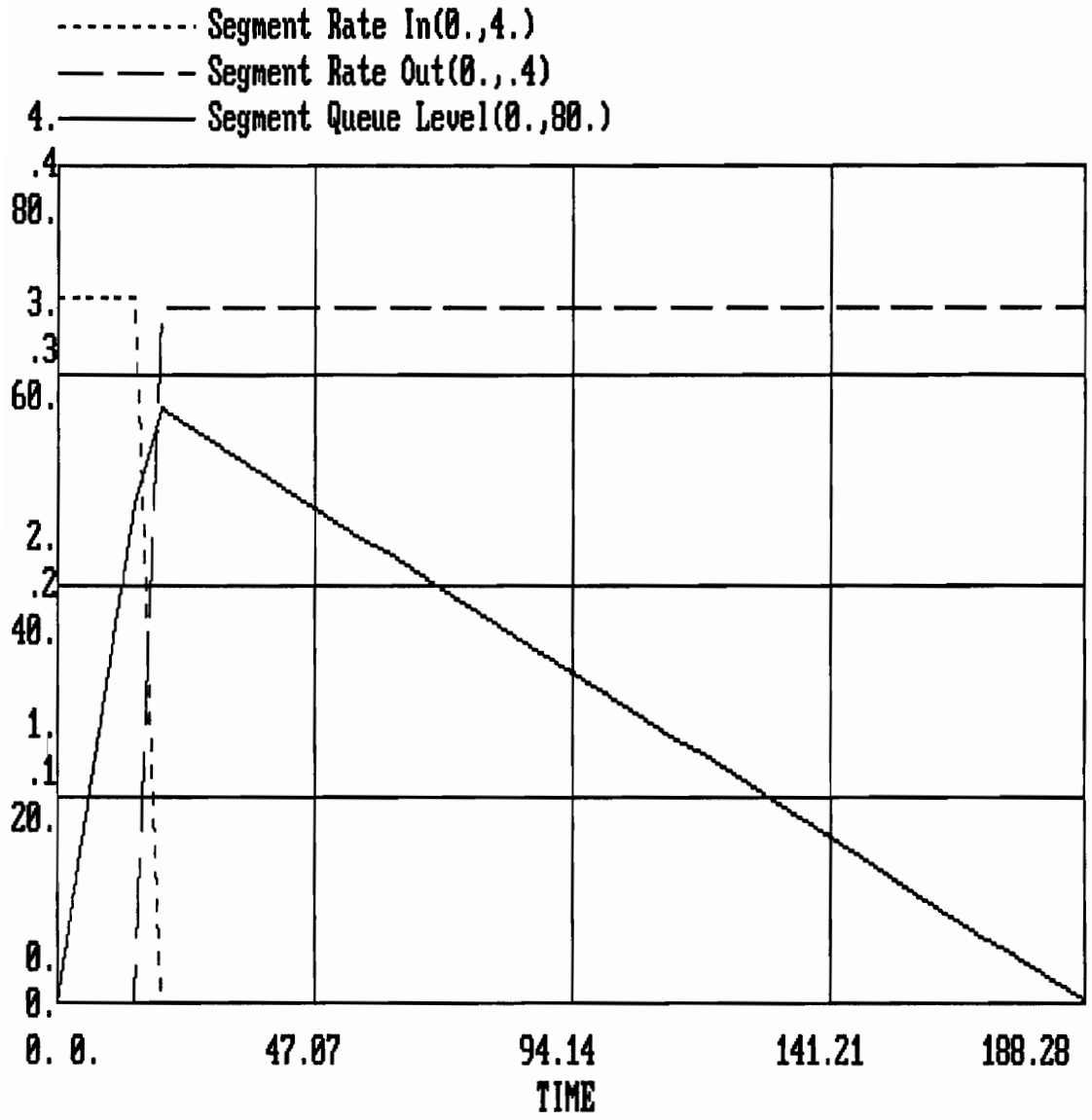

Figure 7. BulkData with X9.DC: 19.2 KBPS

Making some minor modifications to the base DYNAMO model, the current system can be compared with the proposed ANSI compression scheme for both 2400 BPS and 9600 BPS dial-lines. Figure 8 and Figure 9 show the current delivery times for 2400 BPS and 9600 BPS dial-lines. Figure 10 and Figure 11 project delivery times with ANSI X9.DC.

Also of interest is the effect of the ANSI X9.DC compression algorithm on the delivery of the time critical MICR service. Users of this time critical service tend to be high volume users who already have installed a high speed line. Figure 12 and Figure 13 show the actual and projected delivery times for the MICR service. Tests using BDCOMPA show that the compression ratio for MICR files is 0.178. The C64 algorithm compression ratio for MICR data is 0.481.

Table III on page 39 summarizes the modeling results. The table compares delivery times of typical length uncompressed files using current algorithms and the proposed ANSI compression algorithm. First the delivery time using the current method of compression is given followed by the delivery times projected by the model for X9.DC. The percent improvement in delivery time is provided for each transmission.

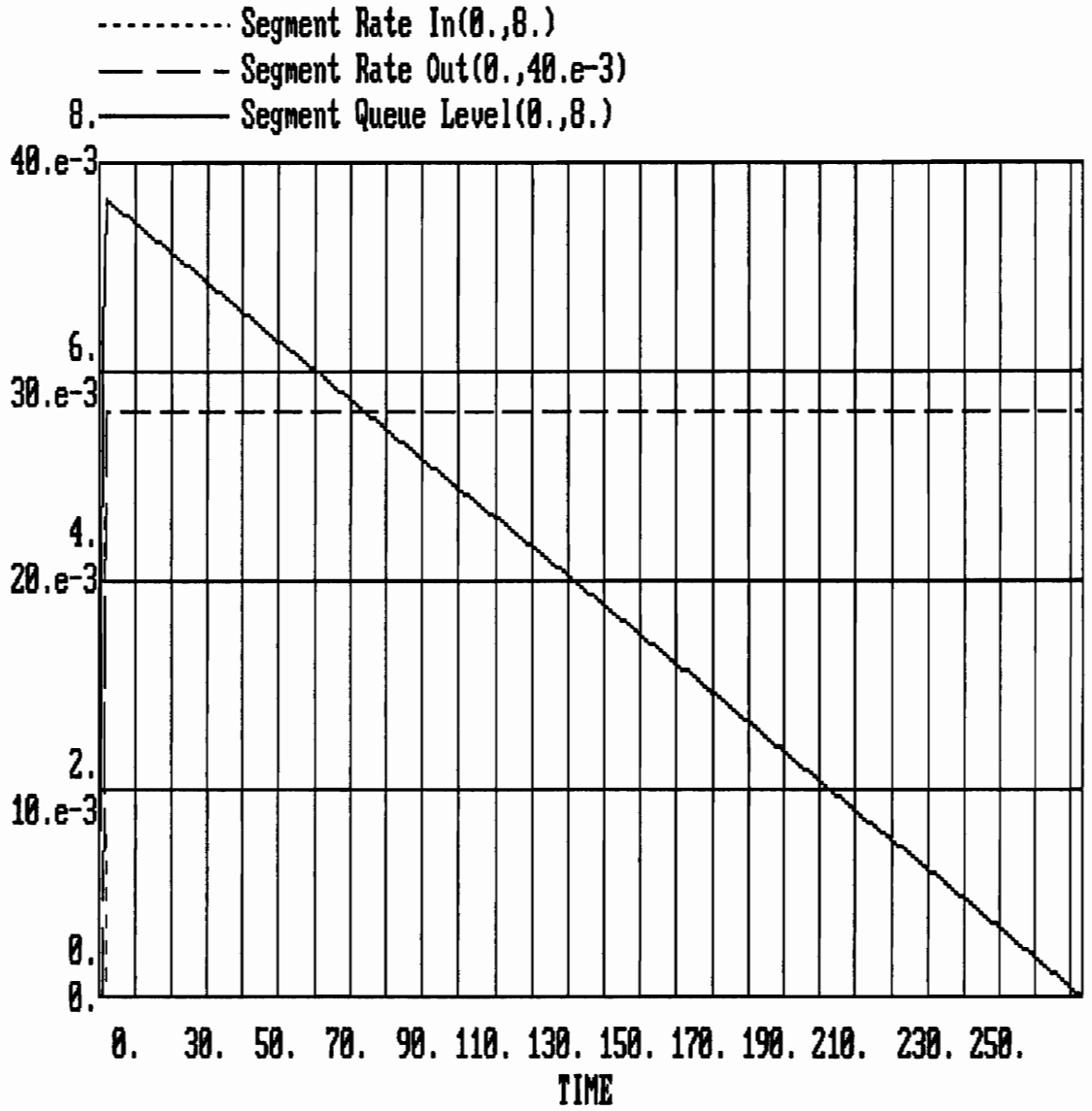

Figure 8. Current System: 2400 BPS Dial

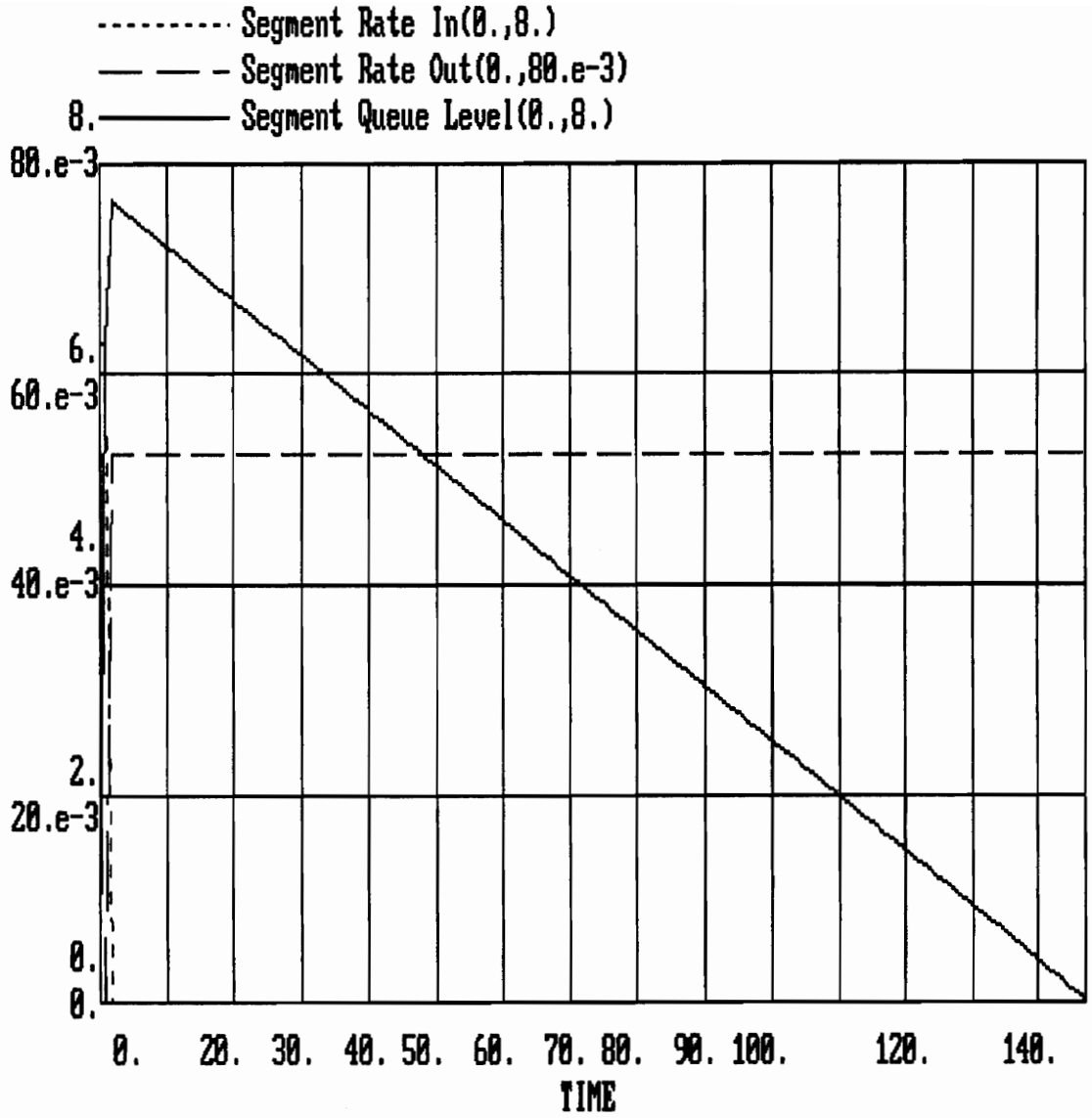

Figure 9. Current System: 9600 BPS Dial

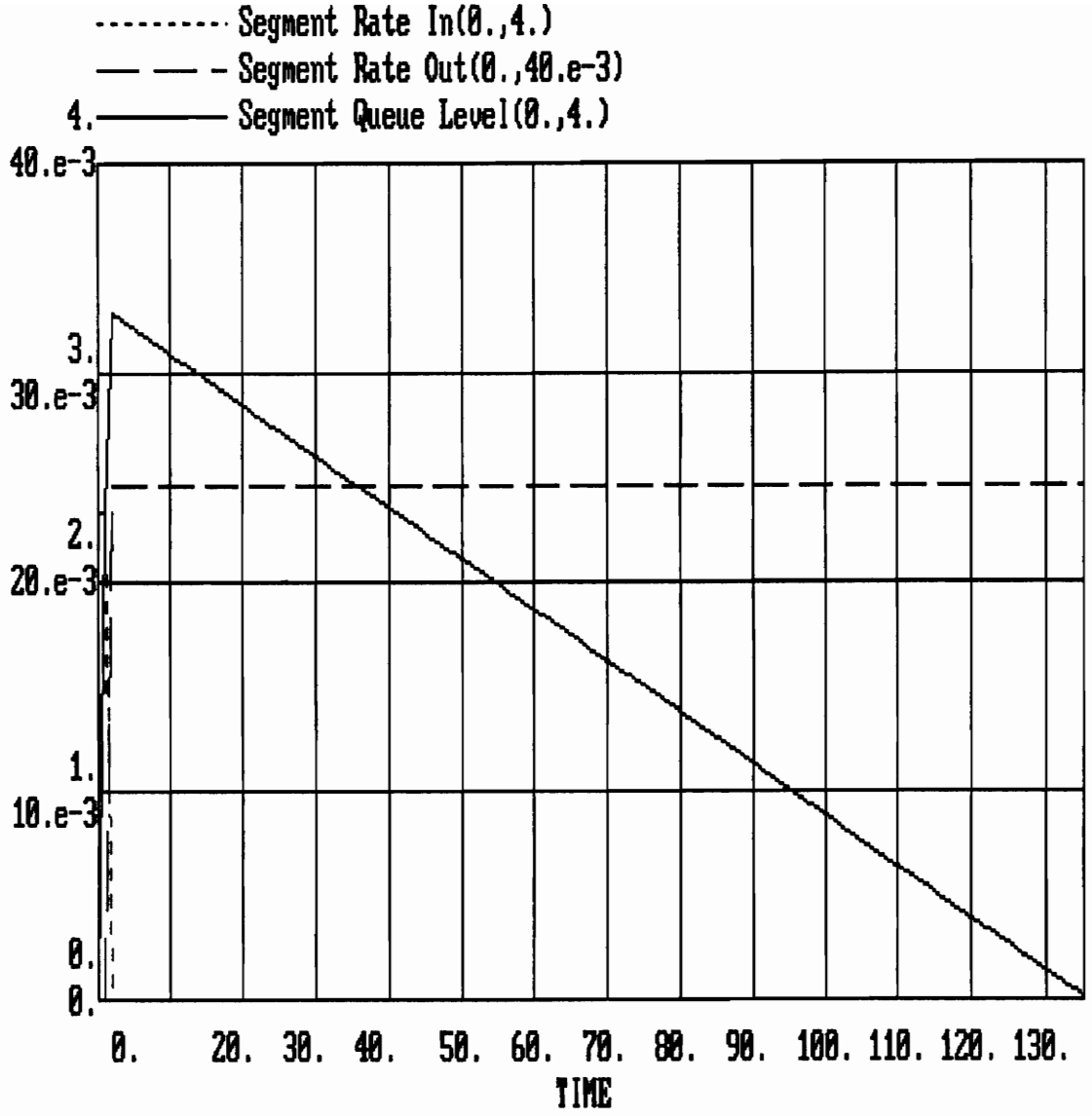

Figure 10. System with X9.DC: 2400 BPS Dial

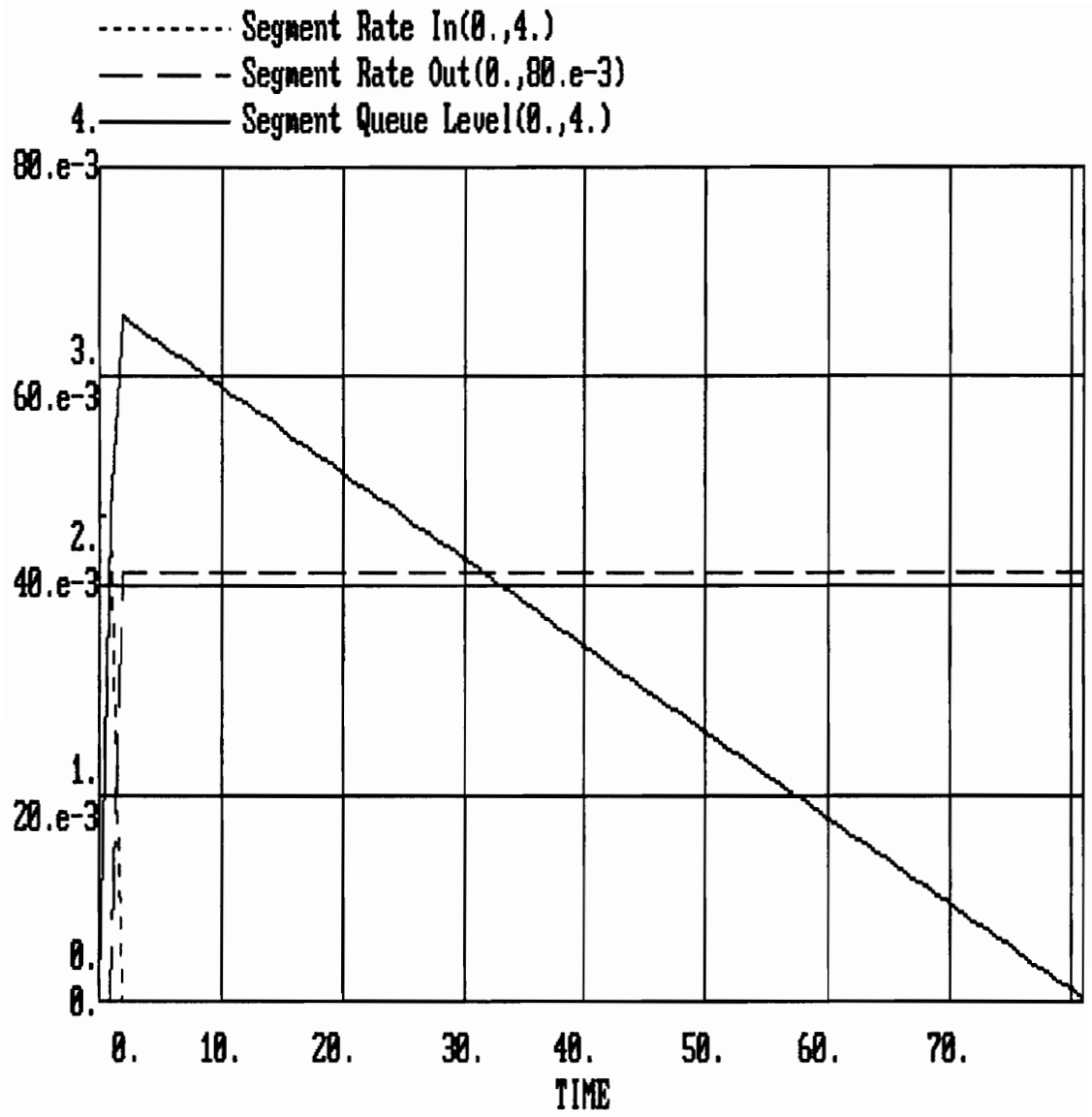

Figure 11. System with X9.DC: 9600 BPS Dial

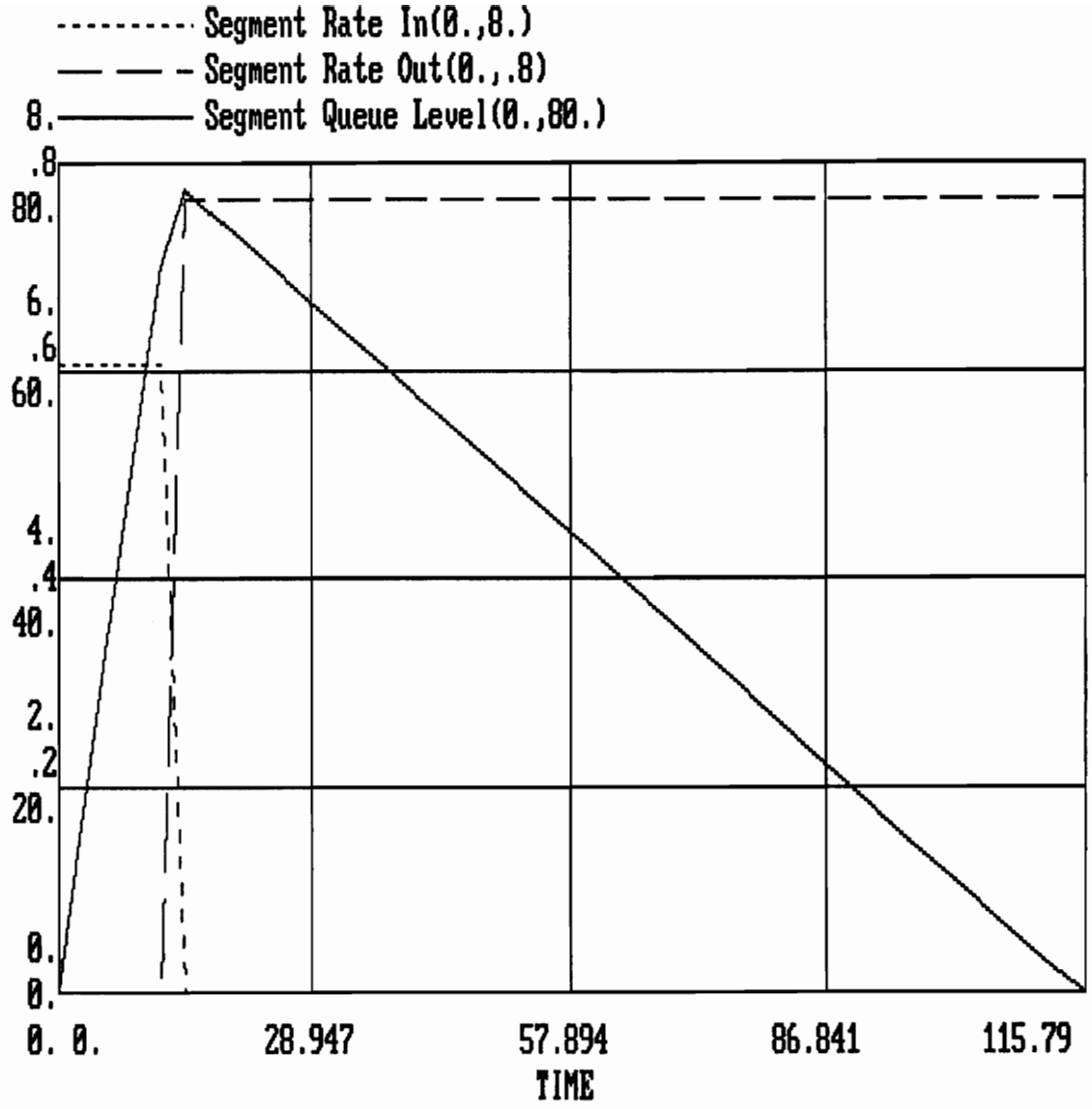

Figure 12. Current MICR: 56 KBPS

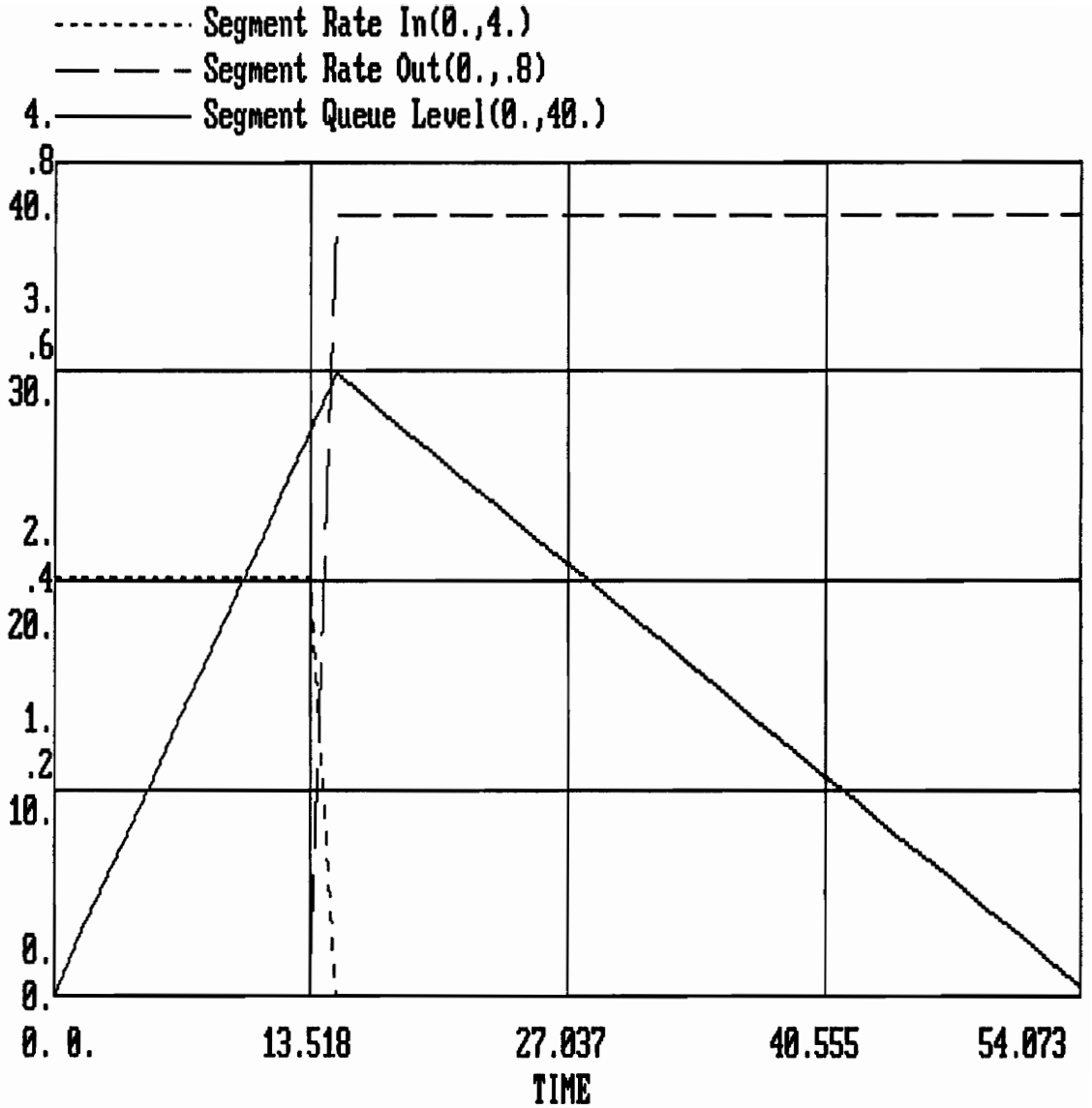

Figure 13. X9.DC MICR: 56 KBPS

Table III: Effect of compression on delivery time

RATIO	METHOD	DATA	BDPRE TIME	UNCOMPRESSED CHARACTERS	LINE TIME	TOTAL TIME	LINE TYPE	LINE SPEED KBPS
.55	C64	ACH	13.129	1,000,000	269.81	282.94	Dedicated	19200
.347	X9.DC	ACH	16.929	1,000,000	171.35	188.28	Dedicated	19200
Percent improvement in total delivery time								**33.46**
.55	C64	ACH	13.129	1,000,000	117.20	130.33	Dedicated	56000
.347	X9.DC	ACH	16.929	1,000,000	74.744	91.67	Dedicated	56000
Percent improvement in total delivery time								**29.66**
.81	LU1	ACH	1.2194	58,000	272.75	273.97	Switched	2400
.347	X9.DC	ACH	1.4152	58,000	134.54	135.96	Switched	2400
Percent improvement in total delivery time								**50.37**
.81	LU1	ACH	1.2194	58,000	146.56	147.78	Switched	9600
.347	X9.DC	ACH	1.4152	58,000	80.196	81.61	Switched	9600
Percent improvement in total delivery time								**44.78**
.481	C64	MICR	13.017	1,000,000	102.77	115.79	Dedicated	56000
.178	X9.DC	MICR	14.676	1,000,000	39.397	54.07	Dedicated	56000
Percent improvement in total delivery time								**53.30**

CHAPTER 5 - EVALUATION OF ALTERNATIVES

System Operational Requirements

The following requirements are based on the assumption that the current algorithms must be supported for some interim period. These requirements are used in establishing evaluation criteria.

1. System availability must be maintained at 99.95%.
2. The algorithm must not expand any data type, including binary data, data that is already compressed, or encrypted data.
3. CPU usage must be constrained to a reasonable increase.
4. Any alternative must decrease the average transmission time by 25%.
5. Memory allocation is constrained by the PC DOS environment to less than 30,000 additional bytes.
6. A phased implementation is required to limit operational risks and logistics support complexity.

7. A flexible path for migration to future algorithms is required.

System Maintenance Concept

If the proposed ANSI standard data compression is selected for use in BulkData, three development sites will be involved in making this change. As an aid to initial development as well as any needed maintenance (error corrections, parameter adjustments, etc.), an extensive tutorial guide on all aspects of the algorithm should be developed if the ANSI X9.DC algorithm is adopted.

CPU Usage Evaluation

In order to evaluate the use of proposed standard ANSI X9.DC, a compression program, BDCOMPA, was written and tested in the BulkData system. A listing of this program is included as Appendix C. The equations that follow are based on testing with BDCOMPA and the existing compression routines in BulkData on an IBM 3090-400E.

Based on tests of BDCOMPA, a formula can be derived to predict CPU usage by the ANSI X9.DC compression algorithm. Use of CPU resource by BDPRE using this algorithm can be roughly estimated by computing the time spent in initialization processing, processing the input data, and the

time spent creating the output codes. Therefore, the CPU usage for the new algorithm is

$$CPUA = A1*IB + A2*OB + C1 \quad (5.1)$$

where A1, A2, and C1 are constants that depend on the relative power of the computer's processor, IB is the number of input bytes, and OB is the number of output bytes. Testing with a variety of data types and BDCOMPA provided several sets of three equations in three unknowns (i.e. A1, A2, C1). Solving these equations shows that A1 is approximately 0.0000012, A2 is 0.0000117, and C1 is 0.22. CPUA denotes use of CPU by the ANSI algorithm.

Based on tests with either the C64 or LU1 compression algorithms, the amount of CPU used by these algorithms is approximately

$$CPUC = C2*IB + C1 \quad (5.2)$$

where, as above, C2 and C1 are constants dependant on the relative power of the computer's processor. Testing with a variety of data types and the existing compression routines provided several sets of two equations in two unknowns (i.e.

C2 and C1). Solving these equations shows that C2 is approximately 0.00000113 and C1 is 0.22 as above.

Cost Allocation Comparison

BulkData costs are allocated to the user departments within the Fifth District based on CPU use and characters sent. The current rates are $0.0709 for 1,000 characters transmitted and $41.8860 for a CPU minute.

The CPU time needed for post processing which includes the CPU time to decompress using either X9.DC or the existing decompression routines is approximately the same as the corresponding preprocessing step. Assuming an equal distribution of received data, the CPU costs in Table IV are doubled. This is done to account for the additional overhead of post processing these files by BDPOST, the BulkData post processing application.

Table IV: BulkData Transmissions - Full Day's Traffic Comparison

DATA TYPE	NUMBER OF FILES	UNCOMPRESSED CHARACTERS	CURRENT COMPRESSED CHARACTERS	X9.DC COMPRESSED CHARACTERS	NETWORK COST SAVINGS	INCREMENTAL CPU COST IMPACT
ACHP	414	115,378,420	73,306,276	40,036,312	$2,358.84	$1,299.79
LOCP	24	40,064,680	23,467,803	13,902,444	$678.18	$418.88
CRZP	147	15,110,500	9,477,907	5,243,344	$300.23	$168.22
BUDD	13	17,169,668	5,221,080	2,060,360	$224.10	$100.51
XYMI	18	2,956,864	2,645,215	526,322	$150.23	$45.83
ALL OTHERS	201	11,774,895	5,696,554	4,085,889	$114.20	$103.50
TOTAL	**817**	**202,455,027**	**119,814,835**	**66,136,671**	**$3,805.78**	**$2,136.73**

Dial Line Costs

For the Fifth District, all costs associated with dial lines are based on actual minutes used for each line. The current cost per minute is $0.08. Given the distribution of line usage for 9600 and 2400 dial lines, the overall percentage cost reduction in line time charges for both incoming and outgoing files is approximately 50 percent. However, the use of dial lines for sending BulkData traffic is only a small portion of the total use of dial lines. The average direct monthly savings in dial line usage fees would be approximately $1,000.00.

Relative Line Costs

The costs of leased lines is based mainly on distance and line speed. The following discussion begins with the selection of a hypothetical 19,200 BPS line costing $19,200.00 per month. As an approximation lines can be grouped into three price ranges:

1) Speed One lines are 2400 BPS to 19,200 BPS. These lines use the same physical circuits with perhaps different line conditioning. Selecting a hypothetical 19,200 BPS line that costs $19,200.00 per month, then each bit-per-second of throughput capability costs $1.00.

2) Speed Two lines are 56,000 BPS. These lines cost roughly twice as much as Speed One lines. Upgrading the 19,200 BPS line to 56,000 BPS would raise the monthly cost to about $38,400.00. Then each bit-per-second of throughput capability costs $0.69.

3) Speed Three line are 1,544,000 BPS. These lines (known as T1 circuits) cost roughly nine times the cost of Speed One lines. Upgrading the 19,200 BPS line to 1,544,000 BPS would raise the monthly cost to about $172,800.00. Then each bit-per-second of throughput capability costs $0.11.

X9.DC Implementation Costs

The cost for development and integration of X9.DC into all Federal Reserve PC and mainframe platforms (UNISYS and IBM) is approximately six man months or roughly $50,000.00. This is based upon an allocation of two man months for each development team. There are three vendors that hold patents on the X9.DC algorithm. The total cost to obtain the three required licenses is $60,000.00. Therefore, the total implementation cost is estimated at $110,000.00.

Criteria for Decision Making

Alternative solutions to the stated needs should be weighed based on the following factors. Weights from one (1) to three (3) signify the importance of each category (weight 3 being the most important designation).

1) Initial Costs includes software development, license fees, modems, communications controller upgrades, and communication management equipment. This criteria is given a weight of two (2) based on the fact that this effort is an upgrade to an existing system rather than a total system replacement.

2) Operating Costs includes software maintenance, CPU utilization, network utilization, technical support

costs, and line costs. This criteria is given a weight of three (3) based on the importance of maintaining operational costs at levels that allow competitive unit costs of electronic services.

3) Operational Complexity accounts for the ease of use and simplicity of procedures for operating the system. This criteria rates a weight of one (1) based on the need for reductions in overall system complexity coupled with an historically well-trained operational staff.

4) Contingency Costs includes the costs of establishing and maintaining backup facilities consistent with current business requirements for contingency operations. The brunt of contingency costs are monthly costs such as backup line costs and are therefore given a weight of three (3).

5) Reliability and availability measures the overall impact alternatives have on achieving this requirement. Based on an analysis of the desires of the users of our electronic services, this criteria must be rated at three (3).

Context for Decision Making

There are three possible contexts that must be applied to the decision making process. Context C1 involves making a decision faced with an immediate operational need that requires significant bandwidth increases not achievable through the implementation of the X9.DC algorithm. Context C2 is a situation of no need for increased bandwidth. Context C3 requires an increase in communications bandwidth that is achievable by either implementing the X9.DC algorithm or increasing line speeds.

Contextual Decision Model

Table V shows a qualitative decision model comparing the alternatives in each context as stated above.

Table V: Contextual Decision Model

	Initial Cost Economy			Operating Costs Economy			Operational Simplicity			Contingency Cost Economy			System Reliability and Availability			Total for each Context		
Context (*)	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
Weights	2	2	2	3	3	3	1	1	1	3	3	3	3	3	3			
Alternative (**)																		
1) Do Nothing	U	3	U	U	1	U	U	1	U	U	2	U	U	3	U	U	25	U
2) Line Upgrades	3	2	3	2	1	1	1	1	1	2	1	1	3	1	3	28	14	22
3) X9.DC	U	1	2	U	2	3	U	2	3	U	2	3	U	1	1	U	19	28
4) 2 & 3 Above	1	1	1	3	1	2	3	1	2	3	1	2	1	1	1	26	12	19

* Explanation of Contexts:

- C1) Immediate need for bandwidth greater than can be achieved with the compression improvements.
- C2) There is no immediate need for additional bandwidth based on the volume of traffic currently on the network.
- C3) There is an immediate need for additional network bandwidth that can be satisfied by either implementing improved compression methods or increasing line speeds.

** Explanation of Grading:

- 1) A low level of achievement or objective satisfaction.
- 2) A moderately successful level of achievement or objective satisfaction.
- 3) A highly successful level of achievement or objective satisfaction.
- U) This alternative is unacceptable in the particular solution context.

Table V suggests a slight preference for upgrading lines over implementing both alternatives if the problem context is perceived as the situation where the implementation of ANSI X9.DC alone would be insufficient. However, after this upgrade is made the situation or problem context will eventually degrade to a context favoring the implementation of ANSI X9.DC because of normal traffic growth. Therefore, the decision model shows a fairly strong preference for the eventual adoption of the proposed data compression method based on operating cost savings, a reduction in contingency costs, and operational simplicity.

CHAPTER 6 - OPTIMIZATION MODEL

ANSI X9.DC Algorithm Tutorial

In order to understand the methods used to optimize the ANSI X9.DC algorithm, some of the details of operation of the algorithm should be understood.

The ANSI X9.DC data compression algorithm operates on strings of data and is dependant on the frequency of repeating strings in the data. String compression takes a variable number of characters (a string) and converts them to a fixed codeword. One codeword can represent any number of bytes up to the maximum allowable string depth, String_Max, specified in the algorithm. The codeword size is initially nine bits and is incremented to some maximum, Bits_Max, which must be less than or equal to fifteen bits.

A dictionary is dynamically created and updated based on the frequency of strings of characters in the data to be compressed. [10] The dictionary typically is stored in a tree data base structure for the efficient storage of a large number of strings. Each node of the tree structure represents a single codeword and a single character. The path from the current node back to the root yields the string of characters that the current node represents in reverse order.

Most systems would be configured for eight-bit characters, which requires 256 separate trees in the dictionary. The dictionary is initialized to have 256 root nodes with values ranging from zero to 255. As strings of recurring data are found, nodes are added to these trees. Code_Number is the maximum number of nodes that can be added to these trees.

Test Methodology

In selecting the appropriate parameters, ACH data was used to baseline the tests. Other types of data were used to verify that there was no significant deviation from this established baseline. This method was used because ACH data represents the majority of all data types transmitted by BulkData currently.

ANSI X9.DC Parameter Optimization

Figure 14 shows the results of tests using different codeword sizes. Figure 14 demonstrates that for the typical ACH transmission type data, increasing the Bits_Max parameter beyond eleven (11) bits does not yield a substantial gain in compression ratio efficiency.

Two factors to be considered when increasing the Bits_Max parameter are memory allocation and CPU time usage. Nine bytes are required to represents each node in the tree structure. One byte for the character stored at the node, and four (4) "pointers" or indices that connect the node to the other nodes in the tree. Each pointer consists of two (2) bytes.

Figure 14 shows that tree structure memory allocation is 9,216, 18,432, 36,864, 73,728, 147,456 and 294,912 for ten (10) bits through fifteen (15) bits respectively for the codeword size. Given that this algorithm needs to work in the PC environment, memory utilization needs to be contained to an appropriate level for the PC DOS compatible machine.

Another factor to be considered is that there is an increase in the percent of CPU used as the maximum codeword size is increased. Therefore the optimal codeword size for

most of the data types transmitted using BulkData is eleven (11) bits.

Figure 15 shows that the optimum string depth is thirty-two (32) bytes. This result was obtained using a codeword size of eleven (11) bits.

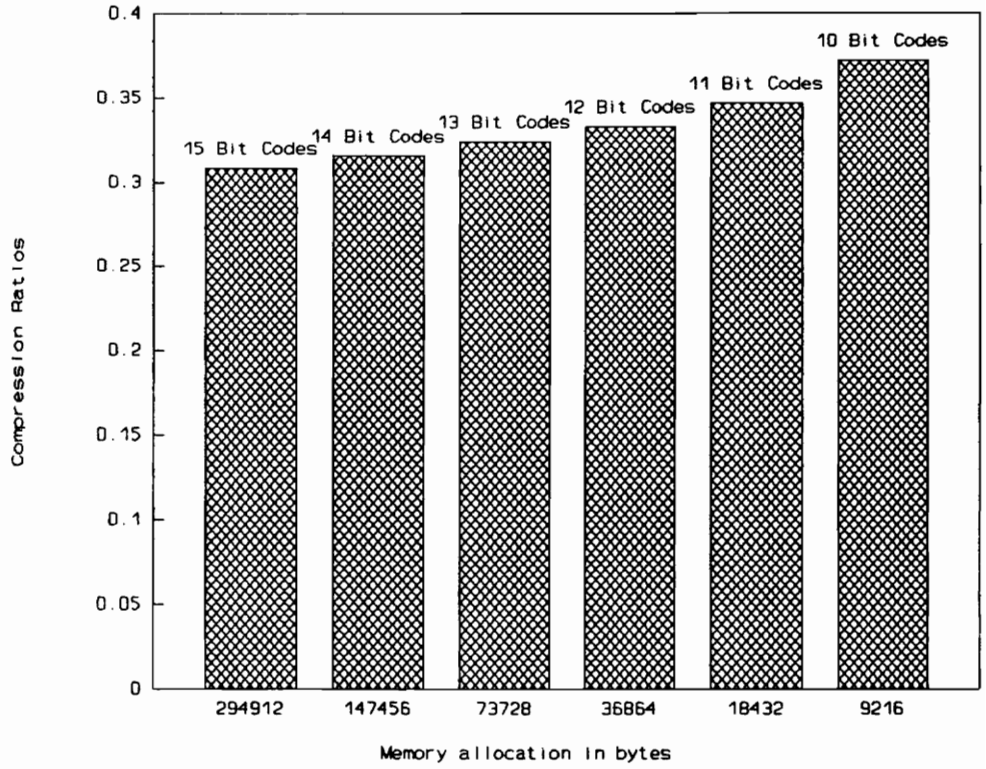

Figure 14. Code size and Compression Ratio

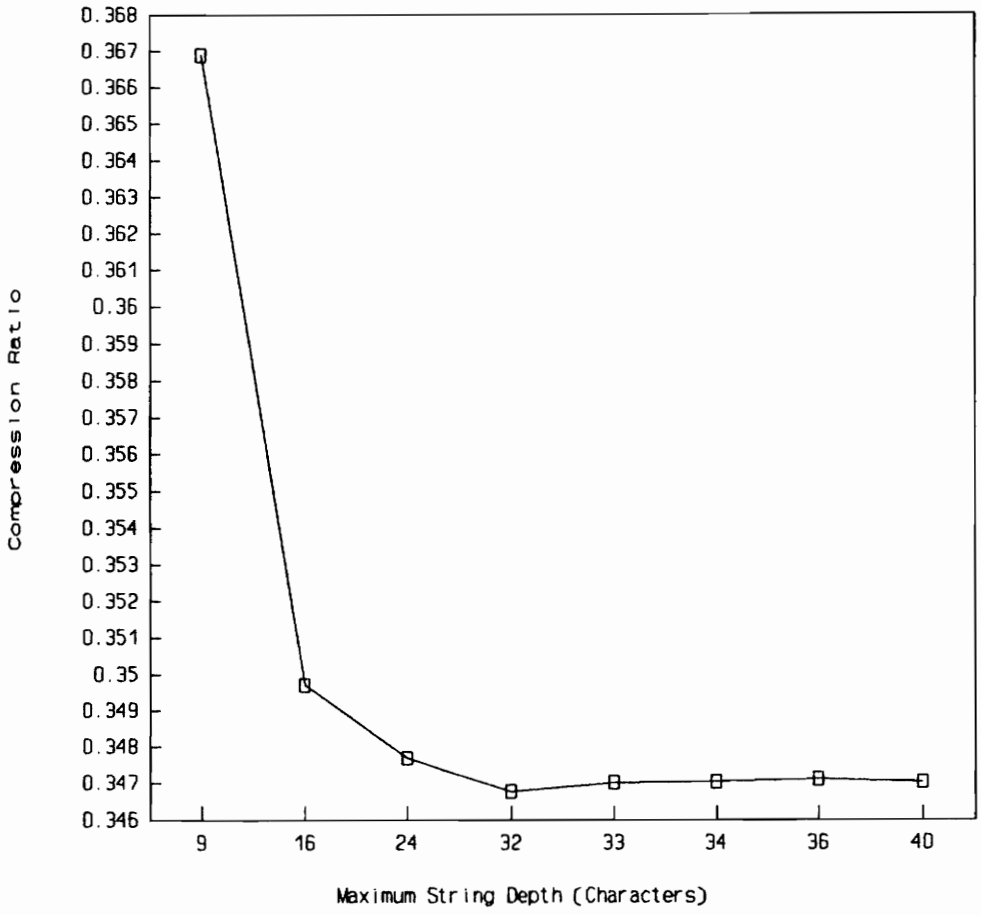

Figure 15. String Depth and Compression Ratio

CHAPTER 7 - MANAGEMENT RECOMMENDATIONS

Conclusions

The results of this modelling process have clearly demonstrated several advantages in employing the ANSI X9.DC algorithm. The most important advantages are simplicity of operation, throughput improvements for time critical services, and operational cost containment.

Maintenance Approach

This algorithm is very complex and will be difficult for many maintenance programmers to understand. For ease of maintenance and initial program development, extensive tutorial information should be provided for the development team. This information must be retained and updated to reduce overall maintenance costs, which are dramatically impacted by the time spent in understanding a system prior to being able to reliably modify the system.

Development Approach

A phased implementation of the new compression algorithm is advisable to limit operational risks, adjust compression parameters, and gauge any processing impacts in advance of a full implementation. The initial program development should begin with the MVS version of BulkData to gain experience with the algorithm prior to expanding its use to the PC, UNISYS, and other customer unique platforms.

Proposed Project Schedule

Table VI lists project tasks to give management a clearer picture of what needs to be done to complete this project. The project must first get approval and appropriate funding prior to the development of a concrete development schedule that will be supported by the various groups responsible for this effort.

Table VI: Proposed schedule of events

DATE (*)	EVENT
II Quarter 1991	Begin budgeting for the project. As a worst case, budget $60,000.00 for license fees to use the algorithm.
IV Quarter 1991	Proposed standard begins ANSI approval process which should be complete by the second quarter of 1992.
IV Quarter 1991	Obtain approval and funding for the BulkData ANSI compression project.
I Quarter 1992	Develop MVS ANSI compression capability as a supported compression option within BulkData.
II Quarter 1992	Conduct an MVS pilot using selected BulkData applications and two Federal Reserve Bank pilot sites.
III Quarter 1992	Develop PC ANSI compression function within Fedline as an option.
III Quarter 1992	Release MVS version to all Districts with ANSI compression as a standard supported option.
IV Quarter 1992	Develop UNISYS ANSI compression method within UNISYS BulkData as a standard compression option.
IV Quarter 1992	Conduct a Fedline pilot in a single district using selected BulkData applications.
I Quarter 1993	Release Fedline version to all Districts with ANSI compression as a standard supported compression method.
I Quarter 1993	Conduct a UNISYS pilot using a single District with a UNISYS BulkData system communicating with MVS BulkData.
II Quarter 1993	Release UNISYS version to all UNISYS BulkData Districts as a standard compression option.
II Quarter 1993	Make a decision concerning the continued support of C64/LU1 compression algorithms for customer unique BulkData platforms.
* Note: These dates are for discussion purposes only. Actual schedules will be dependant on development team priorities, scheduled software release dates, and allocation of resources.	

CHAPTER 8 - PROJECT BENEFITS

Operating Costs

Direct cost savings for monthly dial usage charges would be about $1,000.00 for the Fifth District. This is a relatively new service and is expected to increase significantly in the next three years as the Federal Reserve promotes all-electronic ACH as well as additional services.

The bandwidth reduction caused by sending fewer characters will delay the acquisition of higher speed lines. The difference between the CPU cost impact and the network cost savings in Table IV shows a Fifth District cost allocation savings of over $1,500 per business day based on standard rates in the Fifth District. Eventually this opportunity costs would translate into direct monthly costs savings on dedicated lines.

Quality of Service

For time critical services, such as the delivery of higher volume detail MICR data, the total delivery time can be cut by over 50%. Table III shows that for a million character MICR file the total preprocessing and delivery time is reduced by 53.3%.

Promoting the Standard

The findings of the project can be added to an appendix of the draft proposed ANSI X9.DC standard to make the standard more attractive to the financial industry. More users of the standard can mean higher availability and lower costs of products that implement the standard.

These findings can also be used to provide the needed justification for the creation of an application-level International Standards Organization (ISO) data compression standard. The availability of an approved ANSI standard and the quantification of benefits from using the ANSI standard should be very helpful in getting acceptance for this type of data compression standard within the ISO community.

BulkData Computer Model

The model that simulates BulkData operation for sending files over dedicated and switched lines should be useful in

researching other potential areas for performance improvements. Questions that could be answered with this model are:

- What is the optimum segment size to maximize BulkData throughput?
- What happens to line utilization as line speed increases and why?
- What are the performance bottlenecks within BulkData that cause low utilization of high speed lines?

BulkData Preprocessor Problem

While gathering data for the BulkData model, a discrepancy in the number of disk operations was found. The current compression routines are searched for by the system and read into memory for each block being compressed. This is a district implementation oversight that exists in the Fifth District and probably many other districts as well.

For a typical ACH file of about one million bytes, the CPU time for the preprocessor with C64 compression was 4.26 seconds. A test run with the problem corrected reduced the CPU time to 1.56 seconds.

REFERENCE LIST

1. Sherman, Ken, *Data Communications - A User's Guide*, Reston Publishing Company, Inc., Reston, Virginia, 1985.

2. Shannon, C. E., "A Mathematical Theory of Communication," *Bell System Technical Journal*, July 1948.

3. Shannon, C. E., "Communications Theory of Secrecy Systems", *Computer Security Journal*, Computer Security Institute, San Francisco, California, Volume VI, Number 2, 1990.

4. CCITT Recommendation, *Data compression procedures for data circuit terminating equipment (DCE) using error correction procedures, Rec. V.42bis*, International Telecommunications Union, Geneva, Switzerland, 1990.

5. American National Standard X9.DC-19xx, *Data Compression for Wholesale Financial Telecommunications Networks.* An unpublished (as of the date of this project) ANSI draft standard included for reference in this paper.

6. Blanchard, B.S., *Logistics Engineering and Management*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

7. Richardson, George P. and Pugh, Alexander L., III, *Introduction to Systems Dynamics Modeling with DYNAMO*, The MIT Press, Cambridge, Massachusetts, 1981.

8. Forrester, Jay W., "Market Growth as Influenced by Capital Investment", *Industrial Management Review* (now the *Sloan Management Review*), 1968.

9. Blanchard, B.S., *Logistics Engineering and Management*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

10. Turner, Steven E., "How V.42bis Cuts Costs", *Data Communications*, December 1990.

APPENDIX A - DRAFT STANDARD X9.DC

ACCREDITED STANDARDS COMMITTEE X9 *dpANS X9.DC-19xx*
TITLE: X9-FINANCIAL SERVICES

Accredited by the
American National Standards Institute

AMERICAN NATIONAL STANDARD, X9.DC-19xx
Financial Institution Data Compression (Wholesale)

X9.DC-19xx

Price per copy: $30.00

NOTICE - WARNING TO READERS OF THIS DOCUMENT

This document is in the working document stage. It has not yet been processed through the consensus procedures of X9 and ANSI. Many changes which may greatly affect the contents can occur before this document becomes an AMERICAN NATIONAL STANDARD. The developmental committee may not be held responsible for the contents of this document as it currently exists.

Implementation or design based on this working paper is at the risk of the user. No advertisement implying compliance with this "STANDARD" should appear as it is erroneous and misleading to so state.

Copies of the draft proposed AMERICAN NATIONAL STANDARD will also be available from the X9 Secretariat when the document is finally announced for the two months public comment period. Notice of the comment period will be issued in the Trade Press.

Copies of this document are available from the X9 SECRETARIAT at the price listed above. Please send a self-addressed mailing label with your order.

SECRETARIAT: AMERICAN BANKERS ASSOCIATION
Standards Department
1120 Connecticut Ave., N.W.
Washington, D.C. 20036

Forward

(This forward is not part of the American National Standard for Financial Institution Data Compression for Wholesale Financial Systems, X9.DC-19xx.)

The compression algorithm in this standard is based on the compression algorithm in CCITT V.42bis. Wherever possible the original language of V.42bis was used in this standard with the permission (permission has been requested, but not granted as of the date of this draft) of the International Telecommunications Union (General Secretariat for CCITT).

REMAINDER OF FORWARD WILL BE INCLUDED HERE AT A LATER DATE

Suggestions for the improvement or revision of this standard will be welcome. They should be sent to the X9 Committee Secretariat, American Bankers Association, 1120 Connecticut Avenue, N.W., Washington, D.C. 20036.

This Standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Financial Services, X9. Committee approval of the Standard does not necessarily imply that all the committee members voted for its approval. At the time it approved this Standard, the X9 Committee had the following members:

Robert Kaminski,	*Chairman*
Cynthia Fuller,	*Secretariat*
Alice Droogen,	*Vice Chairman*
Jack Easten,	*Vice Chairman*

Organization Represented	*Representative*
American Bankers Association	*Robert Buskas*
American Designer Checks	*Neal Keenum*
American Express Company	*Eileen Bell*
Bank of America	*John Coombs*
Chase Manhattan Bank, N.A.	*John McKessy*
Chemical Bank, N.A.	*John Youngken*
Citibank, N.A.	*Seymour Rosen*
Continental Bank	*Joseph Coriaci*
Credit Union National Association	*Lori Warrens*
Deluxe Data Systems, Inc.	*Honora Norton*
EDS Corporation	*Tod Franklin*
Electronic Funds Transfer Association	*Anne Brown*
Federal Reserve Bank	*Janet Absher*
Huntington National Bank	*Charles Gearhart*
IBM Corporation	*Daniel Sundberg*
Independent Bankers Assn. of America	*Owen Freeman*
Manufacturers Hanover Trust Company	*Paul Mayland*
MasterCard International	*Alice Droogan*
Mellon Bank, N.A.	*David Taddeo*
Moore Research Center	*Delmer Oddy*
National Security Agency	*Gerard Rainville*
NCNB Corporation	*Harold Deal*
NCR Corporation	*A.R. Daniels*
New York Clearing House	*Vincent De Santis*
PNC Financial Corporation	*Jack Easton*
U.S. League of Savings Institutions	*O. Tom Thomas*
UNISYS Corporation	*Karl Sammons*
VISA International	*Jean McKenna*
Wells Fargo Bank	*Loraine Boland*
XEROX Corporation	*Glenn Mulligan*

The Standard was developed by the X9E13 Working Group - Data Compression in Wholesale Financial Telecommunications. X9E13 consisted of the following dedicated participants:

Suzanne Murphy,	*Chairman*
Organization Represented	*Representative*
Bank of America	*Suzanne Murphy*
Citicorp Investment Bank	*Richard Lefkon*
Chemical Bank	*Lillian Cimino*
Federal Reserve System	*Jim Berlin*
Federal Reserve System	*Gary Chaulklin*
Federal Reserve System	*Leon S. Chojnacki*
Hayes Microcomputer Products	*John Chiang, Ph.D.*
Hayes Microcomputer Products	*John Copeland, Ph.D.*
Ironwood Software	*Bill Becker*
National Security Agency	*Reneé Dankwerth*
National Security Agency	*Pud Reaver*
New York Clearing House	*Joeseph Pawelczyk*
New York Clearing House	*George Thomas*
New York Clearing House	*Al Wood*
System Enhancement Associates	*Thom Henderson*
Unisys Corporation	*Bill Banta*

1 Scope

SCOPE WILL BE INCLUDED HERE AT A LATER DATE

2 References

a. American National Standard X9.9-1986, Financial Institution Message Authentication (Wholesale).

b. American National Standard X9.23-1988, Financial Institution Encryption of Wholesale Financial Messages.

c. CCITT Recommendation Data compression procedures for data circuit terminating equipment (DCE) using error correction procedures, Rec. V.42bis.

3 Definitions and Abbreviations

Abbreviations or acronyms and compression variable names are described with ALL CAPS in their respective definitions.

alphabet
Set of all possible characters which may be sent or received. It is assumed in this Standard that the ordinal values of the alphabet are contiguous from 0 to Alphabet_Size - 1, where Alphabet_Size is the number of characters.

ALPHABET_SIZE
Number of characters in the alphabet.

BITS_MAX
Maximum codeword size (bits).

BITS_NOW
Current codeword size.

BYTE_SIZE
Character size (bits): Byte_Size = 8.

character
Single data element, encoded using a predefined number of bits (Byte_Size = 8).

CODE_MAX_NOW
Threshold for codeword size change.

CODE_NUMBER
Total number of codewords.

CODE_NUMBER_PARAMETER
Number of codewords parameter (see Section 4.5).

codeword
A codeword, within the context of this Standard, is a binary number in the range 0 to Code_Number - 1 which represents a string of characters in compressed form. A codeword is encoded using a number of bits equal to Bits_Now, where Bits_Now is initially 9 (i.e. Byte_Size + 1) and increases to a maximum of Bits_Max (see Section 6).

command code
Octet which is used for signalling of control information related to the compression function while in the transparent mode of operation. Command codes are distinguished from normal characters by a preceding escape character (see escape character definition above).

compressed mode
A mode of operation in which data is transmitted in codewords.

compressed operation
Compressed operation has two modes. Transitions between the modes may be automatic based on the content of the data (see Section 6.1).

COMPRESSION_INDICATOR_PARAMETER
ANSI X9.DC data compression request indicator parameter (see Section 4.5).

control codeword
A control codeword is reserved for use in signalling of control information related to the compression function while in the compressed mode of operation (see Section 8).

CONTROL_NUMBER
Number of control codewords: Control_Number = 3 (see Section 8).

ECM
Enter Compressed Mode, a command code defined in Section 8.

EID
Escape in Data, a command code defined in Section 8.

EMPTY_FIRST
index number of first dictionary entry used to store a string:
Empty_First = Alphabet_Size + Control_Number.

EMPTY_NEXT
Next empty dictionary entry.

escape character
Within the context of this Standard, the escape character is a character which, in transparent mode, indicates the beginning of a command code sequence. This has an initial value of zero, and is adjusted on each appearance of the escape character in the data stream, whether in transparent mode or compressed mode (see Section 8.2).

ETM
Enter Transparent Mode, a control codeword defined in Section 8.

leaf node
Point on a tree which represents, within the context of this Standard, the last character in a string (see Section 5.1).

ordinal value
The ordinal value of a character is the numerical equivalent of the binary encoding of the character. For example, the character "A" when encoded as 01000001 would have an ordinal value of 65_{10}.

root node
A root node is a point on a tree which represents, within the context of this Standard, the first character in a string (see Figure 16 and Section 5.1).

STRING_MAX
Maximum string length.

STRING_MAX_PARAMETER
Maximum string length (see Section 4.5).

transparent mode
A mode of operation in which compression has been selected but data is being transmitted in uncompressed form. Transparent mode command code sequences may be inserted into the data stream.

tree structure
Abstract data structure which is used in this Standard to represent a set of strings with the same initial character (see Figure 16 and Section 5.1).

uncompressed operation
A mode of operation in which compression has not been selected. No data compression functions are performed on the data.

4 Supporting structures

4.1 Data compression function capabilities

The data compression function shall implement the procedures defined in this Standard, which result in the efficient encoding of data prior to transmission over the error controlled connection, and shall have the following capabilities:

a) initialization of the data compression function;
b) data compression encoding and decoding;
c) a mechanism for switching between compressed and transparent modes of operation.

4.2 Requirements for error correcting procedures

For correct operation of the data compression function it is necessary that an error correcting procedure be implemented between the two entities using this Standard. Undetected bit errors will cause a failure of the data compression function. Use of an appropriate frame check sequence (FCS) substantially reduces the possibility of such errors.

4.3 Compression sequence

Because encrypted data is not compressible, compression shall take place prior to encryption. If application data is going to be authenticated, authentication should occur prior to any manipulation of the data in preparing the data for transmission (such as encryption or compression). If the particular network is sensitive to any characters (e.g. transmission control characters) filtering shall occur after compression and encryption. If all four functions are provided, Table VII shows logical sequence of these functions:

Data Input	*Function*	*Data Output*	*Standard*
(Data)	*Authentication*	*(Data, MAC)*	*ANSI X9.9*
(Data, MAC)	*Compress*	*Compress(data, MAC)*	*ANSI X9.DC*
Compress(data, MAC)	*Encryption*	*Encrypt(compress (data, MAC))*	*ANSI X9.23*
Encrypt(compress(data, MAC))	*Filtering*	*Filter(encrypt(compress(data, MAC)))*	*ANSI X9.23*

TABLE *VII:* COMPRESSION SEQUENCE

4.4 Filtering

SECTION ON FILTERING WILL BE ADDED HERE

4.5 Determination of data compression parameters

The decision to use compression and the mechanism used to make compression parameters known to the compression encoder/decoder is beyond the scope of this standard. This standard provides a description of the required parameters and an optional format for communicating these parameters for interoperability in Table VIII. The parameters defined below indicate the compression algorithm to use, the maximum number of codewords, and the maximum string length.

Compression_Indicator_Parameter may provide a more flexible migration to future ANSI compression algorithms. Compression_Indicator_Parameter specifies whether or not compression is to be used and the method used to compress the data.

Code_Number_Parameter represents a proposed value of Code_Number the total number of codewords. Code_Number_Parameter shall have a default value of 512, which is its minimum value; a maximum value is not specified within this Standard. Any attempt to specify less than the minimum value shall be considered a procedural error. (See Appendix I for guidance on the choice of value of Code_Number, and its effect on performance.)

String_Max_Parameter is the proposed value for String_Max, the maximum string length. The default value of String_Max_Parameter is 6, and the permitted range is from 6 to 250. The values outside this range are invalid; and attempt to specify such values shall be regarded as a procedural error.

Optional compression parameter data stream format for interoperability			
BYTE	PARAMETER NAME	VALUES	MEANING
0	(length)	4	Decimal length of the following compression parameters
1	Compression_Indicator_Parameter	0	Compression not used
		1 to 99	Values may be used by implementors to indicate non-ANSI defined compression algorithms
		100	Indicates the ANSI X9.DC-19xx compression algorithm
		101 to 255	Reserved for future ANSI compression algorithms
2-3	Code_Number_Parameter	512 1024 2048 4096 8192 16384 32768	The maximum number of codewords. Values of 512, 1024, 2049, 4096, 8192,16384, and 32768 correspond to maximum code bit-lengths of 9, 10, 11, 12, 13, 14, and 15 respectively.
4	String_Max_Parameter	6-250	The maximum string length.

TABLE VIII: OPTIONAL FORMAT FOR DATA COMPRESSION PARAMETERS

4.6 Compression function error detection

The following conditions recognized by the decoder result in the generation of an error indication:

a) receipt of a STEPUP codeword when it would cause the value of Bits_Now to exceed Bits_Max;
b) receipt of a codeword, at any time, equal to Empty_Next;
c) receipt of a codeword representing an empty dictionary entry;
d) receipt of a reserved command code.

5 Procedures for dictionary use and maintenance

5.1 General

The data compression function employs an algorithm in which a string of characters is encoded using a dynamically increasing codeword size. The process builds dictionaries, in which the strings are stored, and which are dynamically updated during normal operation.

The dictionary functions are:

a) string matching, in which a sequence of characters is read, and the dictionary searched for the resulting string (see Section 5.3);
b) updating, in which a new string is added to the dictionary (see Section 5.4);
c) the deletion of infrequently used strings in order that storage capacity may be reused (see Section 5.5).

The dictionary used to store strings for use in the encoding and decoding process may be logically represented using an abstract data structure. The dictionary can be considered to contain a set of trees, as shown in Figure 16, each with a root corresponding to a character in the alphabet. With the 8-bit character format there will be 256 trees.

A tree represents the set of known strings beginning with one specific character, and each node or point in the tree represents one of this set of strings. The trees in Figure 16 represent the strings A, B, BA, BAG, BAR, BAT, BI, BIN, C, D, DE, DO and DOG.

A node that has no dependant nodes, represented by the hierarchically lower level in the tree, is a leaf node. A leaf node represents the last character in a string.

A node that has no parent, represented by the hierarchically higher level in the tree, is a root node. A root node represents the first character in a string.

Associated with each node is a codeword used to uniquely identify the node. The assignment of codewords within the encoder dictionary of a data compression function, and the corresponding assignment of codewords within the decoder dictionary of a peer data compression function are equivalent, and the codeword thus provides a reversible encoding of a string.

5.2 Dictionary initialization procedure

In the initial condition, each tree in the dictionary shall consist only of a root node. The codeword associated with each root node shall be Control_Number (the number of control codewords) plus the ordinal value of the character represented by the node. The counter Empty_Next, used in the allocation of new nodes (see Section 5.5), shall be set to Empty_First.

5.3 String matching procedure

This procedure has the function of matching a sequence of characters (string) with a dictionary entry. The procedure shall commence with a single character representing the first character in the string. The following steps are then applied:

a) a string shall be formed from the first character;
b) if the string matches a dictionary entry, and the entry is not that entry created by the last invocation of the string matching procedure, then the next character shall be read and appended to the string and this step repeated;
c) If the string does not match a dictionary entry or matches the entry created by the last invocation of the string matching procedure, the last character appended to the string shall be removed. The string thus shortened represents

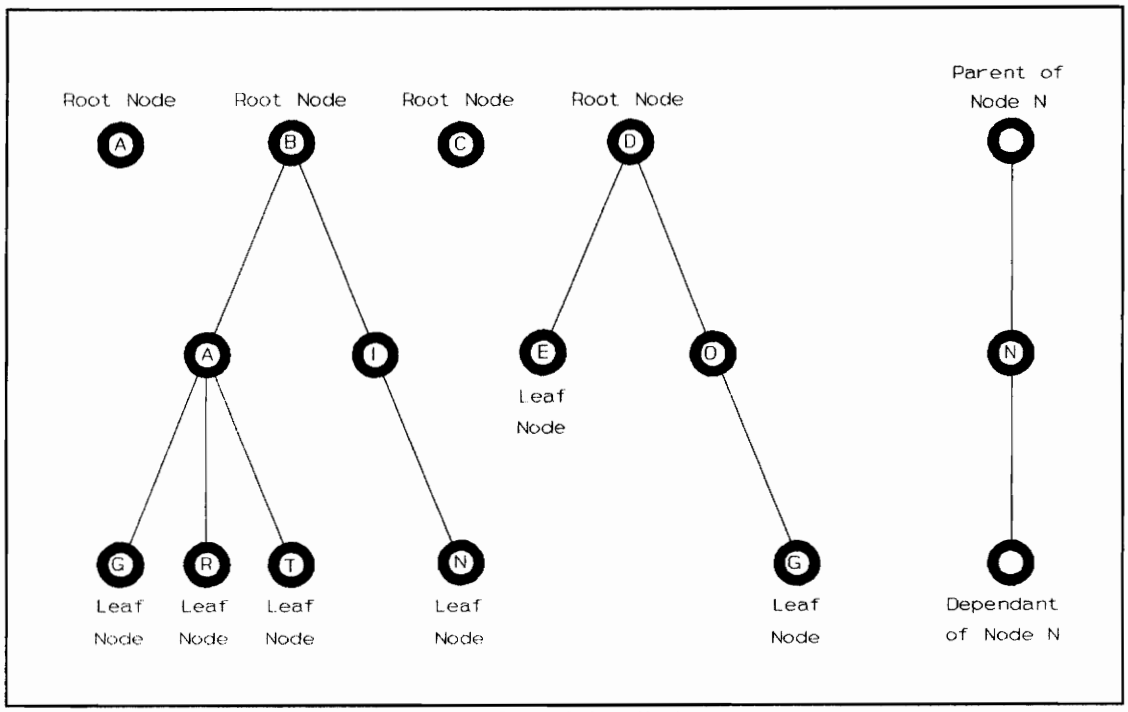

Figure 16: Tree based representation of the dictionary

the longest matched string and the last character represents the unmatched character.

This procedure will match the longest string of characters, except when a transition between transparent and compressed modes of operation occurs.

When in transparent mode the encoder shall use only the criteria specified above for terminating the string matching procedure.
If the string matching procedure is terminated before a longest match is found, the next character shall be considered to be the "unmatched character" for the purposes of updating the dictionary and restarting the string matching procedure.

5.4 Procedure for adding strings to the dictionary

In order to maintain efficient compression, the dictionary is adapted by the addition of new strings. A new string shall be formed by appending a single character to an existing string, thereby adding a new node onto a tree. The single character shall be the unmatched character resulting from the string matching operation, or the prefix character resulting from the string decoding operation. Following this procedure, the single character required to restart the string matching procedure will be the unmatched character.

There are two conditions under which a new string shall not be added:

a) if this would result in the maximum string length, String_Max, being exceeded;
b) if the string is already in the dictionary.

Immediately after the creation of a dictionary entry, the procedure for recovering a dictionary entry shall be applied.

5.5 Procedure for recovering a dictionary entry

This section defines a systematic procedure for recovering dictionary entries for re-use when all available entries have been filled. When the last available dictionary entry has been assigned, this procedure recovers a single entry, maintaining the association between the empty entry and its codeword.

A counter Empty_Next indicates the codeword associated with the next empty dictionary entry, and is maintained in the range Empty_First to Code_Number - 1. On entry to this procedure, the value of Empty_Next remains as updated during the procedure in Section 5.4.

The procedure shall be applied only after the creation of a new dictionary entry, and shall consist of the following steps:

a) counter Empty_Next shall be incremented;
b) if the value of Empty_Next exceeds Code_Number - 1 then Empty_Next shall be set to Empty_First;
c) if the node identified by the codeword with value Empty_Next is in use and not a leaf node, then go to step a);
d) if the node is a leaf node, then it shall be detached from its parent.

6 Operation of the encoding function

6.1 General

The encoding function has five principal operations:

a) string matching, in which a sequence of input characters is matched with a dictionary entry (see Section 6.3);
b) encoding, in which the codeword of the matched dictionary entry is represented as a binary value of length Bits_Now bits (see Section 6.4);
c) transfer, in which either the codeword(s) in compressed mode or the characters in transparent mode are output (see Section 6.5);
d) dictionary updating, in which a new dictionary entry is created, using the matched dictionary entry and the unmatched character (see Section 6.6);
e) node recovery, in which a dictionary entry is recovered for use in the next dictionary update (see Section 6.7).

The encoding function operates in one of two modes, transparent mode and compressed mode, switching between these modes on the basis of the test applied in f) below. The sequence of operations, and the cycling of the escape character (see Section 8) are identical in the two modes of operation.

The encoder shall support one further operation, which shall be applied only during the string matching procedure in accordance with Section 5.3:

f) data compressibility testing, in which the efficiency of the encoding process is estimated and transparent mode or compressed mode selected to maximize efficiency (see Section 6.8);

6.2 Initial conditions

The data compression function shall initialize the encoder to the following state:

a) the dictionary shall be set to the initial condition described in Section 5.2;
b) the codeword size Bits_Now shall be set to Byte_Size + 1;
c) the threshold Code_Max_Now shall be set to two times Alphabet_Size;
d) the function shall be set to transparent mode;
e) the escape character shall be assigned the ordinal value 0.

6.3 String matching

The data compression function shall apply the string matching procedure defined in Section 5.3. The initial character required shall be the unmatched character resulting from the most recent invocation of this procedure.

6.4 Encoding

This procedure is used when in the compressed mode of operation. Its purposes is to represent the codeword as a sequence of Bits_Now bits; the order and numbering of the bits is shown in Table IX.

If the codeword corresponding to the matched dictionary entry is numerically equal to or greater than the threshold Code_Max_Now then:

a) the STEPUP control codeword shall be encoded and transferred using the current codeword size (Bits_Now);
b) the codeword size Bits_Now, shall be increased by 1;
c) multiply Code_Max_Now by 2;
d) if the codeword is still numerically greater than or equal to Code_Max_Now, steps a) to c) shall be repeated.

The codeword is then transferred in accordance with the procedures defined in Section 6.5.

BIT NUMBER WITHIN OCTET									
8	7	6	5	4	3	2	1	OCTET	MODE
A_5	A_4	A_3	A_2	A_1	..	..	..	i	COMPRESSED MODE
B_2	B_1	A_{11}	A_{10}	A_9	A_8	A_7	A_6	i+1	
B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	B_3	i+2	
0	0	0	0	0	0	0	B_{11}	i+3	
C_8	C_7	C_6	C_5	C_4	C_3	C_2	C_1	i+4	TRANSPARENT MODE

TABLE *IX:* MAPPING OF CODEWORDS INTO OCTETS

6.5 Transfer

In transparent mode, characters shall be output for transmission in octet aligned form. They may be transferred individually during the string matching procedure, or as a sequence following completion of the string matching procedure.

In compressed mode, the matched string shall be encoded according to the procedure defined in Section 6.4 and output in packed form, with the least significant bit of a codeword immediately following the most significant bit of the preceding codeword.

When the encoder changes state from transparent to compressed mode, the least significant bit of the first codeword to be transferred shall be bit 1 of the next octet position.

When the encoder changes state from compressed to transparent mode following transfer of the ETM control codeword (see Section 8) in the sequence, sufficient 0 bits shall be transmitted to ensure that the next transmitted character is octet aligned.

Table IX provides an example of the data stream output during a transition from compressed to transparent mode. Two 11-bit codewords A and B are transmitted in compressed form followed by a transition to transparent mode. In this example, the transition requires insertion of seven 0 bits in order that the first uncompressed character, C sent in transparent mode, is octet aligned.

6.6 Dictionary updating

A new dictionary node shall be created from the matched string and corresponding unmatched character returned by the string matching procedure, using the procedures defined in Section 5.4.

6.7 Node recovery

Following the creation of a new dictionary node, the node recovery procedure defined in Section 5.5 shall be applied.

6.8 Data compressibility test

The data compression function shall periodically apply a test to determine the compressibility of the data. The nature of the test is not specified in this Standard; however it would consist of a comparison of the number of bits required to represent a segment of the data stream before and after compression.

6.8.1 Transition to compressed mode

If the data compression function is in the transparent mode and determines that data compression would be effective, it shall:

a) perform the dictionary update procedure using the current accumulated string and the next character to be processed by the string matching procedure (which will be the first character of the string represented by the first codeword transmitted in compressed mode);

b) indicate to the peer data compression function that a transition to compressed mode is required, using the ECM transparent mode command sequence (see Section 8.1);

c) enter compressed mode.

6.8.2 Transition to transparent mode

If the data compression function is in the compressed mode and determines that the data stream is currently not compressible, it shall:

a) ensure that the codeword representing any partially encoded data has been transferred in accordance with the procedure given in Section 6.4 and 6.5;

b) perform the dictionary update procedure using the current accumulated string and the next character to be processed by the string matching procedure (which will be the first character transmitted in transparent mode);

c) indicate to the peer data compression function by transferring the ETM control codeword (see Section 8) a transition to transparent mode;

d) transmit sufficient 0 bits to recover octet alignment (see Section 6.5);

e) change the state to transparent mode

6.8.3 RESET function

In transparent mode the RESET command code may be used to indicate to the peer data compression function that the encoder dictionary is about to be re—initialized according to the procedures given in Section 5.2 and 6.2. The RESET command code is sent using the escape character value before re-initialization takes place.

The circumstances under which the encoder requests a dictionary reset are not defined in this Standard, but would generally result from the encoder determining that some improvement in performance would result from resetting the dictionary.

7 Operation of the decoding function

The decoding function shall be capable of operation in both compressed and transparent modes, and shall operate in a manner consistent with that defined in Section 5, 6 and 8.

On initialization or receipt of a RESET command code from the peer data compression function, the data compression function shall initialize the decoding function in accordance with the procedures defined in Section 5.2 and 6.2.

In transparent mode, the decoding function shall apply the string matching procedure given in Section 5.3, in order that the decoder dictionary may be maintained in a compatible state to the peer (remote) encoder dictionary. On receipt of the ECM or EID command code, the decoding function shall operate in a manner consistent with the encoder operations defined in Section 6.8.1 and 8.2. New dictionary entries shall be created in a manner consistent with the procedures defined in Section 5.4 and 6.3.

In compressed mode the decoding function shall recover the encoded strings. On receipt of an ETM codeword the decoder shall operate in a manner consistent with the encoder operations defined in Section 6.8.2. New dictionary entries shall be created using the procedure defined in Section 5.4, with the first (prefix) character of the most recently decoded string being appended to the previous decoded string.

The decoder shall regard the STEPUP control codeword as an indication that the encoder has increased the codeword size in accordance with the procedures defined in Section 6.4.

8 Communications between peer data compression functions

8.1 Control codewords and command codes

The control codewords and command codes allocated for communication between peer data compression functions are given in Table X.

8.2 Procedures for use of the escape sequence

A transparent mode command sequence shall consist of the escape character followed by one of the command codes listed in Section 8.1 above.

To reduce data expansion resulting from the escape mechanism defined below, if the current escape character is detected within the data stream, the data compression function shall:

- *a) if in transparent mode, transfer the detected escape character, and transmit the EID code, and then*
- *b) in both transparent and compressed modes, modify the value of the escape character by adding to it the decimal value 51, the addition to be performed modulo 256.*

CONTROL CODE WORDS (USED IN COMPRESSED MODE)		
Code word	***Name***	***Description***
0	*ETM*	*Enter transparent mode*
1	*Reserved*	*Reserved for future use*
2	*STEPUP*	*Step up codeword size*
COMMAND CODES (USED IN TRANSPARENT MODE)		
Value	***Name***	***Description***
0	*ECM*	*Enter compressed mode*
1	*EID*	*Escape character in data*
2	*RESET*	*Force reinitialization*
3 to 255	*Reserved*	

TABLE X: CONTROL CODEWORDS AND COMMAND CODES

APPENDIX B - MODEL LISTING

```
SL.K=SL.J+DT*(SRI.JK-SRO.JK)                                    L,1
SL=SLN                                                          N,1.1
SLN=0                                                           C,1.2
     SL     - segment level (segments) <1>
     SRI    - segment rate in (seg/sec) <2>
     SRO    - seg rate out (seg/sec) <14>

SRI.KL=CLIP(0,ARI.K,TIME.K,TI.K)                                R,2
     SRI    - segment rate in (seg/sec) <2>
     ARI    - average segment rate in (seg/sec) <3>
     TI     - time for segments (sec) <9>

ARI.K=NS.K/TI.K                                                 A,3
     ARI    - average segment rate in (seg/sec) <3>
     NS     - number of segments (seg) <4>
     TI     - time for segments (sec) <9>

NS.K=(CBO.K+BOV.K+SOV.K)/SLEN                                   A,4
     NS     - number of segments (seg) <4>
     CBO    - comp. Bytes out (bytes) <6>
     BOV    - block overhead (bytes) <7>
     SOV    - segment overhead (bytes) <8>
     SLEN   - segment length (bytes) <24>

SP.K=ARI.K*TIME.K                                               A,5
     SP     - seg processed (seg) <5>
     ARI    - average segment rate in (seg/sec) <3>

CBO.K=BI*CR                                                     A,6
BI=1000000                                                      C,6.1
CR=.481                                                         C,6.2
     CBO    - comp. Bytes out (bytes) <6>
     BI     - bytes in (bytes) <6>
     CR     - compression ratio <6>

BOV.K=(BI/BS)*BOVC                                              A,7
BS=940                                                          C,7.1
BOVC=3                                                          C,7.2
     BOV    - block overhead (bytes) <7>
     BI     - bytes in (bytes) <6>
     BS     - block size (bytes) <7>
     BOVC   - block overhead (bytes) <7>

SOV.K=((CBO.K+BOV.K)/SLEN)*SOVC                                 A,8
SOVC=5                                                          C,8.1
     SOV    - segment overhead (bytes) <8>
     CBO    - comp. Bytes out (bytes) <6>
```

```
     BOV    - block overhead (bytes) <7>
     SLEN   - segment length (bytes) <24>
     SOVC   - segment overhead (bytes) <8>

TI.K=CPUNS.K+IONS.K                                          A,9
     TI     - time for segments (sec) <9>
     CPUNS  - cpu time required (sec) <10>
     IONS   - dasd i/o time (sec) <11>



CPUNS.K=(BI*CPUPB)+CPUOVC                                    A,10
CPUPB=.00000113                                              C,10.1
CPUOVC=.22                                                   C,10.2
     CPUNS  - cpu time required (sec) <10>
     BI     - bytes in (bytes) <6>
     CPUPB  - cpu per byte (sec) <10>
     CPUOVC - cpu overhead (sec) <10>

IONS.K=SDAT*TIO.K                                            A,11
SDAT=0.01                                                    C,11.1
     IONS   - dasd i/o time (sec) <11>
     SDAT   - seq dasd access time (sec) <11>
     TIO    - total dasd accesses <12>

TIO.K=BIN.K+NS.K+ODC                                         A,12
     TIO    - total dasd accesses <12>
     BIN    - blocks read <13>
     NS     - number of segments (seg) <4>
     ODC    - overhead disk accesses <13>

BIN.K=BI/BS                                                  A,13
ODC=24                                                       C,13.1
     BIN    - blocks read <13>
     BI     - bytes in (bytes) <6>
     BS     - block size (bytes) <7>
     ODC    - overhead disk accesses <13>

SRO.KL=CLIP(RO.K,0,TIME.K,TI.K)                              R,14
     SRO    - seg rate out (seg/sec) <14>
     RO     - rate out (seg/sec) <15>
     TI     - time for segments (sec) <9>

RO.K=SWITCH(0,ARO.K,SL.K)                                    A,15
     RO     - rate out (seg/sec) <15>
     ARO    - avg seg rate out (seg/sec) <17>
     SL     - segment level (segments) <1>
```

```
TT.K=TI.K+TOU.K                                          A,16
     TT     - total time (sec) <16>
     TI     - time for segments (sec) <9>
     TOU    - time for output (sec) <18>

ARO.K=NS.K/TOU.K                                         A,17
     ARO    - avg seg rate out (seg/sec) <17>
     NS     - number of segments (seg) <4>
     TOU    - time for output (sec) <18>

TOU.K=SST+BT.K                                           A,18
SST=1.5                                                  C,18.1
     TOU    - time for output (sec) <18>
     SST    - session start time (sec) <18>
     BT     - bit time (sec) <19>

BT.K=TB.K/TRIB.K                                         A,19
     BT     - bit time (sec) <19>
     TB     - total bits (bits) <20>
     TRIB   - trans rate (bits/sec) <21>


TB.K=NS.K*BYTE*SLEN                                      A,20
BYTE=8                                                   C,20.1
     TB     - total bits (bits) <20>
     NS     - number of segments (seg) <4>
     BYTE   - bits per byte (bits) <20>
     SLEN   - segment length (bytes) <24>

TRIB.K=(BPB.K*PS.K)/TPB.K                                A,21
     TRIB   - trans rate (bits/sec) <21>
     BPB    - bits per block (bits) <22>
     PS     - probability of success <26>
     TPB    - time per block (sec) <27>

BPB.K=BYTE*IB.K                                          A,22
     BPB    - bits per block (bits) <22>
     BYTE   - bits per byte (bits) <20>
     IB     - info block size (bytes) <23>

IB.K=SBS.K-SDOV.K                                        A,23
     IB     - info block size (bytes) <23>
     SBS    - sdlc block size (bytes) <24>
     SDOV   - sdlc overhead (bytes) <25>
```

```
SBS.K=SLEN+SDOV.K                                          A,24
SLEN=6144                                                  C,24.1
     SBS    - sdlc block size (bytes) <24>
     SLEN   - segment length (bytes) <24>
     SDOV   - sdlc overhead (bytes) <25>

SDOV.K=LH+TH+RH+LT                                         A,25
LH=3                                                       C,25.1
TH=26                                                      C,25.2
RH=3                                                       C,25.3
LT=3                                                       C,25.4
     SDOV   - sdlc overhead (bytes) <25>
     LH     - link header (bytes) <25>
     TH     - transmission header (bytes) <25>
     RH     - request header (bytes) <25>
     LT     - link trailer (bytes) <25>

PS.K=1-PF                                                  A,26
PF=.001                                                    C,26.1
     PS     - probability of success <26>
     PF     - probability of failure <26>

TPB.K=BLT.K+IBLT                                           A,27
     TPB    - time per block (sec) <27>
     BLT    - block time (sec) <28>
     IBLT   - inter-block time (sec) <29>

BLT.K=SBS.K/CLS.K                                          A,28
     BLT    - block time (sec) <28>
     SBS    - sdlc block size (bytes) <24>
     CLS    - line speed (bytes/sec) <29>


CLS.K=BLS/BYTE                                             A,29
BLS=56000                                                  C,29.1
IBLT=.4                                                    C,29.2
     CLS    - line speed (bytes/sec) <29>
     BLS    - line speed (bits/sec) <29>
     BYTE   - bits per byte (bits) <20>
     IBLT   - inter-block time (sec) <29>

LENGTH.K=TT.K                                              A,30
     LENGTH - length of simulation <30>
     TT     - total time (sec) <16>

SAVPER.K=TT.K/40                                           A,31
```

```
     SAVPER - plot 10 points <31>
     TT     - total time (sec) <16>

SAVE SRO,SL,SRI,TRIB,TI,TOU,TT,CPUNS                                  32
     SRO    - seg rate out (seg/sec) <14>
     SL     - segment level (segments) <1>
     SRI    - segment rate in (seg/sec) <2>
     TRIB   - trans rate (bits/sec) <21>
     TI     - time for segments (sec) <9>
     TOU    - time for output (sec) <18>
     TT     - total time (sec) <16>
     CPUNS  - cpu time required (sec) <10>

SPEC DT=.25/REL_ERR=1E-6/ABS_ERR=1E-6                                  33
```

LIST OF VARIABLES

SYMBOL	T	WHR-CMP	DEFINITION
ABS_ERR	C	33	
ARI	A	3	average segment rate in (seg/sec) <3>
ARO	A	17	avg seg rate out (seg/sec) <17>
BI	C	6.1	bytes in (bytes) <6>
BIN	A	13	blocks read <13>
BLS	C	29.1	line speed (bits/sec) <29>
BLT	A	28	block time (sec) <28>
BOV	A	7	block overhead (bytes) <7>
BOVC	C	7.2	block overhead (bytes) <7>
BPB	A	22	bits per block (bits) <22>
BS	C	7.1	block size (bytes) <7>
BT	A	19	bit time (sec) <19>
BYTE	C	20.1	bits per byte (bits) <20>
CBO	A	6	comp. Bytes out (bytes) <6>
CLS	A	29	line speed (bytes/sec) <29>
CPUNS	A	10	cpu time required (sec) <10>
CPUOVC	C	10.2	cpu overhead (sec) <10>
CPUPB	C	10.1	cpu per byte (sec) <10>
CR	C	6.2	compression ratio <6>
DT	C	33	
IB	A	23	info block size (bytes) <23>
IBLT	C	29.2	inter-block time (sec) <29>
IONS	A	11	dasd i/o time (sec) <11>
LENGTH	A	30	length of simulation <30>
LH	C	25.1	link header (bytes) <25>
LT	C	25.4	link trailer (bytes) <25>
NS	A	4	number of segments (seg) <4>
ODC	C	13.1	overhead disk accesses <13>
PF	C	26.1	probability of failure <26>
PS	A	26	probability of success <26>
REL_ERR	C	33	
RH	C	25.3	request header (bytes) <25>
RO	A	15	rate out (seg/sec) <15>
SAVPER	A	31	plot 10 points <31>
SBS	A	24	sdlc block size (bytes) <24>
SDAT	C	11.1	seq dasd access time (sec) <11>
SDOV	A	25	sdlc overhead (bytes) <25>
SL	L	1	segment level (segments) <1>
	N	1.1	

SLEN	C	24.1	segment length (bytes) <24>
SLN	C	1.2	
SOV	A	8	segment overhead (bytes) <8>
SOVC	C	8.1	segment overhead (bytes) <8>
SP	A	5	seg processed (seg) <5>
SRI	R	2	segment rate in (seg/sec) <2>
SRO	R	14	seg rate out (seg/sec) <14>
SST	C	18.1	session start time (sec) <18>
TB	A	20	total bits (bits) <20>
TH	C	25.2	transmission header (bytes) <25>
TI	A	9	time for segments (sec) <9>
TIO	A	12	total dasd accesses <12>
TOU	A	18	time for output (sec) <18>
TPB	A	27	time per block (sec) <27>
TRIB	A	21	trans rate (bits/sec) <21>
TT	A	16	total time (sec) <16>

Number of symbol table entries - 57

```
SL.K=SL.J+DT*(SRI.JK-SRO.JK)                                    L,1
SL=SLN                                                          N,1.1
SLN=0                                                           C,1.2
     SL     - segment level (segments) <1>
     SRI    - segment rate in (seg/sec) <2>
     SRO    - seg rate out (seg/sec) <13>

SRI.KL=CLIP(0,ARI.K,TIME.K,TI.K)                                R,2
     SRI    - segment rate in (seg/sec) <2>
     ARI    - average segment rate in (seg/sec) <3>
     TI     - time for segments (sec) <8>

ARI.K=NS.K/TI.K                                                 A,3
     ARI    - average segment rate in (seg/sec) <3>
     NS     - number of segments (seg) <4>
     TI     - time for segments (sec) <8>

NS.K=(CBO.K+BOV.K+SOV.K)/SLEN                                   A,4
     NS     - number of segments (seg) <4>
     CBO    - comp. Bytes out (bytes) <5>
     BOV    - block overhead (bytes) <6>
     SOV    - segment overhead (bytes) <7>
     SLEN   - segment length (bytes) <23>

CBO.K=BI*CR                                                     A,5
BI=1000000                                                      C,5.1
CR=.178                                                         C,5.2
     CBO    - comp. Bytes out (bytes) <5>
     BI     - bytes in (bytes) <5>
     CR     - compression ratio <5>

BOV.K=(BI/BS)*BOVC                                              A,6
BS=940                                                          C,6.1
BOVC=3                                                          C,6.2
     BOV    - block overhead (bytes) <6>
     BI     - bytes in (bytes) <5>
     BS     - block size (bytes) <6>
     BOVC   - block overhead (bytes) <6>

SOV.K=((CBO.K+BOV.K)/SLEN)*SOVC                                 A,7
SOVC=5                                                          C,7.1
     SOV    - segment overhead (bytes) <7>
     CBO    - comp. Bytes out (bytes) <5>
     BOV    - block overhead (bytes) <6>
     SLEN   - segment length (bytes) <23>
     SOVC   - segment overhead (bytes) <7>
```

```
TI.K=CPUNS.K+IONS.K                                          A,8
     TI     - time for segments (sec) <8>
     CPUNS  - cpu time required (sec) <9>
     IONS   - dasd i/o time (sec) <10>


CPUNS.K=(BI*CPUPBI)+(BI*CR*CPUPBO)+CPUOVC                    A,9
CPUPBI=.0000012                                              C,9.1
CPUPBO=.0000117                                              C,9.2
CPUOVC=.22                                                   C,9.3
     CPUNS  - cpu time required (sec) <9>
     BI     - bytes in (bytes) <5>
     CPUPBI - cpu per byte of input (sec) <9>
     CR     - compression ratio <5>
     CPUPBO - cpu per byte of output (sec) <9>
     CPUOVC - cpu overhead (sec) <9>

IONS.K=SDAT*TIO.K                                            A,10
SDAT=0.01                                                    C,10.1
     IONS   - dasd i/o time (sec) <10>
     SDAT   - seq dasd access time (sec) <10>
     TIO    - total dasd accesses <11>

TIO.K=BIN.K+NS.K+ODC                                         A,11
     TIO    - total dasd accesses <11>
     BIN    - blocks read <12>
     NS     - number of segments (seg) <4>
     ODC    - overhead disk accesses <12>

BIN.K=BI/BS                                                  A,12
ODC=24                                                       C,12.1
     BIN    - blocks read <12>
     BI     - bytes in (bytes) <5>
     BS     - block size (bytes) <6>
     ODC    - overhead disk accesses <12>

SRO.KL=CLIP(RO.K,0,TIME.K,TI.K)                              R,13
     SRO    - seg rate out (seg/sec) <13>
     RO     - rate out (seg/sec) <14>
     TI     - time for segments (sec) <8>

RO.K=SWITCH(0,ARO.K,SL.K)                                    A,14
     RO     - rate out (seg/sec) <14>
     ARO    - avg seg rate out (seg/sec) <16>
     SL     - segment level (segments) <1>
```

```
TT.K=TI.K+TOU.K                                                    A,15
     TT     - total time (sec) <15>
     TI     - time for segments (sec) <8>
     TOU    - time for output (sec) <17>

ARO.K=NS.K/TOU.K                                                   A,16
     ARO    - avg seg rate out (seg/sec) <16>
     NS     - number of segments (seg) <4>
     TOU    - time for output (sec) <17>

TOU.K=SST+BT.K                                                     A,17
SST=1.5                                                            C,17.1
     TOU    - time for output (sec) <17>
     SST    - session start time (sec) <17>
     BT     - bit time (sec) <18>


BT.K=TB.K/TRIB.K                                                   A,18
     BT     - bit time (sec) <18>
     TB     - total bits (bits) <19>
     TRIB   - trans rate (bits/sec) <20>

TB.K=NS.K*BYTE*SLEN                                                A,19
BYTE=8                                                             C,19.1
     TB     - total bits (bits) <19>
     NS     - number of segments (seg) <4>
     BYTE   - bits per byte (bits) <19>
     SLEN   - segment length (bytes) <23>

TRIB.K=(BPB.K*PS.K)/TPB.K                                          A,20
     TRIB   - trans rate (bits/sec) <20>
     BPB    - bits per block (bits) <21>
     PS     - probability of success <25>
     TPB    - time per block (sec) <26>

BPB.K=BYTE*IB.K                                                    A,21
     BPB    - bits per block (bits) <21>
     BYTE   - bits per byte (bits) <19>
     IB     - info block size (bytes) <22>

IB.K=SBS.K-SDOV.K                                                  A,22
     IB     - info block size (bytes) <22>
     SBS    - sdlc block size (bytes) <23>
     SDOV   - sdlc overhead (bytes) <24>

SBS.K=SLEN+SDOV.K                                                  A,23
```

```
SLEN=6144                                                    C,23.1
     SBS    - sdlc block size (bytes) <23>
     SLEN   - segment length (bytes) <23>
     SDOV   - sdlc overhead (bytes) <24>

SDOV.K=LH+TH+RH+LT                                           A,24
LH=3                                                         C,24.1
TH=26                                                        C,24.2
RH=3                                                         C,24.3
LT=3                                                         C,24.4
     SDOV   - sdlc overhead (bytes) <24>
     LH     - link header (bytes) <24>
     TH     - transmission header (bytes) <24>
     RH     - request header (bytes) <24>
     LT     - link trailer (bytes) <24>

PS.K=1-PF                                                    A,25
PF=.001                                                      C,25.1
     PS     - probability of success <25>
     PF     - probability of failure <25>

TPB.K=BLT.K+IBLT                                             A,26
     TPB    - time per block (sec) <26>
     BLT    - block time (sec) <27>
     IBLT   - inter-block time (sec) <28>


BLT.K=SBS.K/CLS.K                                            A,27
     BLT    - block time (sec) <27>
     SBS    - sdlc block size (bytes) <23>
     CLS    - line speed (bytes/sec) <28>

CLS.K=BLS/BYTE                                               A,28
BLS=56000                                                    C,28.1
IBLT=.4                                                      C,28.2
     CLS    - line speed (bytes/sec) <28>
     BLS    - line speed (bits/sec) <28>
     BYTE   - bits per byte (bits) <19>
     IBLT   - inter-block time (sec) <28>

LENGTH.K=TT.K                                                A,29
     LENGTH - length of simulation <29>
     TT     - total time (sec) <15>

SAVPER.K=TT.K/40                                             A,30
     TT     - total time (sec) <15>
```

```
SAVE SRO,SL,SRI,TRIB,TI,TOU,TT,CPUNS                                    31
     SRO    - seg rate out (seg/sec) <13>
     SL     - segment level (segments) <1>
     SRI    - segment rate in (seg/sec) <2>
     TRIB   - trans rate (bits/sec) <20>
     TI     - time for segments (sec) <8>
     TOU    - time for output (sec) <17>
     TT     - total time (sec) <15>
     CPUNS  - cpu time required (sec) <9>

SPEC DT=.25/REL_ERR=1E-6/ABS_ERR=1E-6                                    32
```

LIST OF VARIABLES

SYMBOL	T	WHR-CMP	DEFINITION
ABS_ERR	C	32	
ARI	A	3	average segment rate in (seg/sec) <3>
ARO	A	16	avg seg rate out (seg/sec) <16>
BI	C	5.1	bytes in (bytes) <5>
BIN	A	12	blocks read <12>
BLS	C	28.1	line speed (bits/sec) <28>
BLT	A	27	block time (sec) <27>
BOV	A	6	block overhead (bytes) <6>
BOVC	C	6.2	block overhead (bytes) <6>
BPB	A	21	bits per block (bits) <21>
BS	C	6.1	block size (bytes) <6>
BT	A	18	bit time (sec) <18>
BYTE	C	19.1	bits per byte (bits) <19>
CBO	A	5	comp. Bytes out (bytes) <5>
CLS	A	28	line speed (bytes/sec) <28>
CPUNS	A	9	cpu time required (sec) <9>
CPUOVC	C	9.3	cpu overhead (sec) <9>
CPUPBI	C	9.1	cpu per byte of input (sec) <9>
CPUPBO	C	9.2	cpu per byte of output (sec) <9>
CR	C	5.2	compression ratio <5>
DT	C	32	
IB	A	22	info block size (bytes) <22>
IBLT	C	28.2	inter-block time (sec) <28>
IONS	A	10	dasd i/o time (sec) <10>
LENGTH	A	29	length of simulation <29>
LH	C	24.1	link header (bytes) <24>
LT	C	24.4	link trailer (bytes) <24>
NS	A	4	number of segments (seg) <4>
ODC	C	12.1	overhead disk accesses <12>
PF	C	25.1	probability of failure <25>
PS	A	25	probability of success <25>
REL_ERR	C	32	
RH	C	24.3	request header (bytes) <24>
RO	A	14	rate out (seg/sec) <14>
SAVPER	A	30	
SBS	A	23	sdlc block size (bytes) <23>
SDAT	C	10.1	seq dasd access time (sec) <10>
SDOV	A	24	sdlc overhead (bytes) <24>
SL	L	1	segment level (segments) <1>
	N	1.1	
SLEN	C	23.1	segment length (bytes) <23>
SLN	C	1.2	

SOV	A	7	segment overhead (bytes) <7>
SOVC	C	7.1	segment overhead (bytes) <7>
SRI	R	2	segment rate in (seg/sec) <2>
SRO	R	13	seg rate out (seg/sec) <13>
SST	C	17.1	session start time (sec) <17>
TB	A	19	total bits (bits) <19>
TH	C	24.2	transmission header (bytes) <24>
TI	A	8	time for segments (sec) <8>
TIO	A	11	total dasd accesses <11>
TOU	A	17	time for output (sec) <17>
TPB	A	26	time per block (sec) <26>
TRIB	A	20	trans rate (bits/sec) <20>
TT	A	15	total time (sec) <15>

```
SL.K=SL.J+DT*(SRI.JK-SRO.JK)                                    L,1
SL=SLN                                                          N,1.1
SLN=0                                                           C,1.2
     SL      - segment level (segments) <1>
     SRI     - segment rate in (seg/sec) <2>
     SRO     - seg rate out (seg/sec) <14>

SRI.KL=CLIP(0,ARI.K,TIME.K,TI.K)                                R,2
     SRI     - segment rate in (seg/sec) <2>
     ARI     - average segment rate in (seg/sec) <3>
     TI      - time for segments (sec) <9>

ARI.K=NS.K/TI.K                                                 A,3
     ARI     - average segment rate in (seg/sec) <3>
     NS      - number of segments (seg) <4>
     TI      - time for segments (sec) <9>

NS.K=(CBO.K+BOV.K+SOV.K)/SLEN                                   A,4
     NS      - number of segments (seg) <4>
     CBO     - comp. Bytes out (bytes) <6>
     BOV     - block overhead (bytes) <7>
     SOV     - segment overhead (bytes) <8>
     SLEN    - segment length (bytes) <24>

SP.K=ARI.K*TIME.K                                               A,5
     SP      - seg processed (seg) <5>
     ARI     - average segment rate in (seg/sec) <3>

CBO.K=BI*CR                                                     A,6
BI=58000                                                        C,6.1
CR=.81                                                          C,6.2
     CBO     - comp. Bytes out (bytes) <6>
     BI      - bytes in (bytes) <6>
     CR      - compression ratio <6>

BOV.K=(BI/BS)*BOVC                                              A,7
BS=940                                                          C,7.1
BOVC=3                                                          C,7.2
     BOV     - block overhead (bytes) <7>
     BI      - bytes in (bytes) <6>
     BS      - block size (bytes) <7>
     BOVC    - block overhead (bytes) <7>

SOV.K=((CBO.K+BOV.K)/SLEN)*SOVC                                 A,8
SOVC=5                                                          C,8.1
     SOV     - segment overhead (bytes) <8>
     CBO     - comp. Bytes out (bytes) <6>
```

```
     BOV    - block overhead (bytes) <7>
     SLEN   - segment length (bytes) <24>
     SOVC   - segment overhead (bytes) <8>

TI.K=CPUNS.K+IONS.K                                          A,9
     TI     - time for segments (sec) <9>
     CPUNS  - cpu time required (sec) <10>
     IONS   - dasd i/o time (sec) <11>



CPUNS.K=(BI*CPUPB)+CPUOVC                                    A,10
CPUPB=.00000113                                              C,10.1
CPUOVC=.22                                                   C,10.2
     CPUNS  - cpu time required (sec) <10>
     BI     - bytes in (bytes) <6>
     CPUPB  - cpu per byte (sec) <10>
     CPUOVC - cpu overhead (sec) <10>

IONS.K=SDAT*TIO.K                                            A,11
SDAT=0.01                                                    C,11.1
     IONS   - dasd i/o time (sec) <11>
     SDAT   - seq dasd access time (sec) <11>
     TIO    - total dasd accesses <12>

TIO.K=BIN.K+NS.K+ODC                                         A,12
     TIO    - total dasd accesses <12>
     BIN    - blocks read <13>
     NS     - number of segments (seg) <4>
     ODC    - overhead disk accesses <13>

BIN.K=BI/BS                                                  A,13
ODC=24                                                       C,13.1
     BIN    - blocks read <13>
     BI     - bytes in (bytes) <6>
     BS     - block size (bytes) <7>
     ODC    - overhead disk accesses <13>

SRO.KL=CLIP(RO.K,0,TIME.K,TI.K)                              R,14
     SRO    - seg rate out (seg/sec) <14>
     RO     - rate out (seg/sec) <15>
     TI     - time for segments (sec) <9>

RO.K=SWITCH(0,ARO.K,SL.K)                                    A,15
     RO     - rate out (seg/sec) <15>
     ARO    - avg seg rate out (seg/sec) <17>
     SL     - segment level (segments) <1>
```

```
TT.K=TI.K+TOU.K                                          A,16
     TT     - total time (sec) <16>
     TI     - time for segments (sec) <9>
     TOU    - time for output (sec) <18>

ARO.K=NS.K/TOU.K                                         A,17
     ARO    - avg seg rate out (seg/sec) <17>
     NS     - number of segments (seg) <4>
     TOU    - time for output (sec) <18>

TOU.K=SST+BT.K                                           A,18
SST=30                                                   C,18.1
     TOU    - time for output (sec) <18>
     SST    - session start time (sec) <18>
     BT     - bit time (sec) <19>

BT.K=TB.K/TRIB.K                                         A,19
     BT     - bit time (sec) <19>
     TB     - total bits (bits) <20>
     TRIB   - trans rate (bits/sec) <21>


TB.K=NS.K*BYTE*SLEN                                      A,20
BYTE=8                                                   C,20.1
     TB     - total bits (bits) <20>
     NS     - number of segments (seg) <4>
     BYTE   - bits per byte (bits) <20>
     SLEN   - segment length (bytes) <24>

TRIB.K=(BPB.K*PS.K)/TPB.K                                A,21
     TRIB   - trans rate (bits/sec) <21>
     BPB    - bits per block (bits) <22>
     PS     - probability of success <26>
     TPB    - time per block (sec) <27>

BPB.K=BYTE*IB.K                                          A,22
     BPB    - bits per block (bits) <22>
     BYTE   - bits per byte (bits) <20>
     IB     - info block size (bytes) <23>

IB.K=SBS.K-SDOV.K                                        A,23
     IB     - info block size (bytes) <23>
     SBS    - sdlc block size (bytes) <24>
     SDOV   - sdlc overhead (bytes) <25>
```

```
SBS.K=PU2LEN+SDOV.K                                          A,24
PU2LEN=256                                                   C,24.1
SLEN=6144                                                    C,24.2
     SBS    - sdlc block size (bytes) <24>
     PU2LEN - ru size for pu type 2 <24>
     SDOV   - sdlc overhead (bytes) <25>
     SLEN   - segment length (bytes) <24>

SDOV.K=LH+TH+RH+LT                                           A,25
LH=3                                                         C,25.1
TH=6                                                         C,25.2
RH=3                                                         C,25.3
LT=3                                                         C,25.4
     SDOV   - sdlc overhead (bytes) <25>
     LH     - link header (bytes) <25>
     TH     - transmission header (bytes) <25>
     RH     - request header (bytes) <25>
     LT     - link trailer (bytes) <25>

PS.K=1-PF                                                    A,26
PF=.01                                                       C,26.1
     PS     - probability of success <26>
     PF     - probability of failure <26>

TPB.K=BLT.K+IBLT                                             A,27
     TPB    - time per block (sec) <27>
     BLT    - block time (sec) <28>
     IBLT   - inter-block time (sec) <29>

BLT.K=SBS.K/CLS.K                                            A,28
     BLT    - block time (sec) <28>
     SBS    - sdlc block size (bytes) <24>
     CLS    - line speed (bytes/sec) <29>


CLS.K=BLS/BYTE                                               A,29
BLS=2400                                                     C,29.1
IBLT=.4                                                      C,29.2
     CLS    - line speed (bytes/sec) <29>
     BLS    - line speed (bits/sec) <29>
     BYTE   - bits per byte (bits) <20>
     IBLT   - inter-block time (sec) <29>

LENGTH.K=TT.K                                                A,30
     LENGTH - length of simulation <30>
     TT     - total time (sec) <16>
```

```
SAVPER.K=1                                                          A,31
     SAVPER - plot 10 points <31>

SAVE SRO,SL,SRI,TRIB,TI,TOU,TT,CPUNS                                 32
     SRO    - seg rate out (seg/sec) <14>
     SL     - segment level (segments) <1>
     SRI    - segment rate in (seg/sec) <2>
     TRIB   - trans rate (bits/sec) <21>
     TI     - time for segments (sec) <9>
     TOU    - time for output (sec) <18>
     TT     - total time (sec) <16>
     CPUNS  - cpu time required (sec) <10>

SPEC DT=.25/REL_ERR=1E-6/ABS_ERR=1E-6                                33
```

LIST OF VARIABLES

SYMBOL	T	WHR-CMP	DEFINITION
ABS_ERR	C	33	
ARI	A	3	average segment rate in (seg/sec) <3>
ARO	A	17	avg seg rate out (seg/sec) <17>
BI	C	6.1	bytes in (bytes) <6>
BIN	A	13	blocks read <13>
BLS	C	29.1	line speed (bits/sec) <29>
BLT	A	28	block time (sec) <28>
BOV	A	7	block overhead (bytes) <7>
BOVC	C	7.2	block overhead (bytes) <7>
BPB	A	22	bits per block (bits) <22>
BS	C	7.1	block size (bytes) <7>
BT	A	19	bit time (sec) <19>
BYTE	C	20.1	bits per byte (bits) <20>
CBO	A	6	comp. Bytes out (bytes) <6>
CLS	A	29	line speed (bytes/sec) <29>
CPUNS	A	10	cpu time required (sec) <10>
CPUOVC	C	10.2	cpu overhead (sec) <10>
CPUPB	C	10.1	cpu per byte (sec) <10>
CR	C	6.2	compression ratio <6>
DT	C	33	
IB	A	23	info block size (bytes) <23>
IBLT	C	29.2	inter-block time (sec) <29>
IONS	A	11	dasd i/o time (sec) <11>
LENGTH	A	30	length of simulation <30>
LH	C	25.1	link header (bytes) <25>
LT	C	25.4	link trailer (bytes) <25>
NS	A	4	number of segments (seg) <4>
ODC	C	13.1	overhead disk accesses <13>
PF	C	26.1	probability of failure <26>
PS	A	26	probability of success <26>
PU2LEN	C	24.1	ru size for pu type 2 <24>
REL_ERR	C	33	
RH	C	25.3	request header (bytes) <25>
RO	A	15	rate out (seg/sec) <15>
SAVPER	A	31	plot 10 points <31>
SBS	A	24	sdlc block size (bytes) <24>
SDAT	C	11.1	seq dasd access time (sec) <11>
SDOV	A	25	sdlc overhead (bytes) <25>
SL	L	1	segment level (segments) <1>
	N	1.1	

SLEN	C	24.2	segment length (bytes) <24>
SLN	C	1.2	
SOV	A	8	segment overhead (bytes) <8>
SOVC	C	8.1	segment overhead (bytes) <8>
SP	A	5	seg processed (seg) <5>
SRI	R	2	segment rate in (seg/sec) <2>
SRO	R	14	seg rate out (seg/sec) <14>
SST	C	18.1	session start time (sec) <18>
TB	A	20	total bits (bits) <20>
TH	C	25.2	transmission header (bytes) <25>
TI	A	9	time for segments (sec) <9>
TIO	A	12	total dasd accesses <12>
TOU	A	18	time for output (sec) <18>
TPB	A	27	time per block (sec) <27>
TRIB	A	21	trans rate (bits/sec) <21>
TT	A	16	total time (sec) <16>

Number of symbol table entries - 58

```
SL.K=SL.J+DT*(SRI.JK-SRO.JK)                                    L,1
SL=SLN                                                          N,1.1
SLN=0                                                           C,1.2
     SL      - segment level (segments) <1>
     SRI     - segment rate in (seg/sec) <2>
     SRO     - seg rate out (seg/sec) <14>

SRI.KL=CLIP(0,ARI.K,TIME.K,TI.K)                                R,2
     SRI     - segment rate in (seg/sec) <2>
     ARI     - average segment rate in (seg/sec) <3>
     TI      - time for segments (sec) <9>

ARI.K=NS.K/TI.K                                                 A,3
     ARI     - average segment rate in (seg/sec) <3>
     NS      - number of segments (seg) <4>
     TI      - time for segments (sec) <9>

NS.K=(CBO.K+BOV.K+SOV.K)/SLEN                                   A,4
     NS      - number of segments (seg) <4>
     CBO     - comp. Bytes out (bytes) <6>
     BOV     - block overhead (bytes) <7>
     SOV     - segment overhead (bytes) <8>
     SLEN    - segment length (bytes) <24>

SP.K=ARI.K*TIME.K                                               A,5
     SP      - seg processed (seg) <5>
     ARI     - average segment rate in (seg/sec) <3>

CBO.K=BI*CR                                                     A,6
BI=58000                                                        C,6.1
CR=.347                                                         C,6.2
     CBO     - comp. Bytes out (bytes) <6>
     BI      - bytes in (bytes) <6>
     CR      - compression ratio <6>

BOV.K=(BI/BS)*BOVC                                              A,7
BS=940                                                          C,7.1
BOVC=3                                                          C,7.2
     BOV     - block overhead (bytes) <7>
     BI      - bytes in (bytes) <6>
     BS      - block size (bytes) <7>
     BOVC    - block overhead (bytes) <7>

SOV.K=((CBO.K+BOV.K)/SLEN)*SOVC                                 A,8
SOVC=5                                                          C,8.1
     SOV     - segment overhead (bytes) <8>
     CBO     - comp. Bytes out (bytes) <6>
```

```
     BOV    - block overhead (bytes) <7>
     SLEN   - segment length (bytes) <24>
     SOVC   - segment overhead (bytes) <8>

TI.K=CPUNS.K+IONS.K                                          A,9
     TI     - time for segments (sec) <9>
     CPUNS  - cpu time required (sec) <10>
     IONS   - dasd i/o time (sec) <11>



CPUNS.K=(BI*CPUPBI)+(BI*CR*CPUPBO)+CPUOVC                    A,10
CPUPBI=.0000012                                              C,10.1
CPUPBO=.0000117                                              C,10.2
CPUOVC=.22                                                   C,10.3
     CPUNS  - cpu time required (sec) <10>
     BI     - bytes in (bytes) <6>
     CPUPBI - cpu per byte of input (sec) <10>
     CR     - compression ratio <6>
     CPUPBO - cpu per byte of output (sec) <10>
     CPUOVC - cpu overhead (sec) <10>

IONS.K=SDAT*TIO.K                                            A,11
SDAT=0.01                                                    C,11.1
     IONS   - dasd i/o time (sec) <11>
     SDAT   - seq dasd access time (sec) <11>
     TIO    - total dasd accesses <12>

TIO.K=BIN.K+NS.K+ODC                                         A,12
     TIO    - total dasd accesses <12>
     BIN    - blocks read <13>
     NS     - number of segments (seg) <4>
     ODC    - overhead disk accesses <13>

BIN.K=BI/BS                                                  A,13
ODC=24                                                       C,13.1
     BIN    - blocks read <13>
     BI     - bytes in (bytes) <6>
     BS     - block size (bytes) <7>
     ODC    - overhead disk accesses <13>

SRO.KL=CLIP(RO.K,0,TIME.K,TI.K)                              R,14
     SRO    - seg rate out (seg/sec) <14>
     RO     - rate out (seg/sec) <15>
     TI     - time for segments (sec) <9>

RO.K=SWITCH(0,ARO.K,SL.K)                                    A,15
```

```
     RO      - rate out (seg/sec) <15>
     ARO     - avg seg rate out (seg/sec) <17>
     SL      - segment level (segments) <1>

TT.K=TI.K+TOU.K                                          A,16
     TT      - total time (sec) <16>
     TI      - time for segments (sec) <9>
     TOU     - time for output (sec) <18>

ARO.K=NS.K/TOU.K                                         A,17
     ARO     - avg seg rate out (seg/sec) <17>
     NS      - number of segments (seg) <4>
     TOU     - time for output (sec) <18>

TOU.K=SST+BT.K                                           A,18
SST=30                                                   C,18.1
     TOU     - time for output (sec) <18>
     SST     - session start time (sec) <18>
     BT      - bit time (sec) <19>


BT.K=TB.K/TRIB.K                                         A,19
     BT      - bit time (sec) <19>
     TB      - total bits (bits) <20>
     TRIB    - trans rate (bits/sec) <21>

TB.K=NS.K*BYTE*SLEN                                      A,20
BYTE=8                                                   C,20.1
     TB      - total bits (bits) <20>
     NS      - number of segments (seg) <4>
     BYTE    - bits per byte (bits) <20>
     SLEN    - segment length (bytes) <24>

TRIB.K=(BPB.K*PS.K)/TPB.K                                A,21
     TRIB    - trans rate (bits/sec) <21>
     BPB     - bits per block (bits) <22>
     PS      - probability of success <26>
     TPB     - time per block (sec) <27>

BPB.K=BYTE*IB.K                                          A,22
     BPB     - bits per block (bits) <22>
     BYTE    - bits per byte (bits) <20>
     IB      - info block size (bytes) <23>

IB.K=SBS.K-SDOV.K                                        A,23
     IB      - info block size (bytes) <23>
     SBS     - sdlc block size (bytes) <24>
```

```
     SDOV   - sdlc overhead (bytes) <25>

SBS.K=PU2LEN+SDOV.K                                          A,24
PU2LEN=256                                                   C,24.1
SLEN=6144                                                    C,24.2
     SBS    - sdlc block size (bytes) <24>
     PU2LEN - ru size for pu type 2 <24>
     SDOV   - sdlc overhead (bytes) <25>
     SLEN   - segment length (bytes) <24>

SDOV.K=LH+TH+RH+LT                                           A,25
LH=3                                                         C,25.1
TH=6                                                         C,25.2
RH=3                                                         C,25.3
LT=3                                                         C,25.4
     SDOV   - sdlc overhead (bytes) <25>
     LH     - link header (bytes) <25>
     TH     - transmission header (bytes) <25>
     RH     - request header (bytes) <25>
     LT     - link trailer (bytes) <25>

PS.K=1-PF                                                    A,26
PF=.01                                                       C,26.1
     PS     - probability of success <26>
     PF     - probability of failure <26>

TPB.K=BLT.K+IBLT                                             A,27
     TPB    - time per block (sec) <27>
     BLT    - block time (sec) <28>
     IBLT   - inter-block time (sec) <29>

BLT.K=SBS.K/CLS.K                                            A,28
     BLT    - block time (sec) <28>
     SBS    - sdlc block size (bytes) <24>
     CLS    - line speed (bytes/sec) <29>

CLS.K=BLS/BYTE                                               A,29
BLS=2400                                                     C,29.1
IBLT=.4                                                      C,29.2
     CLS    - line speed (bytes/sec) <29>
     BLS    - line speed (bits/sec) <29>
     BYTE   - bits per byte (bits) <20>
     IBLT   - inter-block time (sec) <29>

LENGTH.K=TT.K                                                A,30
     LENGTH - length of simulation <30>
     TT     - total time (sec) <16>
```

```
SAVPER.K=1                                                          A,31
     SAVPER - plot 10 points <31>

SAVE SRO,SL,SRI,TRIB,TI,TOU,TT,CPUNS                                 32
     SRO    - seg rate out (seg/sec) <14>
     SL     - segment level (segments) <1>
     SRI    - segment rate in (seg/sec) <2>
     TRIB   - trans rate (bits/sec) <21>
     TI     - time for segments (sec) <9>
     TOU    - time for output (sec) <18>
     TT     - total time (sec) <16>
     CPUNS  - cpu time required (sec) <10>

SPEC DT=.25/REL_ERR=1E-6/ABS_ERR=1E-6                                 33
```

LIST OF VARIABLES

SYMBOL	T	WHR-CMP	DEFINITION
ABS_ERR	C	33	
ARI	A	3	average segment rate in (seg/sec) <3>
ARO	A	17	avg seg rate out (seg/sec) <17>
BI	C	6.1	bytes in (bytes) <6>
BIN	A	13	blocks read <13>
BLS	C	29.1	line speed (bits/sec) <29>
BLT	A	28	block time (sec) <28>
BOV	A	7	block overhead (bytes) <7>
BOVC	C	7.2	block overhead (bytes) <7>
BPB	A	22	bits per block (bits) <22>
BS	C	7.1	block size (bytes) <7>
BT	A	19	bit time (sec) <19>
BYTE	C	20.1	bits per byte (bits) <20>
CBO	A	6	comp. Bytes out (bytes) <6>
CLS	A	29	line speed (bytes/sec) <29>
CPUNS	A	10	cpu time required (sec) <10>
CPUOVC	C	10.3	cpu overhead (sec) <10>
CPUPBI	C	10.1	cpu per byte of input (sec) <10>
CPUPBO	C	10.2	cpu per byte of output (sec) <10>
CR	C	6.2	compression ratio <6>
DT	C	33	
IB	A	23	info block size (bytes) <23>
IBLT	C	29.2	inter-block time (sec) <29>
IONS	A	11	dasd i/o time (sec) <11>
LENGTH	A	30	length of simulation <30>
LH	C	25.1	link header (bytes) <25>
LT	C	25.4	link trailer (bytes) <25>
NS	A	4	number of segments (seg) <4>
ODC	C	13.1	overhead disk accesses <13>
PF	C	26.1	probability of failure <26>
PS	A	26	probability of success <26>
PU2LEN	C	24.1	ru size for pu type 2 <24>
REL_ERR	C	33	
RH	C	25.3	request header (bytes) <25>
RO	A	15	rate out (seg/sec) <15>
SAVPER	A	31	plot 10 points <31>
SBS	A	24	sdlc block size (bytes) <24>
SDAT	C	11.1	seq dasd access time (sec) <11>
SDOV	A	25	sdlc overhead (bytes) <25>

SL	L	1	segment level (segments) <1>
	N	1.1	
SLEN	C	24.2	segment length (bytes) <24>
SLN	C	1.2	
SOV	A	8	segment overhead (bytes) <8>
SOVC	C	8.1	segment overhead (bytes) <8>
SP	A	5	seg processed (seg) <5>
SRI	R	2	segment rate in (seg/sec) <2>
SRO	R	14	seg rate out (seg/sec) <14>
SST	C	18.1	session start time (sec) <18>
TB	A	20	total bits (bits) <20>
TH	C	25.2	transmission header (bytes) <25>
TI	A	9	time for segments (sec) <9>
TIO	A	12	total dasd accesses <12>
TOU	A	18	time for output (sec) <18>
TPB	A	27	time per block (sec) <27>
TRIB	A	21	trans rate (bits/sec) <21>
TT	A	16	total time (sec) <16>

Number of symbol table entries - 59

APPENDIX C - COMPRESSION TEST ROUTINE

```
         TITLE 'BDCOMPA - ANSI DATA COMPRESSION'
         SPACE 3
*
*      Author:    Gary Chaulklin
*      Function:  This program was written to test the usefulness of
*                 the forthcoming ANSI compression standard.  This is
*                 strictly a test program and has not been fully
*                 tested for production operation.
*********************************************************************
*    INPUT : REG 1 WILL POINT TO AN ADCON LIST WHOSE ENTRIES ARE:    *
*        ADCON1 : WILL BE THE ADDRESS OF THE INPUT DATA              *
*                 TO BE COMPRESSED                                   *
*        ADCON2 : WILL BE AN ADDRESS WHICH POINTS TO                 *
*                 A 4 BYTE FIELD THAT CONTAINS THE LENGTH            *
*                 IN BINARY OF THE INPUT DATA.                       *
*        ADCON3 : WILL BE BEGINNING ADDRESS OF THE OUTPUT            *
*                 AREA TO CONTAIN THE COMPRESSED DATA.               *
*        ADCON4 : WILL BE THE ADDRESS OF A 4 BYTE AREA               *
*                 FOR THIS ROUTINE TO PLACE THE LENGTH               *
*                 OF THE OUTPUT DATA PLACED IN ADCON3                *
*                                                                    *
*    REGISTER USAGE                                                  *
*        R0 - WORKREG                                                *
*        R1 - ADDRESS OF PARM LIST ON ENTRY & WORKREG                *
*        R2 - WORKREG                                                *
*        R3 - WORKREG                                                *
*        R4 - WORKREG                                                *
*        R5 - WORKREG                                                *
*        R6 - WORKREG                                                *
*        R7 - WORKREG                                                *
*        R8 - BASEREG                                                *
*        R9 - CURRENT ADDRESS IN INPUT AREA                          *
*        R10 - USED TO CONTAIN LENGTH OF DATA                        *
*        R11 - CURRENT ADDRESS IN OUTPUT AREA                        *
*        R12 - ADDRESS OF OUTPUT AREA LENGTH FIELD                   *
*        R13 - SAVE AREA REGISTER                                    *
*        R14 - NOT USED EXCEPT ON ENTRY AND EXIT                     *
*        R15 - WORKREG
*********************************************************************
*I N I T I A L I Z A T I O N
*********************************************************************
         EJECT
BDCOMPA   ENTER REG=(8)
* EDIT FOR A VALID PARMLIST
         LA    R12,3               CHECK PARM FOR 4 ADDRESSES
         LR    R11,R1              MOVE START OF PARM LIST TO R11
```

```
AA010    DS    0H
         TM    0(R11),X'80'        ENSURE THAT
         BO    AA050                 THE PARMLIST
         LA    R11,4(,R11)             IS 4 ENTRIES
         BCT   R12,AA010                 LONG
         TM    0(R11),X'80'          IS THE 4TH ADDRESS THE LAST
         BO    AA100               IF YES, BRANCH
AA050    DS    0H
         MVC   AA010,X'0000'       DISABLE FUTURE CALLS
         LA    R2,4                SET RETURN CODE
         B     RETCALL             GO EXIT WITH RC=4
AA100    DS    0H
         EJECT
MAINLINE DS    0H
         LM    R9,R12,0(R1)   LOAD PARM ADDRESSES.
         L     R10,0(R10)     PUT LENGTH OF INPUT DATA IN R10
         ST    R12,OUTLEN               SAVE ADDRESS OF OUTPUT LENGTH
         SR    R12,R12                  LENGTH OF OUTPUT ZERO
         LTR   R10,R10                  TEST INPUT LENGTH
         BNZ   BEGIN
         MVC   AA010,X'0000'       DISABLE FUTURE CALLS
         LA    R2,4                SET RETURN CODE
         B     RETCALL             GO EXIT WITH RC=4
BEGIN    DS    0H
         MVC   LZ_BUF(1),ESC_BYTE
         MVI   LZ_BUF+1,CP_ON
         MVC   LZ_COUNT,TWO
         TM    STATE,CALL_1             FIRST CALL?
         BNO   EOBLOOP                  NO, DON'T RESET VACO
         NI    STATE,FF-CALL_1
         MVC   VACO,BYTEFFP1
         B     EOBLOOP                  GO LOOP UNTIL END OF BLOCK
TERM     DS    0H
**-------------------------------------------------------------------**
** TERM CLEARS STATE,COUNTERS,BUFFERS,BUT NOT TABLES                  **
**-------------------------------------------------------------------**
         NI    STATE,FF-UPDATE          NOT IN UPDATE STATE
         TM    STATE,NEW_OK             NEW CHARACTER TO PROCESS?
         BNO   TERMSR                   NO, GO CHECK FOR SEND RAW
         BAL   R14,ADDTORAW
         NI    STATE,FF-NEW_OK          NO NEW CHARACTER TO PROCESS
TERMSR   DS    0H
         LH    R2,RAWCOUNT              TEST IF GT ZERO
         LTR   R2,R2
         BZ    TERMTM                   IF NOT, TEST MODE
         CLC   RAWCOUNT,LZ_COUNT        IF MORE IN RAW BUF
         BH    TERMSL                   SEND LZ BUF
```

```
         BAL   R14,SEND_RAW
         B     TERMSS                    GO CHECK RESET FLAG
TERMSL   DS    0H
         BAL   R14,OFF_RAW
         BAL   R14,SEND_RAW
         B     TERMSS
TERMTM   DS    0H
         TM    STATE,RAW_MODE            IN RAW MODE NOW?
         BO    TERMSS                    YES, GO SET STATE
         DC    H'0'       ABEND TO SEE HOW THIS HAPPENED
         BAL   R14,SEND_RAW
TERMSS   DS    0H
**-------------------------------------------------------------------**
**  RESET STATE, COUNTERS, BUFFERS                                   **
**-------------------------------------------------------------------**
         MVI   STATE,RAW_MODE            STATE = RAW MODE ONLY
         XC    LZ_BUF,LZ_BUF
         XC    RAW_BUF,RAW_BUF
         XC    RAWCOUNT,RAWCOUNT
         XC    LZ_COUNT,LZ_COUNT
         XC    BITS,BITS                 ZERO BIT COUNTER
         SR    R2,R2                     ZERO RETURN CODE
         B     RETCALL
EOBLOOP  DS    0H                        LOOP UNTIL END OF BLOCK
         TM    STATE,PREFXRAW            NEED TO PREFIX RAW BUFFER?
         BNO   ENDPRAW                   NO, GO END PREFIX RAW
         NI    STATE,FF-PREFXRAW
         BAL   R14,OFF_RAW
ENDPRAW  DS    0H
         TM    STATE,NEW_OK              LEFT OVER CHARACTER?
         BO    SETFIRST                  YES, GO SET FIRST IN STRING
         TM    STATE,RESET_F
         BO    TERM
         LTR   R10,R10
         BZ    TERM                      END OF INPUT
         BCTR  R10,0
         MVC   CH_NEW,0(R9)              ROOT NODE CHARACTER
         LA    R9,1(R9)
SETFIRST DS    0H
         LH    R2,CH_NEWH
         AH    R2,BYTE_00
         STH   R2,NODE                   NODE=CH_NEW+BYTE_00
         XC    DEPTH,DEPTH               STRING DEPTH
         OI    STATE,UPDATE              TABLE TO BE UPDATED
         BAL   R14,ADDTORAW              ADD BYTE TO RAW BUF
DEPTHLOO DS    0H
         MVC   CODE,NODE
```

```
         LH    R2,DEPTH
         LA    R2,1(R2)
         STH   R2,DEPTH                 INCREMENT DEPTH OF STRING
         NI    STATE,FF-NEW_OK
         TM    STATE,RESET_F
         BO    MATCHCOM                 MATCH COMPLETED
         LTR   R10,R10
         BNZ   GETNEXT                  END OF INPUT
         OI    STATE,RESET_F
         B     MATCHCOM
GETNEXT  DS    0H
         BCTR  R10,0
         MVC   CH_NEW,0(R9)             INTERMEDIATE NODE
         LA    R9,1(R9)
         OI    STATE,NEW_OK
         CLC   CHECK,RAWCOUNT           RAW BUF FULL?
         BH    DEEPCHK                  NO, GO CHECK DEPTH
         BAL   R14,SENDCODE             EMTPY LZ BUF
DEEPCHK  DS    0H
         CLC   DEPTHLIM,DEPTH           MAX STRING DEPTH?
         BH    FINDNODE                 NO, GO FIND NODE IN TABLE
         NI    STATE,FF-UPDATE          PROHIBIT TABLE UPDATING
         B     MATCHCOM                 MATCH COMPLETED
FINDNODE DS    0H
         LA    R6,TABLESP
         USING T_NODE,R6                NODE ADDRESSABILITY
         L     R4,BYTE_PTR              ADDRESS OF BYTES
         USING T_BYTE,R4
         LH    R2,NODE
         SLL   R2,3                     NODE OFFSET
         AR    R6,R2                    CURRENT NODE POINTER
         CLC   D_NODE,ZEROS             NO DESCENDANT NODES?
         BE    MATCHCOM                 NO, MATCH COMPLETED
         MVC   NODE,D_NODE              NODE=DOWN(NODE)
MATCHRN  DS    0H                       MATCH RIGHT NODE LOOP
         L     R4,BYTE_PTR              ADDRESS OF BYTES
         LH    R2,NODE
         AR    R4,R2                    CURRENT BYTE POINTER
         SLL   R2,3                     NODE OFFSET
         LA    R6,TABLESP
         AR    R6,R2                    CURRENT NODE POINTER
         CLC   CH_NEW,BYTE
         BH    NXTRNODE                 GO ONE NODE RIGHT
         CLC   CH_NEW,BYTE              CH_NEW DESCENDENT OF NODE?
         BNE   MATCHCOM                 NO, MATCH COMPLETED
         CLC   NODE,NODE_NEW            NEWLY ADDED NODE?
         BE    NXTRNEW                  YES, CAN'T UPDATE CODE
```

```
         BAL   R14,ADDTORAW          GO ADD CHAR TO RAW BUF
         B DEPTHLOO                  GO GET NEXT CHARACTER
NXTRNEW  DS    OH
         NI    STATE,FF-UPDATE       CAN'T UPDATE NEW CODE
         B     MATCHCOM              MATCH COMPLETED
NXTRNODE DS    OH
         CLC   R_NODE,ZEROS
         BE    MATCHCOM              NO MATCH
         MVC   NODE,R_NODE
         B     MATCHRN               CHECK NEXT NODE TO THE RIGHT
MATCHCOM DS    OH
         CLC   CODE_MAX,CODE
         BH    MATCHCJA              JUST ADD CODE TO LZ BUF
         MVC   XCODE,STEP_UP
         BAL   R14,ADD_CODE
         LH    R2,CODE_MAX           MULTIPLY BY 2
         SLL   R2,1
         STH   R2,CODE_MAX
         LH    R2,CODE_LEN
         LA    R2,1(R2)
         STH   R2,CODE_LEN
         MVC   XCODE,CODE
         BAL   R14,ADD_CODE
         BAL   R14,SENDCODE
         B     MATCHCLZ             CHECK LZ BUF
MATCHCJA DS    OH
         MVC   XCODE,CODE
         BAL   R14,ADD_CODE
MATCHCLZ DS    OH
         CLC   CHECK,LZ_COUNT
         BH    TABUP                 UPDATE TABLE
         BAL   R14,SEND_RAW
TABUP    DS    OH
         TM    STATE,UPDATE
         BO    TABUPDUP              CHECK DUPLICATE
         MVC   NODE_NEW,ZEROS
         B     EOBLOOP
TABUPDUP DS    OH
         XC    NODE,NODE
         LA    R5,TABLESP
         USING T_CODE,R5             NODE ADDRESSABILITY
         L     R4,BYTE_PTR           ADDRESS OF BYTES
         LH    R2,CODE
         SLL   R2,3                  NODE OFFSET
         AR    R5,R2                 CURRENT NODE POINTER
         CLC   D_CODE,ZEROS          NO DESCENDANT NODES?
         BE    ADDN                  NO, GO ADD NEW NODE
```

```
         MVC   NODE,D_CODE
TABUPNR  DS    0H
         LH    R2,NODE
         L     R4,BYTE_PTR              ADDRESS OF BYTES
         AR    R4,R2                    CURRENT BYTE POINTER
         SLL   R2,3                     NODE OFFSET
         LA    R6,TABLESP
         AR    R6,R2                    CURRENT NODE POINTER
         CLC   R_NODE,ZEROS
         BE    TABUPCD                  GO CHECK FOR DUPLICATE
         CLC   CH_NEW,BYTE
         BNH   TABUPCD                  CHECK FOR DUPLICATE
         MVC   NODE,R_NODE              GO ONE NODE TO THE RIGHT
         B     TABUPNR
TABUPCD  DS    0H                       CHECK FOR DUPLICATE
         CLC   CH_NEW,BYTE
         BNE   ADDN                     GO ADD NODE
         XC    NODE_NEW,NODE_NEW        NOT NEW NODE
         B     EOBLOOP                  GO GET NEXT CHARACTER
ADDN     DS    0H
         L     R4,BYTE_PTR
         LH    R2,VACO
         AR    R4,R2                    CURRENT BYTE POINTER
         MVC   BYTE,CH_NEW              CREATE LEAF OF CODE
         SLL   R2,3                     NODE OFFSET
         LA    R7,TABLESP
         USING T_VACO,R7
         AR    R7,R2                    VACANT NODE POINTER
         MVC   U_VACO,CODE              UP OF VACO
         LH    R2,CODE
         SLL   R2,3                     NODE OFFSET
         LA    R5,TABLESP
         AR    R5,R2                    CODE POINTER
         CLC   D_CODE,ZEROS             DOWN OF CODE
         BNE   ADDND                    ADD NODE: DESCENDANTS
         MVC   D_CODE,VACO              NO DESCENDANTS
         XC    R_VACO,R_VACO
         XC    L_VACO,L_VACO
         B     TABUPSN                  SET NEW NODE
ADDND    DS    0H
         L     R4,BYTE_PTR
         AH    R4,NODE                  CURRENT BYTE POINTER
         CLC   CH_NEW,BYTE
         DROP  R4
         BNL   ADDNNR                   NO RIGHT NODE OF VACO
         CLC   D_CODE,NODE
         BNE   ADDNM                    ADD NODE TO MIDDLE
```

```
         MVC   D_CODE,VACO             VACO IS THE NEW YOUNGEST
         B     ADDNEC                  FOR EITHER CASE
ADDNM    DS    0H
         LA    R2,TABLESP              POINT TO LEFT OF NODE
         LH    R3,L_NODE
         SLL   R3,3
         AR    R2,R3
         DROP  R6
         USING T_NODE,R2
         MVC   R_NODE,VACO             RIGHT(LEFT(NODE))=VACO
         DROP  R2
         USING T_NODE,R6
ADDNEC   DS    0H          EITHER CASE
         MVC   R_VACO,NODE
         MVC   L_VACO,L_NODE
         MVC   L_NODE,VACO
         B     TABUPSN                 SET NODE_NEW
ADDNNR   DS    0H
         MVC   L_VACO,NODE
         MVC   R_VACO,ZEROS
         MVC   R_NODE,VACO
TABUPSN  DS    0H
         MVC   NODE_NEW,VACO
TRIMLOOP DS    0H                      TRIM LEAF LOOP
         LH    R2,VACO
         LA    R2,1(R2)
         STH   R2,VACO                 INCREMENT VACO
         CLC   VACO,CODE_LIM
         BL    TRIMCD                  CHECK DOWN
         MVC   VACO,BYTEFFP1           BYTE_FF +1
TRIMCD   DS    0H
         LA    R7,TABLESP
         LH    R2,VACO
         SLL   R2,3
         AR    R7,R2
         CLC   D_VACO,ZEROS
         BNE   TRIMLOOP                TRIM LEAF LOOP
         CLC   L_VACO,ZEROS
         BNE   TRIMNY                  NOT YOUNGEST
         LA    R2,TABLESP              POINT TO UP OF VACO
         LH    R3,U_VACO
         SLL   R3,3
         AR    R2,R3
         LH    R3,R_VACO
         DROP  R7
         USING T_VACO,R2
         STH   R3,D_VACO
```

```
         DROP  R2
         USING T_VAC0,R7
         CLC   R_VAC0,ZEROS
         BE    EOBLOOP
         LA    R2,TABLESP              POINT TO RIGHT OF VAC0
         LH    R3,R_VAC0
         SLL   R3,3
         AR    R2,R3
         DROP  R7
         USING T_VAC0,R2
         MVC   L_VAC0,ZEROS
         DROP  R2
         USING T_VAC0,R7
         B     EOBLOOP
TRIMNY   DS    0H
         CLC   R_VAC0,ZEROS
         BE    TRIMSR
         LA    R2,TABLESP              POINT TO RIGHT OF VAC0
         LH    R3,R_VAC0
         SLL   R3,3
         AR    R2,R3
         LH    R3,L_VAC0
         DROP  R7
         USING T_VAC0,R2
         STH   R3,L_VAC0
         DROP  R2
         USING T_VAC0,R7
TRIMSR   DS    0H
         LA    R2,TABLESP              POINT TO LEFT OF VAC0
         LH    R3,L_VAC0
         SLL   R3,3
         AR    R2,R3
         LH    R3,R_VAC0
         DROP  R7
         USING T_VAC0,R2
         STH   R3,R_VAC0
         DROP  R2
         USING T_VAC0,R7
         B     EOBLOOP
ADDTORAW DS    0H
         LH    R3,RAWCOUNT
         LA    R2,RAW_BUF              ADD CH_NEW TO RAW BUF
         AR    R2,R3
         MVC   0(1,R2),CH_NEW
         LA    R3,1(R3)
         STH   R3,RAWCOUNT
         CLC   ESC_BYTE,CH_NEW
```

```
         BNER  R14                      RETURN
         MVI   1(R2),LV_IN              LEAVE IN COMMAND
         LA    R3,1(R3)
         STH   R3,RAWCOUNT
         LH    R2,ESC_H
         AH    R2,ESC_UP
         STC   R2,ESC_BYTE              ADD ESC_UP (MODULUS 256)
         BR    R14
SEND_RAW DS    0H
         OI    STATE,RAW_MODE
         LH    R2,RAWCOUNT
         AR    R12,R2                   ADD RAWCOUNT TO OUTPUT COUNT
         BCTR  R2,0
         EX    R2,SENDRM                MOVE RAW BUF TO OUTPUT
         AH    R11,RAWCOUNT             ADD RAWCOUNT TO OUTPUT ADDRESS
         XC    RAWCOUNT,RAWCOUNT
         XC    RAW_BUF,RAW_BUF
         XC    LZ_BUF,LZ_BUF
         MVC   LZ_BUF(1),ESC_BYTE       PREFIX LZ BUF
         MVI   LZ_BUF+1,CP_ON
         MVC   LZ_COUNT,TWO
         XC    BITS,BITS
         BR    R14
SENDRM   MVC   0(0,R11),RAW_BUF         EXECUTED
SENDCODE DS    0H
         NI    STATE,FF-RAW_MODE
         LH    R2,LZ_COUNT
         AR    R12,R2                   ADD LZ_COUNT TO OUTPUT COUNT
         BCTR  R2,0
         EX    R2,SENDCM
         AH    R11,LZ_COUNT             UPDATE OUTPUT BUFFER POINTER
         LH    R2,LZ_COUNT
         LA    R3,LZ_BUF(R2)            POINT TO 'BITS' IN LZ_BUF
         SR    R2,R2
         IC    R2,0(R3)
         XC    LZ_BUF,LZ_BUF
         LH    R3,BITS
         LTR   R3,R3
         BZ    SENDCNB                  NO BITS
         STC   R2,LZ_BUF
SENDCNB  DS    0H
         XC    RAW_BUF,RAW_BUF
         XC    LZ_COUNT,LZ_COUNT
         XC    RAWCOUNT,RAWCOUNT
         OI    STATE,PREFXRAW
         BR    R14
SENDCM   MVC   0(0,R11),LZ_BUF          EXECUTED
```

```
OFF_RAW  DS    OH
         LA    R2,LZ_BUF
         LA    R4,RAW_BUF
         LH    R3,LZ_COUNT
         LA    R3,1(R3)                  BITS GET MOVED ALSO
         LH    R5,MODEBUFH               ZERO PAD REST
         MVCL  R4,R2                     MOVE LZ BUF TO RAW BUF
         LA    R3,RAW_BUF
         MVC   RAWCOUNT,LZ_COUNT
         LH    R2,RAWCOUNT
         AR    R3,R2                     POINT TO THE BYTE TO BE UPDATED
         AH    R2,TWO
         STH   R2,RAWCOUNT
         SR    R2,R2
         IC    R2,0(R3)
         LH    R4,CP_OFF
         LH    R5,BITS
         LTR   R5,R5
         BZ    OFFRNS                    NO SHIFT
OFFRS    DS    OH
         SLL   R4,1
         BCT   R5,OFFRS
         OR    R4,R2                     R4='BITS':CP_OFF
OFFRNS   DS    OH
         STC   R4,0(R3)                  UPDATE FIRST BYTE
         STCM  R4,B'0010',1(R3)
         STCM  R4,B'0100',2(R3)
         LH    R2,BITS
         AH    R2,CODE_LEN
         CH    R2,SIXTEEN
         BLR   R14
         LH    R2,RAWCOUNT
         AH    R2,ONE
         STH   R2,RAWCOUNT
         BR    R14
ADD_CODE DS    OH
**-------------------------------------------------------------------**
**  ADD CODE TO LZ_BUF                                               **
**-------------------------------------------------------------------**
         LA    R3,LZ_BUF
         LH    R2,LZ_COUNT
         AR    R3,R2                     POINT TO THE BYTE TO BE UPDATED
         LA    R2,1(R2)
         STH   R2,LZ_COUNT
         SR    R2,R2
         IC    R2,0(R3)
         LH    R4,XCODE
```

```
         LH    R5,BITS
         LTR   R5,R5
         BZ    ADDCNS                    NO SHIFT
ADDCS    DS    0H
         SLL   R4,1
         BCT   R5,ADDCS
         OR    R4,R2                     R4='BITS':XCODE
ADDCNS   DS    0H
         STC   R4,0(R3)                  UPDATE FIRST BYTE
         STCM  R4,B'0010',1(R3)
         STCM  R4,B'0100',2(R3)
         LH    R2,BITS
         AH    R2,CODE_LEN
         CH    R2,SIXTEEN
         BL    ADDCBP1
         SH    R2,SIXTEEN
         STH   R2,BITS                   BITS += CODE_LEN - 16
         LH    R2,LZ_COUNT
         AH    R2,ONE
         STH   R2,LZ_COUNT
         BR    R14
ADDCBP1  DS    0H
         SH    R2,EIGHT
         STH   R2,BITS                   BITS += CODE_LEN - 8
         BR    R14
**-------------------------------------------------------------------**
**   RETURN TO CALLER                                                **
**-------------------------------------------------------------------**
RETCALL  DS    0H
         L     R3,OUTLEN
         ST    R12,0(R3)                 PASS OUTPUT LENGTH TO CALLER
         EXIT  RC=(R2)
CHKMODE  EQU   64                        BYTES IN BUFFERS BEFORE CHECK
MODEBUF  EQU   68                        OUTPUT BUFFER LENGTH
**-------------------------------------------------------------------**
** COMMANDS: THE THREE ANSI DEFINED COMMANDS ARE USED ONLY IN        **
**           TRANSPARENT MODE AND ARE DEFINED AS FOLLOWS             **
**           1. TO GO TO COMPRESSED MODE THE SEQUENCE "ESC_BYTE      **
**              CP_ON" IS OUTPUT.                                    **
**           2. IF THE ESC_BYTE IS FOUND IN THE DATA, IT MUST BE     **
**              FOLLOWED BY "LV_IN".                                 **
**           3. IF IT HAS BEEN DETERMINED THAT IT WOULD BE           **
**              BENEFICIAL TO RESET THE COMPRESSION TABLE            **
**              A "RESET" COMMAND IS OUTPUT.                         **
**-------------------------------------------------------------------**
CP_ON    EQU   0                         COMPRESSION ON COMMAND
LV_IN    EQU   1                         LEAVE ESC BYTE IN DATA
```

```
RESET    EQU   2                          RESET ALL PARAMETERS AND TABLE
STATE    DC    AL1(RAW_MODE+CALL_1)
PREFXRAW EQU   1
UPDATE   EQU   2
NEW_OK   EQU   4
RAW_MODE EQU   8
RESET_F  EQU   16
CALL_1   EQU   32
FF       EQU   255
ESC_H    DS    0H                         ESCAPE BYTE HALFWORD STRUCTURE
         DC    X'00'                      FIRST BYTE ALWAYS ZERO
ESC_BYTE DC    X'00'                      ACTUAL ESC_BYTE INITIALLY ZERO
CH_NEWH  DS    0H
         DC    X'00'
CH_NEW   DC    X'00'
CP_OFF   DC    H'0'                 CODEWORD FOR SWITCH TO TRANSPARENT
*        DC    H'1'                 RESERVED FOR FUTURE USE
STEP_UP  DC    H'2'                 CODEWORD TO INCREMENT CODE LENGTH
CC_MAXE  EQU   2                    MAXIMUM DEFINED CONTROL CODE
CODEMAXE EQU   2048                 MAX # OF CODEWORDS
BITMAXE  EQU   11                   2 TO THE BITMAXE = CODEMAXE
CC_MAX   DC    AL2(CC_MAXE)         LARGEST CONTROL CODEWORD
ESC_UP   DC    H'51'                ESC_BYTE INCREASES BY THIS VALUE
ZEROS    DC    H'0'
ONE      DC    H'1'
TWO      DC    H'2'
EIGHT    DC    H'8'
SIXTEEN  DC    H'16'
VACO     DS    H
CODE     DS    H
CODE_MAX DC    H'512'              CURRENT MAX CODE FOR BIT LENGTH
XCODE    DS    H
BYTE_FF  DC    AL2(CC_MAXE+256)
BYTEFFP1 DC    AL2(CC_MAXE+256+1)
BYTE_00  DC    AL2(CC_MAXE+1)
DEPTH    DC    H'0'
NODE_NEW DC    H'0'
NODE     DC    H'0'
CODE_LEN DC    H'9'                 STARTS AT 9 AND GOES TO BITMAXE
CODE_LIM DC    AL2(CODEMAXE)       2 TO THE BITS_MAX = MAX # CODEWORDS
BITS_MAX DC    AL2(BITMAXE)         BITS IN A CODEWORD
DEPTHLIM DC    H'32'                MAXIMUM STRING DEPTH ***VARIABLE***
CHECK    DC    AL2(CHKMODE)
RAWCOUNT DC    H'0'
MODEBUFH DC    AL2(MODEBUF)
LZ_COUNT DC    H'0'
BITS     DC    H'0'
```

```
OUTLEN    DC    F'0'                        ADDRESS OF OUTPUT LENGTH
RAW_BUF   DC    (MODEBUF)XL1'00'
LZ_BUF    DC    (MODEBUF)XL1'00'
T_CODE    DSECT
U_CODE    DS    H
D_CODE    DS    H
L_CODE    DS    H
R_CODE    DS    H
TABLELEN  EQU   *-T_CODE
T_VACO    DSECT
U_VACO    DS    H
D_VACO    DS    H
L_VACO    DS    H
R_VACO    DS    H
T_NODE    DSECT
U_NODE    DS    H
D_NODE    DS    H
L_NODE    DS    H
R_NODE    DS    H
T_BYTE    DSECT
BYTE      DS    CL1
BDCOMPA   CSECT
TABLESPL  DC    AL4(CODEMAXE*TABLELEN)
BYTEL     DC    AL4(CODEMAXE)
BYTE_PTR  DC    A(BYTESP)
TABLESP   DC    (CODEMAXE*TABLELEN)X'00'
BYTESP    DC    (CODEMAXE)X'00'
          END   BDCOMPA
```