UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER DTNSRDC-81/067	2. GOVT ACCESSION NO. AD-A104 579	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) A BACK-END DATA BASE MANAGEMENT EXPERIMENT		5. TYPE OF REPORT & PERIOD COVERED Final
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Michael A. Wallace		8. CONTRACT OR GRANT NUMBER(s) [illegible]
9. PERFORMING ORGANIZATION NAME AND ADDRESS David W. Taylor Naval Ship Research and Development Center Bethesda, Maryland 20084		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Element 62760N ~~Task Area~~ TF53531019 Work Unit 1828-003
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Supply Systems Command (SUP0431) Washington, D.C. 20376		12. REPORT DATE September 1981
		13. NUMBER OF PAGES 74
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*)		15. SECURITY CLASS. (*of this report*) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

Accession For		
NTIS GRA&I	✓	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Distribution/		
Availability Codes		
Dist	Avail and/or Special	
A		

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

ADP Security
Back-End
Data Base Management

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

A method of testing and validating a back-end data base management system is described. The results of the test are analyzed and recommendations for future systems are made. The back-end concept proved viable and future work is proposed.

TABLE OF CONTENTS

Page

LIST OF FIGURES iv

LIST OF TABLES v

LIST OF ABBREVIATIONS vi

ABSTRACT 1

ADMINISTRATIVE INFORMATION 1

INTRODUCTION 1

BACKGROUND 3

PROTOCOL STANDARDS 6

LINK LEVEL PROTOCOLS 6

COMMUNICATION PROTOCOLS 8

SECURITY 13

SYSTEM DESCRIPTION 16

GENERAL FLOW 17

SYSTEM SOFTWARE COMPONENTS 18

BACK-END SYSTEM LOGIC 24

CONNECTIVITY AND ROUTING 24

MESSAGE SYSTEM COMPONENTS 26

IMPLEMENTATION ON THE PDP 11/70 28

TWIN 34

TEST PROGRAMS 35

PERFORMANCE MEASURE METHODOLOGY 35

MESSAGE TRANSMISSION/RECEPTION LOGGING 36

TIME TAGGING STRATEGY 37

TESTING STRATEGY 38

TEST ENVIRONMENT 39

PERFORMANCE MEASUREMENT RESULTS 39

PDP MESSAGE LOGGING 40

Page

PDP MESSAGE TRACE 42
PDP SYSTEM HISTOGRAM 42
IBM LINE ACTIVITY RECORDER 46
IBM SYSTEM STATISTICS 46

ANALYSIS OF RESULTS 46
TIMING ANALYSIS 46
CODE EXAMINATION 49
PROBLEM AREAS 51
BACK-END VERSUS STAND-ALONE 52
BENEFITS COMPARISON 52

CONCLUSIONS AND RECOMMENDATIONS 53
TESTING FUNCTIONS 54
BACK-END DBMS FUNCTIONS 54
SYSTEMS FUNCTIONS AND CHARACTERISTICS 55
USER ENVIRONMENT 56
ADDITIONAL BENEFITS 58
APPLICATION PROGRAM REQUIREMENTS 58
BACK-END SYSTEM DESIGN 58
SUMMARY 59

ACKNOWLEDGMENTS 64

REFERENCES 65

LIST OF FIGURES

1 - DBMS Host Implementation 1

2 - Host with Back-End Processor and Data Base 2

3 - Data Link Transfer 7

4 - Protocol Peer Levels 9

5 - Digital Network Architecture (DNA) 10

Page

6 - System Network Architecture (SNA). 11

7 - ANSI/ISO Open Systems Reference Model. 12

8 - Host/Back-End Software Distribution. 18

9 - Packet Format. 25

10 - Test Timing Zones. 36

11 - Sample PDP Message Logging 41

12 - Sample PDP Message Trace 42

13 - Back-End with Program Development Machine. 57

14 - Singly Connected Back-End System Configuration 60

15 - Multiple Host Back-End System Configuration. 61

16 - Single Host Multiple Back-End System Configuration 62

17 - Multiply-Connected Back-End System Configuration 63

LIST OF TABLES

1 - Definition of Tested Verbs 40

2 - Selected PDP Message Logging 41

3 - PDP Message Logging Averages 41

4 - Selected Average Queue Times 43

5 - Average Queue Times. 43

6 - Selected Queue Process Times 44

7 - Overall Percentage of Queue Process Time 45

8 - LAR Results. 47

9 - IBM Statistics for Back-End. 48

10 - IBM Statistics for Stand-Alone 48

LIST OF ABBREVIATIONS

ADCCP	Advanced Data Communication Control Procedure
ANSI	American National Standards Institute
ARQ	Automatic Repeat Request
ASCII	American Standard Code for Information Interchange
BENRC	Back-End Network Resource Controller
BET	Back-End Task
BNA	Burrough's Network Architecture
BSC	Binary Synchronous Communication
BTL	Bell Telephone Laboratories
CCITT	International Telephone and Telegraph Consultation Committee
CPU	Central Process Unit
CRC	Cyclic Redundancy Check
DAP	Data Access Protocol
DBA	Data Base Administrator
DBCS	Data Base Control System
DBMS	Data Base Management System
DBO	Data Base Operator
DDCMP	Digital Data Communication Message Protocol
DDL	Data Description Language
DEC	Digital Equipment Corporation
DLC	Data Link Control
DMA	Direct Memory Access
DMCC	Device/Media Control Language
DML	Data Manipulation Language
DNA	Digital Network Architecture
DQ	High Speed Communications Line Interface (1 mega band)
DU	Low Speed Communications Line Interface (9600 band)
FEP	Front-End Processor
HDLC	High Level Data Link Control
HINT	Host Interface Module
HNRC	Host Network Resource Controller
IPF	Incorrect Page Fixer
ISO	International Standards Organization
LAR	Line Activity Recorder
MS	Message System
MSO	Message System Operator

NARDAC	Navy Regional Data Automation Center
NAU	Network Addressable Unit
NSP	Network Services Protocol
PDM	Program Development Machine
PP	Protocol Processor
QIO	Queue Input/Output
SDLC	Synchronous Data Link Control
SGA	Sharable Global Area
SNA	Systems Network Architecture
UICP	Uniform Inventory Control Program
USACSC	U.S. Army Computer Systems Command

ABSTRACT

A method of testing and validating a back-end data base management system is described. The results of the test are analyzed and recommendations for future systems are made. The back-end concept proved viable and future work is proposed.

ADMINISTRATIVE INFORMATION

The tests described in this report were performed at the David W. Taylor Naval Ship Research and Development Center (DTNSRDC) in cooperation with the Naval Regiona Data Automation Command. The work was sponsored by the Naval Supply Systems Command (SUP0431C) under Program Element 62760N, Task Area TF60531019, Sponsor Order Number 09007, Work Unit 1828-003.

INTRODUCTION

The advent of data base management systems (DBMS) has made available a powerful tool for both managers and programmers. A DBMS will reduce programming effort, avoi duplication of data, provide a uniform means of accessing data, and improve data control. The way a large DBMS is usually implemented is shown in Figure 1.

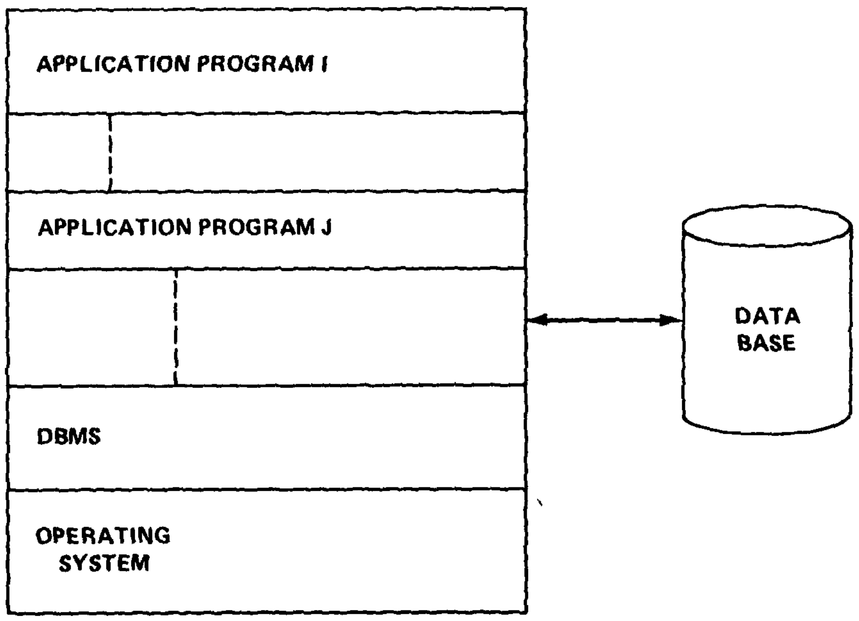


Figure 1 - DBMS Host Implementation

As the capabilities of a DBMS increase, so do the demands on system resources, main memory, disk space, and central processor unit (CPU) time. The more general purpose the DBMS, the more widespread will be its application and the more demanding of system resources it will become. When the DBMS demands become a significant percentage of system resources, some action must be taken. The usual answer has been to purchase a more powerful machine. Electronic advances in the last several years have, however, made other solutions feasible. One such solution, the one this paper addresses, is the back-end processor. This technique, shown in Figure 2, transfers all data management functions to a separate machine (usually a general purpose mini-computer) and requires the application program running on the host machine to communicate with the DBMS over a communication channel such as a telephone line. This communication process is transparent to the program so that existing programs, already using the DBMS, do not have to be rewritten.

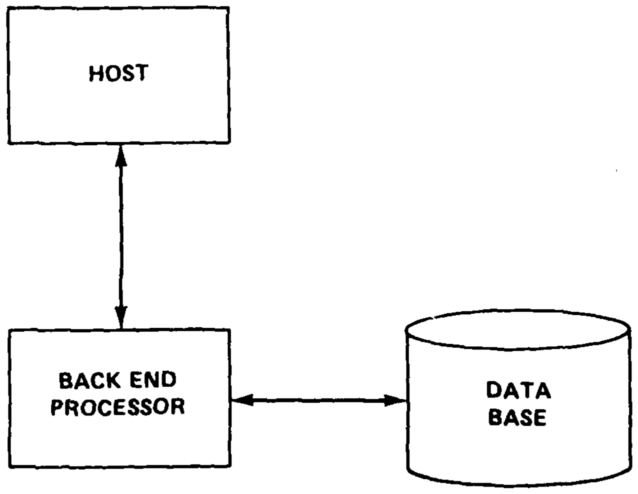


Figure 2 - Host with Back-End Processor and Data Base

A test system was developed under contract to demonstrate the feasibility of the back-end processor approach. This system was installed on the PDP 11/70 (back-end processor) at DTNSRDC and on the IBM 370 (host) at NARDAC. DTNSRDC, with the operational assistance of NARDAC, tested the system for validity and performance. This test effort, sponsored primarily by NAVSUP, was to determine whether the originally perceived benefits could be realized and the disadvantages minimized. The perceived benefits were:

- reduced disk space
- reduced main memory for application programs
- reduced CPU time for application programs
- increased data security
- increased host utilization by linking several hosts to one back-end processor
- independence of the DBMS from the host processor
- extend the useful life cycle of the host by freeing system resources

The known disadvantages are:

- an additional computer to operate and maintain
- a communications link to maintain
- a development, implementation, and maintenance software effort on two machines

The system implementation, testing methodology, analysis of results, and conclusions/recommendations are discussed in this report.

BACKGROUND

The concept of a back-end data base management system (DBMS) was first widely publicized in October 1974 in a paper by Bell Telephone Laboratories (BTL) in the Communications of the ACM. The Department of Defense, U.S. Army Computer Systems Command (USACSC), and a vendor of DBMS software conducted studies of the concept and concluded that a back-end DBMS was feasible and desirable. DTNSRDC evaluated these studies and determined that the back-end DBMS was of potential value in the Naval Supply Systems Command's (NAVSUP's) Uniform Inventory Control Program (UICP). A Joint Working Group, composed of representatives of DTNSRDC, the Department of Defense, Naval Material Command Support Activity, and the USACSC was formed to provide partial funding for the production of a prototype back-end DBMS based on a commercially available CODASYL DBMS. DTNSRDC funding was provided by NAVSUP.

The general objectives of the Joint Working Group were:

1. To develop a high-speed, low-cost alternative to the conventional DBMS hardware and software.

2. To develop a higher degree of security and reliability than could be economically provided by conventional DBMS hardware and software.

3. To speed up what was considered to be the inevitable development of a commercially viable back-end DBMS. (The BTL system was primarily a research effort.) The Joint Working Group felt that only a widely used commercial product would have a sufficiently large user base to support the development and maintenance of highly efficient and reliable software.

4. To obtain, at the earliest possible time, a back-end DBMS for evaluation against the needs of the different representatives. Time was particularly critical to DTNSRDC and its NAVSUP sponsor, because UICP was soon to be converted to new hardware and software, and the possibility of the back-end DBMS as a supplement or replacement for the UICP DBMS seemed very attractive.

5. To obtain, at a very low cost, what might prove to be a state-of-the-art CODASYL DBMS. Of course, there was no assurance that the concept would be workable, given the constraint of basing the development of a commercially available DBMS. Clearly, the back-end would be most efficient and effective if special operating system software, and possibly special hardware, were used, but this would have greatly increased the cost and time of development.

6. To develop a DBMS that could reside on a variety of minicomputers and interface with multiple heterogeneous hosts. The prototype development was directed toward a single minicomputer and either a CDC 6600 or IBM 360/370 host, but the long-range objective was to provide a high degree of flexibility in system hardware.

The following specific objectives were included in the back-end DBMS contract:

1. Minimum Interface Requirements

a. All software modules will be written in a high level language such as FORTRAN or similar language to provide maximum transferability.

b. The delivered back-end DBMS will initially be capable of executing on both a 16-bit minicomputer (e.g., PDP 11/70) and a 32-bit mini (e.g., Interdate 832).

c. As a minimum, interfaces to the IBM 360/370 and CDC 6600 will be provided as a part of the deliverable software.

d. Clean, well defined interface specifications will be provided that will enable the user to develop additional host interfaces.

e. The back-end system must be able to interface with a variety of cummunication lines and line spreads ranging from 300 bps to 50K bps.

f. The selected back-end hardware must be able to interface with a variety of disk systems including 2314, 3330, or CDC 841 and equivalent devices.

g. Interface with the host must allow for a variety of programming languages.

2. Software Requirements

a. The data base management system must adhere to CODASYL specifications.

b. The DBMS implementation must support rapid Boolean operations on indices, such as pointer arrays, and must provide for subschemas.

c. The system must provide for dynamic space allocation and reutilization (physical restructuring) without taking the system down.

d. The system must provide well defined and effective back up and restart capability, as well as audit trails, without taking the system down.

e. The system will provide for logical data base keys.

f. It must allow for concurrent access and update with locks on a record occurrence basis.

g. It must provide for security on a record type level; i.e., access is restricted to individual record types.

3. General Performance Requirements

a. The back-end system must provide performance comparable to that obtainable by running the DBMS on the host.

b. Performance monitoring capabilities must be provided to the Data Base Administrator (DBA).

c. The DBA must be provided with the ability to tune the performance of the system by changing such parameters as page size, page density, locality of reference, or the number of active users versus the number of active commands.

d. The DBA will have available multiple strategies for data base design so that the system can be biased to favor update, retrieval, addition, or deletion.

e. The DBA must be provided with adequate integrity checking features to guard against security breach attempts, system failures, etc. The DBA must have the ability to communicate directly with the back-end through a system console.

PROTOCOL STANDARDS

The decision on which link level protocol to use for the back-end was made by default; binary synchronous communication (BSC) was chosen because it already ran on the IBM host system. Future implementations will, however, be able to choose from any of the link level and high level standard protocols discussed in this section.

A protocol is a set of communication procedures and conventions that ensure the orderly exchange of data. A protocol may be defined either by publishing the rules, such as is done by the standards organizations ANSI, ISO, and CCITT, or by implementation, as was done by the IBM and Digital Equipment Corp. (DEC). Protocols are divided here into link level protocols and communication (higher level) protocols. The protocols govern the computer-to-computer synchronous trnasmission of data over communications channels.

LINK LEVEL PROTOCOLS

The link level protocol is the lowest level of software and is responsible for providing an "essentially" error-free line. The link level must control the data link, shown in Figure 3. The basic functions are error detection and correction, startup, sequencing, transparency, and synchronization (bit, byte, and message).

Only three data link control protocols are recognized as "standards"; they are:

- BSC (Binary Synchronous Communication)
- DDCMP (Digital Data Communication Message Protocol)
- ADCCP (Advanced Data Communication Control Procedure)

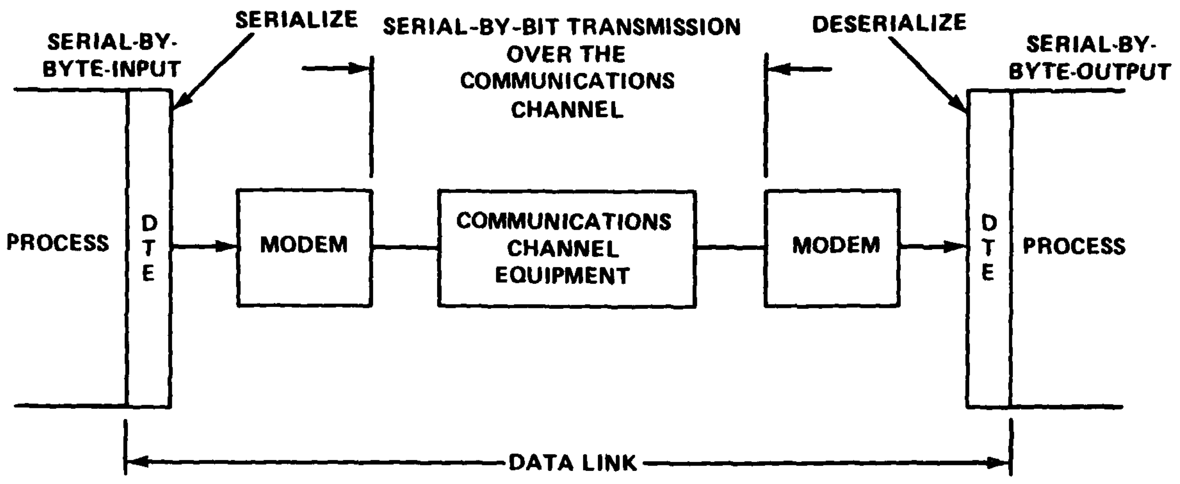


Figure 3 - Data Link Transfer

BSC

BSC was developed by IBM (about 1968) and became, with minor improvements, ISO's Basic Mode Control Procedure (about 1972). BSC is a half-duplex character oriented protocol designed for point-to-point and multipoint operation in a batch environment. All control information is ASCII. BSC is a stop-and-wait automatic repeat request (ARQ) system and may have only one outstanding message (each message must be acknowledged).

BSC was designed primarily for use with unattended batch terminals handling lengthy transmissions and it has performed well in that role. In the interactive environment BSC has been replaced by the byte and bit oriented protocols.

DDCMP

DDCMP, developed by Digital Equipment Corporation, is a byte oriented protocol. The text length is defined by a character count instead of by a delimiting character(s) (BSC) or flag (ADCCP). DDCMP operates in full-duplex and half-duplex modes and in point-to-point and multipoint configurations over serial or parallel links using synchronous or a synchronous communication. It uses go-back-N ARQ and may have up to 255 unacknowledged messages. DDCMP is easily implemented since the data field does not have to be scanned (as required by BSC and ADCCP) but is simply counted. DDCMP requires no special hardware for implementation.

ADCCP

ADCCP defines a method of data link control in terms of the various combinations of allowed commands from a primary of combined station and permissible responses from a secondary or combined station. In ADCCP all transmissions are in frames and each frame is founded by a unique flag sequence (01111110). The uniqueness of this flag is ensured by a process known as bit stuffing (inserting a zero bit after five consecutive one bits) and requires special hardware to implement. ADCCP also uses go-back-N ARQ. Synchronous Data Link Control (SDLC), developed by IBM about 1974, was modified slightly to become ANSI's ADCCP and ISO's high level data link control (HDLC).

All three protocols provide for error detection by use of cyclic redundancy check (CRC), polynomials and error correction by retransmission. BSC is an old, complex, and not highly efficient half-duplex protocol. It is still in use due to its early and widespread acceptance. DDCMP is a simple, powerful, and efficient protocol (10 bytes of overhead per message) that is easy to implement and requires no special hardware. ADCCP is a complex, powerful, and very efficient protocol (overhead is 4 bytes, flags, and bit stuffing per message) that requires special hardware because of the bit stuffing. A subset of ADCCP will be implemented by many computer manufacturers and used as the basis of their network architecture, such as IBM's Systems Netork Architecture (SNA) and Burroughs' Network Architecture (BNA).

COMMUNICATION PROTOCOLS

Communication protocols allow user applications in various parts of a distributed network to communicate conveniently. These protocols are usually divided into peer layer levels as shown in Figure 4. The number of levels and the function of each is determined by their degree of distribution and their management of resources.

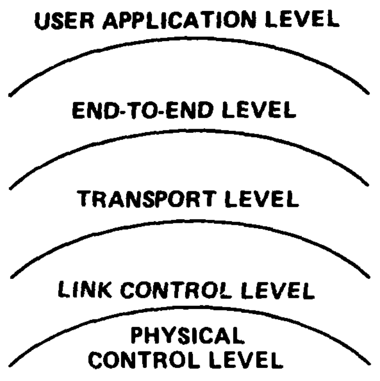


Figure 4 - Protocol Peer Levels

Three examples of network architectures will be described to show the functionality and similarity of communication protocols. These sample systems are:

- DNA (Digital Network Architecture) - Digital Equipment Corporation
- SNA (Systems Network Arthitecture) - IBM
- ANSI/ISO Reference Model of Open Systems Architecture

DNA

DNA is composed of a set of network protocols; DDCMP, network services protocol (NSP), and data access protocol (DAP). These protocols have the following characteristics:

- DDCMP - handles link control and error detection/correction
- NSP - provides process-to-process communication, routes messages between systems, routes messages within systems, creates dynamic logical links, and is the network management protocol.
- DAP - provides network data compatibility, remote file operations, and remote device operations.

Figure 5 shows two systems linked using DNA. The NSP portion of DNA is functionally equivalent to the Message and Packet subsystems of the Message System.

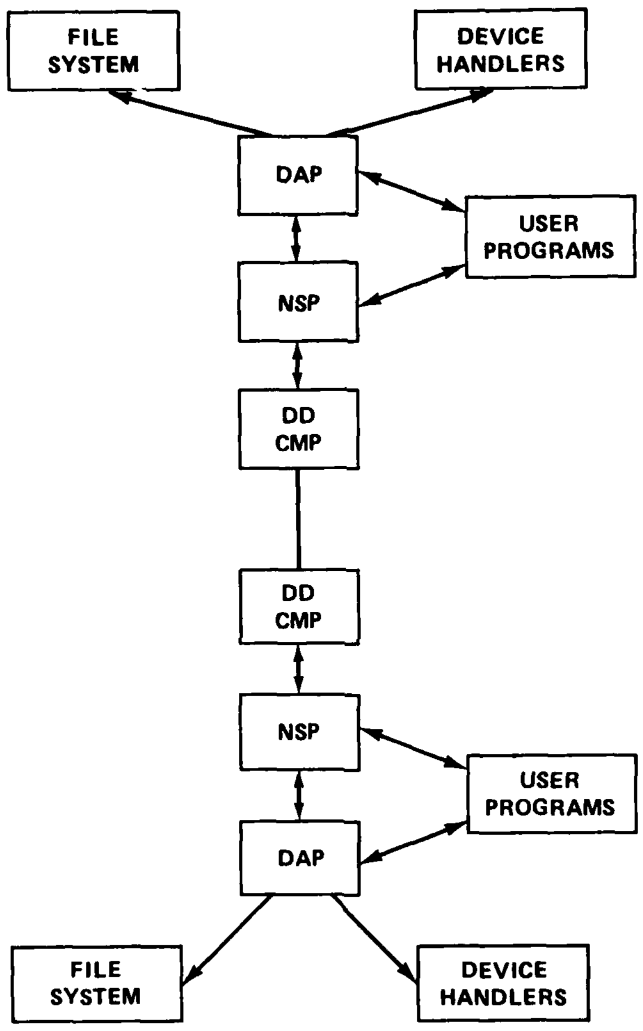


Figure 5 - Digital Network Architecture (DNA)

SNA

Functions in an SNA network are structured into a layered hierarchy. These layers are (1) data link control, (2) path control, (3) transmission control, (4) data flow control, and (5) Network Addressable Units (NAU) services. Each layer is reflected in a protocol implementing the functions of that layer. The data link control layer handles link control and error detection/correction. The path control layer handles message routing. The transmission control layer coordinates a session, or channel, and manages the flow of data on that session. The data flow control layer controls the flow and integrity of data to/from the user. The NAU is the user interface and includes the function managers. This hierarchical network is shown in Figure 6.

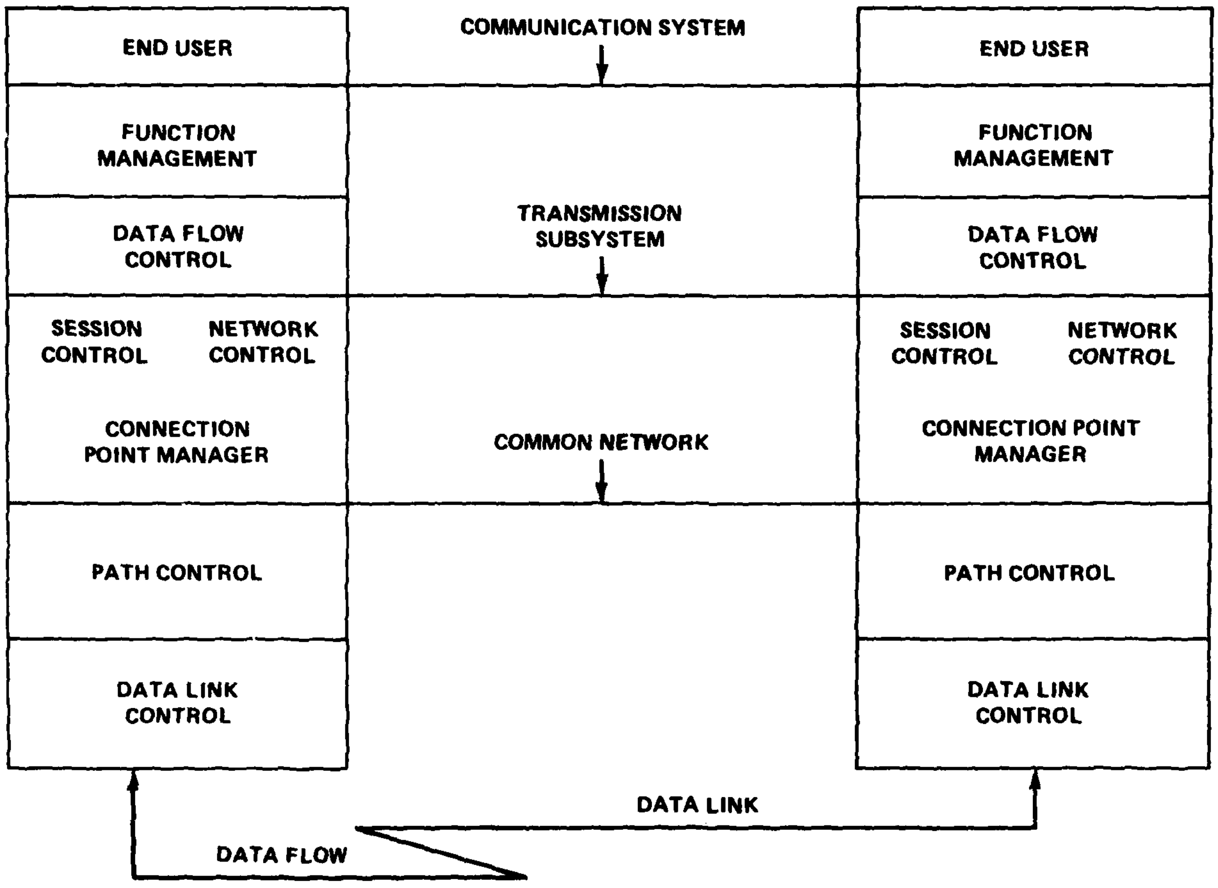


Figure 6 - System Network Architecture (SNA)

ISO

The ANSI/ISO Reference Model of Open Systems Architecture is a hierarchy of seven peer level protocols as shown in Figure 7. The physical layer, link layer, network layer, and transport layer are responsible for moving data from one place to another and are known as the "providers of transport service." The session layer, presentation layer, and application layer are responsible for generating and processing the data and are known as "users of transport service." The main reason the architecture has so many levels is to divide the implementation into clearly defined areas.

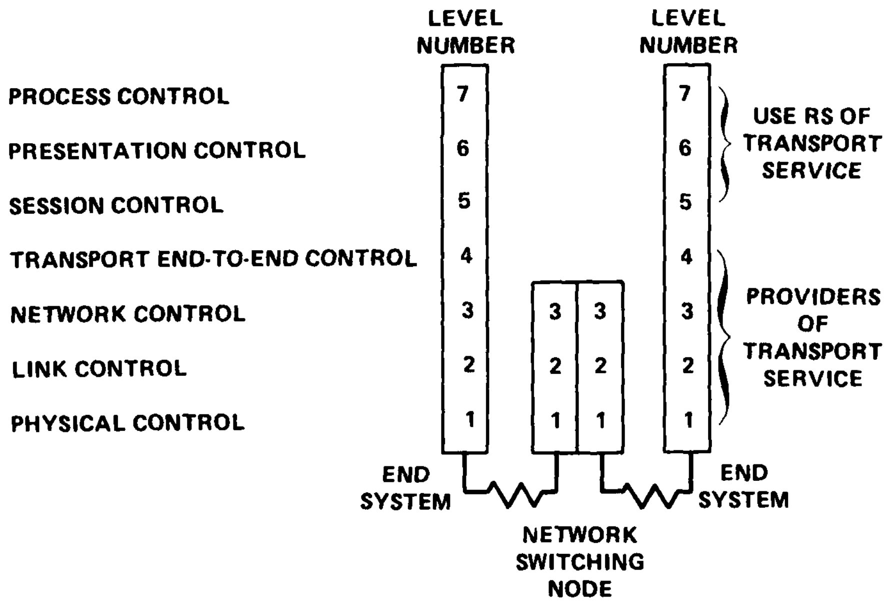


Figure 7 - ANSI/ISO Open Systems Reference Model

The new protocol standards provide an efficient method of transmitting data and have well-defined levels of software implementation. The link level protocol(s) will be dependent on what the host(s) will support. The high level protocol specifications, which are host-dependent, will affect the design of future Message Systems.

SECURITY

The physical separation of the data files and the DBMS system from the host system will greatly enhance the security of the data in the data base. The level of security provided by a back-end system, however, can vary greatly depending on which machine the various software components execute on. This section will define the alternatives which will be drawn upon in the concluding section.

Security is the enforcement of controls on access to data. This means that certain specified people may have specified access privileges to certain data elements. Historically the controls have been enforced by two components: the host machine operating system and the DBMS. The operating system and DBMS performed:

- User Validation - ensuring that the user is who he says he is (operating system).
- File Security - limiting the access (read/write/execute) a user may have to file (operating system).
- Record Security - allowing access (read/write) to the records specified in the subschema (DBMS).
- Authorization Validation - determining whether a user has the right to use a particular entity, such as a system command (operating system) or a subschema (DBMS).

In a general purpose host machine the operating system must provide a number of services to a variety of users with differing requirements, including a minimum of security for every user and the capability of additional protection if the user desires it. Minimum security means that files remain intact or can be restored and programs can be executed without interference. The DBMS security is built upon and uses the capabilities provided by the host operating system. The security provided by the DBMS can be no stronger than that provided by the operating system. Additional security will require changes to the operating system which could be difficult if not impossible because they would affect all other users.

When the DBMS and its utilities are placed on a back-end processor, the conditions change dramatically. The back-end has a very different environment than does the host. One can imagine a wall before the back-end, as impenetrable as one wishes to make it. We are able to impose this wall before the back-end for the following reasons:

- users and their files (if any) are strictly separated
- the back-end does not execute user programs
- the back-end operating system is small and specific and may severely limit the capabilities provided to a user
- the number and types of ways into the system are strictly controlled
- there is no direct access to files; a file may be accessed only via a "system" program (e.g., DBMS)

The concentration of security measures at one site simplifies the problem and requires, at most, minimal changes to the host system.

User validation is the only security function that depends on data received from the other side of the wall. This function is directly related to the problem of transmitting/receiving data to/from a user in a secure manner. The solution to both of these problems is to require that all transmissions be encrypted. If a scheme such as the public-key crypotosystem of Diffie and Hellman[1*] is used, then a "digital signature" technique of assuring a user's identity is available. This system provides a very high degree of security and is easily implemented. Thus the secure transmission of data from/to a user can be assured and the identity of that user can be guaranteed.

If only the data is to be secured, then the data manipulation language (DML) compiler could run on the host machine (this situation will be called level 0), and the steps taken by a user (DML compile, COBOL compile, link, load, execute) using a DBMS on the host would be the same as those for using a DBMS on a back-end. This arrangement is very convenient, but it assumes that the subscheme that contains information about the data base structure is not to be protected in the same way as the data base itself. In such a case, the back-end processor need have only two entries: a system console from which the DBA enters the schema, subschema, and DML and runs the DBMS utilities; and a transmission line controlled by the Message System for

*A complete listing of references is given on page 65.

accessing the data. This is the simplest case and provides for data security. There are three problems with this approach: the subschema is on both the host and back-end machines, the DML compiler is on the host, and the host has DML source, COBOL source, and binary files.

If, however, knowledge of the structure of the data base is to be protected, the back-end functions must be increased. Three levels of protection may be desired. The first (level one) requires only DML compilation on the back-end and allows COBOL source files, which contain imbedded structural information, on the host. Source files are sent to the back-end, precompiled, and returned to the host with the DML calls and data division filled in. The second (level two) requires the DML compiler and a cross COBOL compiler on the back-end. This allows only binary files on the host. In this case execution is still on the host. The third (level three) permits no structural information on the host. The user program runs on the back-end and only results need be shipped to the host. This approach is not actually a back-end DBMS but a stand-alone DBMS machine. Present implementation is at the first level. All three levels make security on the back-end more difficult to implement. They all require access to files on the back-end, which is a relatively minor problem; level three allows user programs to execute which is a major, but not insurmountable, problem since the program I/O functions can be limited. However, in all three cases, there is only one copy of the subschema and it is on the back-end machine. This is perhaps the major advantage of these three levels over level zero.

Level one requires a source transmission to/from the back-end for every compilation, but transmission could be greatly reduced if the back-end included an editor and syntax checker. This level would also require a terminal processing capability if the source updating were to be done interactively. With an editor on the back-end no DML source files need be on the host. This level would transmit to the host only syntactically correct COBOL programs and would be host independent (work for any standard COBOL program).

Level two transmits DML source files to the back-end and binary (link edited) files to the host. The host contains no source files, and compilation and link edit runs are reduced. The back-end, however, has a cross compiler and link editor for each different manufacturer's machine. This approach can be used for all programs, not just DBMS programs, and significantly reduces resource usage on the host.

Level three does not require a host; everything is done in the back-end. The back-end compiler produces code for the back-end processor and requires all code necessary to run, including system libraries, on the back-end. There are files but no programs and no message system on the host. The programmer, however, must debug his program on the back-end.

It is clear that increasing levels require increasing resources on the back-end processor. Off-loading too much to the back-end could make the back-end into a large machine providing many of the services of the original host machine. The decision of how much to off-load is dependent on many factors, the most important of which are:

- development effort needed
- hardware and communications costs

Level one, with editor and syntax checker, appears to offer the best overall solution. It is minimum level at which all data and structure are together and is the highest level of security that is host independent. It will reduce, on the host, file space (DML source, DML compiler, and subschema), compilation time (all COBOL programs will compile), and text editing time. It does not preclude going to level two if it is desirable for some host machine, but takes advantage of optimized compilers and debug facilities on the different hosts.

A system may be started by implementing a level zero configuration and progress in sophistication, but more back-end resources may be required than are available. This problem can be solved by splitting the back-end functions onto two machines with shared disk, which allows specialization of the two machines: DBMS on one and text editing and compilation on the other. This solution is made practical by the drop in the cost of the CPU and memory and the high cost of peripherals.

SYSTEM DESCRIPTION

The back-end system was initially implemented on an IBM 360 and a PDP 11/70 back-end as a test system for the complete software system described in this section. The host software was also implemented on a CDC 6400 to demonstrate the portability of the system.

The back-end system required the host and back-end processor sites to have the following minimum hardware and software.

Host Site:

- IBM 370 CPU
- 270X or 370X transmission control unit
- SVS operating system
- Tape drive to install software
- BELL or equivalent modem

Back-End Site:

- PDP 11/70 CPU
- 160K words of memory
- DQ11 Synchronous Communications Interface
- TU16, TE16 or TU10 tape drive
- IAS release 2.0 operating system
- BELL or equivalent modem

The software configuration, shown in Figure 8, consisted of a Host Interface (HINT) module, a Host Network Resource Controller (HNRC) task, and a host Message System (MS), and a Message System (MS), a Back-End Network Resource Controller (BENRC), a TWIN and a Data Base Management System on the back-end processor.

GENERAL FLOW

In the back-end environment the application program cannot call the DBMS directly. Instead the program calls an interface routine (HINT) with the appropriate code (DML verb) and data. The HINT passes this information through the Message System on the host, over the communications channel, and through the Message System on the back-end to the TWIN. The TWIN reformats the code and data and calls the DBMS. The results of this call are passed back to the application program via the TWIN, Message System pair, and HINT. The Message Systems on the host and back-end are functionally mirror images of each other.

Each application program has its own HINT (linked with it) and is assigned a separate TWIN. There may be, therefore, as many active TWINs as running application programs. Multiple application programs/TWIN's impose two additional requirements; multiple start-up/clean-up capability and multi-threading of one or more communication lines so that an application program and its associated TWIN can exchange messages. The start-up/clean-up is performed separately by the HNRC and BENRC. There is one copy of the HNRC on the host and one copy of the BENRC on the back-end.

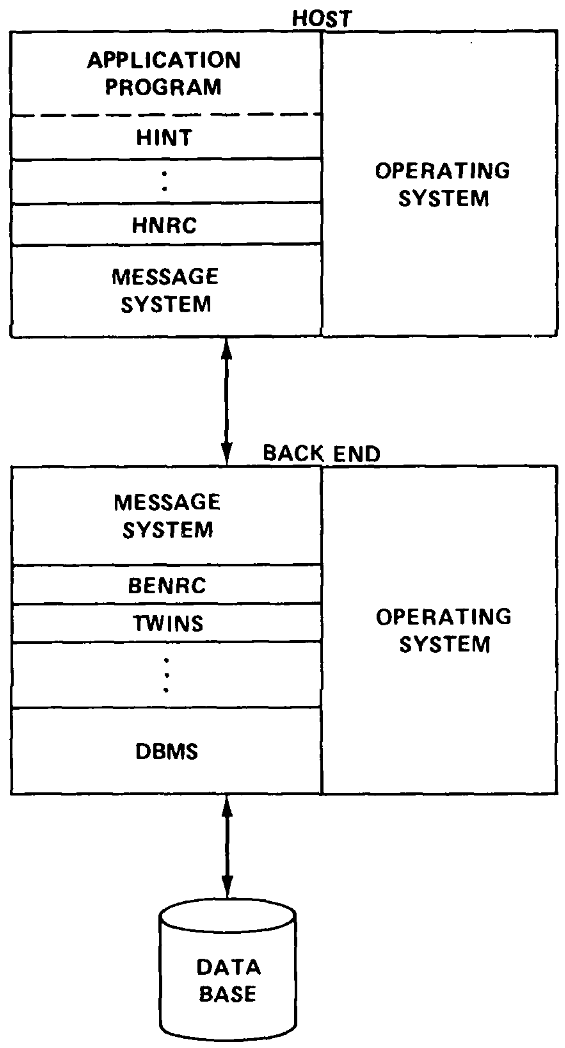


Figure 8 - Host/Back-End Software Distribution

SYSTEM SOFTWARE COMPONENTS

Host Interface (HINT)

A HINT module is link-edited into each application program and serves as the interface to the message system.

Before a task (application program) can communicate with the back-end processor, it must have a unique identifier (ID). The first call to HINT from the application program is a request to sign onto the data base. Hint requests a unique ID from HNRC and then requests the allocation of a TWIN task on the back-end processor. After obtaining the unique ID of the TWIN from the BENRC, HINT issues a connect to the TWIN to establish a communications link between the HINT and the TWIN. The HNRC and BENRC now drop out and the HINT intercepts all data base requests and forwards them to the TWIN for processing.

When the data base is closed, the HINT issues a DISCONNECT request to the message system to release the TWIN.

Host Network Resource Controller (HNRC)

The HNRC is responsible for connecting and disconnecting an application program (via HINT) and a TWIN. The HNRC functions are:

- to assign a unique ID to a HINT at connect time
- to communicate with the BENRC to establish a TWIN on the PDP
- to notify the HINT of the corresponding TWIN ID
- to receive FINISH requests from HINT
- to manage a table of active run units
- to sense abnormal termination of application programs and inform the Message System
- to provide operator communications

The HNRC contains a separately assembled table to establish correspondence between subschema names and BENRC ID's. New subschemas require that an entry be assembled in this table.

Message System (MS)

The MS provides a message-based communications facility between any two processors in a network connected by various copies of the MS. The functions of the MS are:

1. To provide synchronization between tasks as well as between processors
2. To provide a message exchange system between tasks through which both data and control information can pass

3. To handle buffer management in both the host and back-end processor

4. To isolate the application program interface and the DBMS tasks from any knowledge of the physical location of the others, i.e., whether the host and back-end are connected locally or remotely

5. To provide a well-defined interface (the CALL statement) to both the interface and the DBMS task

The MS is modular and hierarchical to permit easy adaptation to most operating and teleprocessing system environments.

A task may be connected to several other tasks concurrently. Each connection is made through a different "port" (p) of the task (t) running on a machine (m) within a cluster (c) of machines. A cluster is a logical grouping of machines and could, for example, conveniently identify a particular site. The complete ID of a particular port of a running task is of the form c.m.t.p. Before a message can be sent to another task, the ID (c.m.t.p.) of that task must be known. A "well known port" is the ID of a task that is fixed and known to all other tasks. The network resource controllers (HNRC and BENRC) are such tasks and can communicate with each other to assign ID's to the application programs and TWIN's so that they may communicate. A "well known port" must not remain connected to any task since it must be available to all tasks in the network.

The MS supports four basic I/O functions that permit bi-directional communication between the DBMS and the remote user on the host. These functions and their descriptions are:

1. Connect Function - establishes a logical connection between a dedicated TWIN and a remote task on the host system. An optional feature is to exchange command and/or data messages when the connection has been established.

2. Disconnect Function - terminates a logical connection between two tasks. The task in question must previously have been successfully connected.

3. Send Function - transmits an (optional) command message and/or an (optional) text message to another task. The receiving task must have been previously connected. Messages are limited to 0-65535 bytes.

4. Receive Function - accepts a message from another task. The message may consist of an optional command message and an optional text message. The two tasks

must have been previously connected. The command message must be in the range from 0 to 128 bytes. A text message must be within the range from 0 to 65535 bytes. If either of the buffers is too small to contain the transmitted message, an appropriate error message is returned to the sender.

Back-End Network Resource Control (BENRC)

The BENRC is constantly ready to receive messages from remote NRC's and performs the following functions:

- Receives a host NRC request to assign a TWIN task for back-end data base access.
- Assigns a TWIN task and updates a common data area for that TWIN, giving it the Task-ID of the host application.
- Activates the TWIN task.
- Sends a message to the host NRC informing it of the ID of the TWIN task.

TWIN Task

TWIN tasks are allocated by the BENRC at run time. Each application task on a host communicates with its own TWIN on a one-to-one basis. The application task issues DML commands, which are packaged into a message by the host interface, transmitted by the message system, and converted to a DBMS call by the TWIN:

- Characters are converted from EBCDIC to ASCII by the TWIN
- The DML command is executed by the DBMS.
- Characters are converted from ASCII to EBCDIC by the TWIN.
- Results are packaged into a message by the TWIN and transmitted by the message system to the application interface which places them appropriately in the application program.

Data Base Management System

The data base management system used in the back-end system adheres to the recommendations of the Data Base Task Group (DBTG) of the Conference on Data Systems Languages (CODASYL).[2]

The data base system comprises a number of components, all of which execute on the PDP 11/70 back-end processor. No portion of the DBMS or its utilities currently executes on the IBM host. Included in the DBMS are:

• data description language (DDL) compilers for describing the data and its inter-relationships,

• data manipulation language (DML) compilers to connect CCBOL and FORTRAN syntax extensions into data base calls,

• a data base control system (DBCS) for run time control of and access to the data base,

• a device/media control language (DMCC) to allow the data base administrator to describe the configuration of the data base, and

• a comprehensive set of utilities for maintenance and control of the data base.

The following paragraphs describe each of these components in more detail.

Data Description Languages (DDL). Two DDL compilers are provided for describing the data base. The first, the SCHEMA compiler, provides a means for describing the total data base. The SCHEMA includes all the data elements and records and describes the inter-relationships. The SCHEMA compiler stores the description in a data dictionary for later reference.

The SUBSCHEMA compiler provides a means of limiting access to the data base. The SUBSCHEMA describes a particular application's view of the data base. This view is a subset of the SCHEMA description. A SUBSCHEMA description includes data element, record, and relationship descriptions as well as access rights information. All user requests to the data base are checked against the SUBSCHEMA for validity. The SUBSCHEMA compiler places a copy of the SUBSCHEMA definition in the data dictionary and produces a module to be used by the DBCS in locating/verifying data base accesses.

Data Manipulation Language (DML). The version of the DBMS used in the back-end system supports one DML COBOL. The DML is a series of syntax extensions to COBOL which allow manipulation of the data and the relationships in the data base. The types of extensions fall into three categories: retrieval, control, and modification. The retrieval extensions provide COBOL verbs for locating and obtain-information in the data base. Control extensions provide a means for specifying SUBSCHEMAS, opening/closing the data base, and checking data relationships. Modification extentions provide the facility to extend, modify, and erase the data base.

The COBOL DML compiler works on standard COBOL program source files. It produces two output files: an error listing, and a new COBOL source file with the DML syntax converted to external call statements. This latter file is then compiled using a standard COBOL compiler.

Device/Media Control Language (DMCL). The DMCL compiler gives the data base administrator control over the configuration of the data base system. Using the DMCL, the data base administrator may declare portions of the data base on-line or off-line. Thus sensitive portions of the data base may be dismounted and kept physically secure until needed. The DMCL also allows the specification of "journaling", the logging of update transactions for recovery purposes. If journaling is specified, it may be turned off at run time, but it can be used only if actually specified.

The DMCL tables may also be used as a data base cache. Pages from the data base are stored in buffers in the DMCL module and are kept in memory as long as possible. When the buffer pool fills up, pages are written back out to the data base using a least-recently-used algorithm.

The DMCL compiler accepts one specification file as input and produces a module to be used by the DBCS. A copy of the DMCL specification is also written into the data dictionary.

Data Base Control System (DBCS). The DBCS performs the run time processing for the data base system. It is responsible for locating information within the data base and transferring the appropriate data to the user. In addition, the DBCS verifies access rights for each user from the user's SUBSCHEMA module.

Utility Programs. The DBMS provides a set of utility programs that allow the data base administrator to monitor and maintain the data base. The more important utility programs are:

Initialize - initializes the data base; it can initialize the entire data base or a specified portion. It can also be used to initialize the data dictionary.

DUMP - copies all or specified portions to a sequential device. It also provides statistics about the areas copied including a list of record types found, number of record occurrences, and total space occupied.

RESTORE - restores the data base using a DUMP produced by the preceding utility.

Roll Forward/Roll Back - modify the data base according to the journal tape. The roll back utility will restore data modified while journaling is enabled. The roll forward utility will restore a data base to a known state by re-applying changes made while journaling is enabled.

Incorrect page fixer - allows direct modification of data base pages. It is useful for recovering corrupted data bases.

Data Dictionary Report - produces a variety of reports from the data dictionary. Reports include record description, relationship description, SUBSCHEMA listing, and DMCL listings.

Data base operator - used for starting/stopping data base operation and specifying which DMCL module to use. Journaling may be turned on/off using DBO.

BACK-END SYSTEM LOGIC

CONNECTIVITY AND ROUTING

The message system is a multithread, multiline connection based system. This means that one or more tasks in one machine may send messages to one or more tasks in other machines over one or more physical lines. In the simplest case a task (application program) is simply connected to a task (TWIN) on another machine via one physical line. In the general case, however, a task may be connected to any number of tasks or ports within a task (up to 256 of each).

The connection ID, in the form c.m.t.p. (cluster, machine, task, port), is the mechanism by which the message system delivers messages to the appropriate receiver. A connection may be either inter- or intra-computer. The message system routes outgoing messages, as a series of one or more packets, to the correct physical line for inter-computer messages and to the incoming queue for intra-computer messages. The message system transmits messages to remote stations as a series of packets. A packet is a unit of transmission up to a fixed size. This technique allows the transmission of long messages, which may exceed the handling capacity of the remote station, in a series of manageable units. The format of a packet is shown in Figure 9.

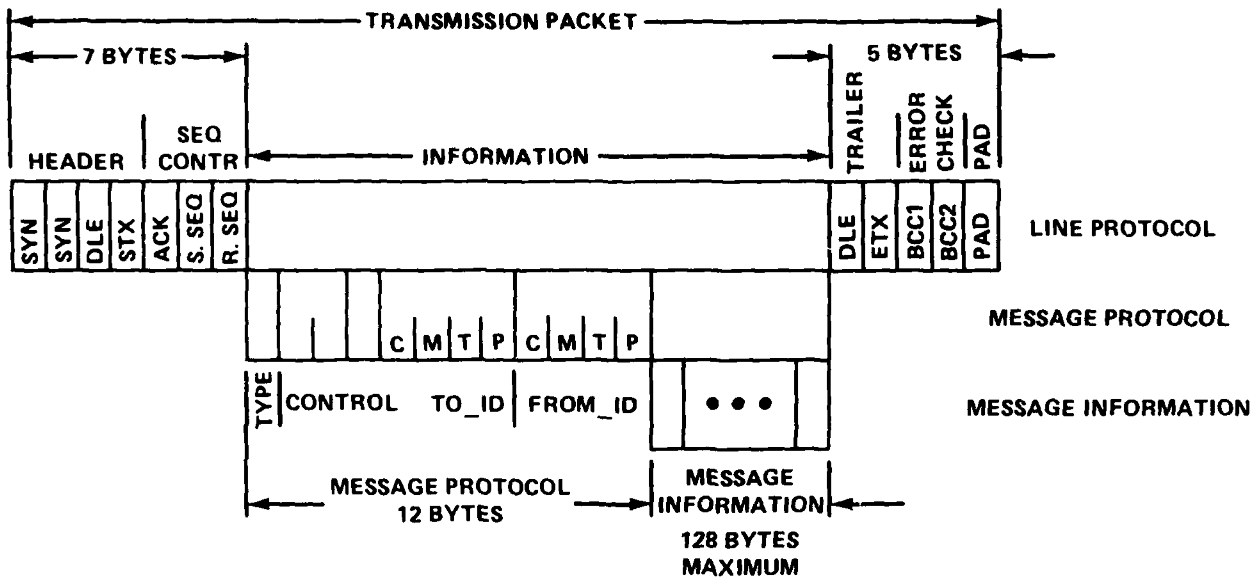


Figure 9 - Packet Format

Routing is controlled by use of the concept of the logical line. A logical line is a design convenience and has no physical counterpart. It is a nodal point where host (machine running this copy of the message system) information meant for one-to-many remote stations converges and then diverges to one-to-many physical lines. Logical lines are governed by the following rules:

1. At any given time, a remote station can be serviced by only one logical line.

2. A remote station may be rerouted to an alternate logical line, but still can be serviced by only one logical line.

3. A logical line can service one-to-many remote stations.

4. A logical line can be serviced by one-to-many physical lines.

Rules 1, 2, and 4 state that, although there is only one path to a remote station, that path may consist of one or more physical lines. Rule 3 states that messages for more than one remote station may use the same logical line. By this means messages can be forwarded to remote stations which are not directly connected to the host.

Routing is implemented by a series of tables interconnected by queues. These tables are:

Remote Station Table - description of a remote station

Logical Line Table - description of a logical line

Physical Line Table - description of a physical connection between this host and a remote station. This table also describes the physical line to the scheduler

Host Table - description of the host and its cluster.

Within the message system one logical line is predefined as the intra-machine communications path (logical line zero). A process that acts as a line driver for the internal data path transfers outgoing packets to the incoming queues.

MESSAGE SYSTEM COMPONENTS

The message system is composed of five logical components: message subsystem, operator interface, packet subsystem, line protocol subsystem, and line drivers. The functions of the subsystems are:

Message Subsystem

- Processes messages concurrently in a multiprogramming environment.

- Interface to local operating system
- Divides messages into fixed length packets for sending
- Reassembles received packets into messages for local receivers
- Maintains current status of message (start, receiving, sending, waiting, done)
- Maintains connection status

Operator Interface

- Accepts and validates network description from operator
- Invokes message subsystem and line driver on operator request
- Sends compressed network description to message system
- Activates system and logical lines on operator request
- Deactivates system and components on operator request
- Requests gathering of statistics
- Reports statistics gathered

Packet Subsystem

- Manages buffers
- Maintains all queues
- Appends/interprets MS system protocol
- Distributes packets to line protocol processors for sending
- Receives packets from line, queueing them for MS subsystem processing

Line Protocol Sybsystem

- Composes/interprets appropriate line protocol appended to the packet.
- Protocols supported
    - BISYNC
- Supervises line mode
    - HALF-DUPLEX
    - FULL-DUPLEX

Line Driver Subsystem

- Line drivers for the specific hardware which drives the line.
    - DQ 11 SYNCHRONOUS INTERFACE

IMPLEMENTATION ON THE PDP 11/70

Message System

The back-end (BE) message system (MS) is the communications interface that controls the routing of user commands and/or data messages to its subsystem software components (either the data base management system (DBMS) or an associated line driver). The MS implementation being described runs on a Digital Equipment Corporation (DEC) PDP 11/70 minicomputer, under an IAS operating systems environment. Communications between the MS and its subsystem software are implemented by the queue input/output (QIO) IAS directive. When a user task (on the PDP 11/70 issues a QIO directive, an I/O request for the indicated device is placed in an IAS queue in priority order for that device. The MS is regarded as a single device and is addressed as a logical device. When a given text and/or command message enters the message system, intermediate processing in addition to routing to its designated TWIN (on input of a DBMS function) or the associated line driver (on output) begins.

Scheduler

It is the scheduler that enables the system to process a message in an event driven environment (processing procedures are invoked only when necessary). All processes of the message subsystem, packet subsystem, and line driver are monitored through the scheduler, all processes return to it, and it is the module that exits to the operating system on a WAIT (wait on interrupt during idle periods).

The scheduler is driven from a double-ended queue. Each entry in the queue is a packet allocated (or rather, pre-allocated) from the packet buffer pool. Each entry is structured with the scheduler control information followed by a stack/register storage area and finally, the particular table driven procedural processes.

All processes monitored by the scheduler are re-entrant. Rather than thinking of the scheduler as monitoring "programs" or "procedures", it is better to regard it as monitoring physical lines, logical lines, data movers, etc., which require servicing by means of processes. Therefore, the particular table mentioned is one describing the characteristics and addresses of physical lines, logical lines, data movers, etc. (In general, these are called processes.) The characteristics of the physical lines are supplied by the computer operator at systems initialization time, or by the human operator of the BE via the message system operator (MSO) module at run time. Logical lines are established and maintained by the remote user on connection to a TWIN via the connect function.

Once a text message or command packet enters the MS, it is placed in a start queue, and the corresponding event flag is set. When the scheduler detects the presence of the message/command packet, evokes the corresponding procedure. The text packet and/or command then undergoes a series of intermediate processing steps. Each process performs a specific function, and then removes the packet from one queue, places it in another, and sets the corresponding event flag. When the scheduler is reactivated, it detects the presence of a significant event (queue entry flag set) and evokes the corresponding process.

Nine processes are hard coded into the scheduler's process control logic. Each process performs a specific function on the message packet en route to the DBMS via the line driver or from the DBMS to the line driver. The nine queues that correspond to the hard coded processes are in order of priority:

	QUEUE	COMMENTS
1.	MP.QIO	Gets a QIO node from IAS
2.	MP.SRT	Gets a QIO node moving through the system
3.	MP.IC	Entry processing queue for incoming commands
4.	MP.IN	Entry processing queue for incoming text packets
5.	MP.OUT	Entry processing queue for outgoing text packets
6.	MP.DON	Entry processing queue for transmitted packets via QIO exiting the system
7.	PLZERO	Entry processing queue for internal physical line initialization
8.	MP.GARG	Entry processing queue for unrecognized incoming packets (not fully implemented)
9.	MP.CTL	Entry processing queue for operator control

Each process exits to the scheduler for additional processing in these process control queues.

Call Node Queues

A pool of nodes is maintained for various queues which monitor a call to the MS. These queues are distinct from the message queues which are processed by the MS. They are referred to as Call_Node_Queues* which contain nodes that are different from and smaller than Packet_Nodes. The important difference is that Call_Nodes contain information about the message, whereas Packet_Nodes contain the actual message. Five queues are concerned with the state of messages. Each queue holds all message calls for a certain state and provides data for the particular process that advances the message call to the next state. These queues are briefly described as follows:

1. Operating System Queue - the data structure monitored by the MS and maintained by the local operating system. All calls to the MS are obtained from this queue.
2. MS_Start_Que - the queue that contains message system calls for a receive or send function. The appropriate event flag is set if this queue is empty when a send or receive enters the MS. This queue starts a message through the MS.
3. MS_Wait_Que - the queue which contains message call for a send or receive that is not currently in progress.
4. MS_RCV_Que - the queue of MS calls actively receiving an incoming message.
5. MS_Send_Que - the queue of MS calls actively sending a text message.

Internal Message Routing Processes

Message System Queue Input/Output (MS.QIO) Process. This process takes I/O request nodes from the IAS operating system and starts them through the MS. The preliminary processing performed by this process includes QIO function verification, call node initialization and allocation, and message routing verification (verifies the existence of a logical connection). The process determines the function type and places the call node on the appropriate queue (send or receive). It also places a copy of the call node on the MS_Start_Que. The MS.QIO process has the highest priority of all those hard coded into the scheduler.

*The single underline, is a legitimate COBOL character which is used as a spacing character.

Message System Text Start (MS.SRT) Process. This process dispatches the QIO call nodes to the appropriate processor. Its main function is to enable the system to queue send, receive and connect messages. This process has the second highest priority of those hard coded into the scheduler.

Message System Incoming (MS.IC) Process. This process determines whether a connection exists between the command TO-ID and FROM-ID. If a connection does not exist, this process delays the message. If the command is for a send or receive request and a matching QIO exists for the call node, the process dispatches it to the appropriate processor. If a QIO does not exist for the call node, the message is delayed until the required QIO is available. This process controls the synchronization between command sending and receiving tasks and has the third highest priority of those hard coded into the scheduler.

Message System Incoming Message (MS.IN) Process. This process handles the reassembly of fragmented text messages from receive packets into the user area provided by the connected TWIN. It works with text packets only, never with command messages which are handled by MS.IC (see preceding section).

This process obtains the text to be reassembled via the IN_TEXT_QUE. The IN_TEXT_QUE is a queue of all incoming text packets. When this process is evoked, it verifies that there is a host task port (a previously established connection) currently receiving a message in the MS_RCV_QUE. If there is no receiving message request, then the packet is unknown and is placed in the IN_SPOOL_QUE (queue of all unknown packets). If there is a receiving message in the IN_RCV_QUE, then MS.IN copies the text from the packet to the user's buffer area. Upon completion of a text transfer, this process frees the packets from the IN_TEXT_QUE and the associated call node and moves the message to the MS_DONE_QUE. This process has the fourth priority of those hard coded into the scheduler.

Message System Outgoing Message (MS.OUT) Process. This process forms a user message into a packet(s) and queues the packets for transmission. This process works only with text packets, never with command packets which are handled via MS.IC (see previous section).

Each logical line (refer to section entitled CONNECTIVITY AND ROUTING) is limited in the number of packet buffers that may be assigned to it to prevent the entire buffer resource being used by one line and one large user buffer. When the number of buffers allowed for a logical line is reached, this process delays further processing of that user buffer. As packets are transmitted for a delayed line, this process converts more packets for a given user buffer. This process continues until all of the text from a given user's buffer has been placed in packets. Then the message entry is moved to the MS_DONE_QUE.

Message System Done (MS.DON) Process. This process completes processing of QIOs which have been queued for transmission (fully processed by MS processes), including both messages which have been successfully completed and those which have been processed to an error state. This process is activated whenever a call node is placed on the MS_DONE_QUE. It removes QIO call nodes and performs the final processing of the messages, the type of processing done being determined by the message function code. It issues an I/O DONE when final processing is complete. In addition, it will queue aborted packets for eventual transmission, where indicated. This process has the fifth priority of those hard coded into the scheduler.

Physical Line Initialization (PL.INIT) Process. This process reserves logical line zero in all systems for the tasks residing in a given machine, enabling tasks to communicate within the same machine. This process is evoked whenever a text message is placed on an output queue directed to logical line zero. It takes such messages and transfers them to the incoming queues. The scheduler is, in turn, alerted to the presence of an incoming message and repeats all active processes to complete processing of an incoming packet. This process ensures, in most cases, that one pass through the MS will suffice in transferring messages from one port to another, except when the receiving task is not resident at that time.

Operator Interface

The functions of the message system operator (MSO) are to:

1. Define the communication network available to the message system.

2. Initiate the process to terminate execution of the message system (MS....), Line Driver (DQ....), and Twin Allocation (NRC) tasks.

3. Request snap shot dumps of selected tables in the message system.

These functions are performed using the following nine commands in MSO:

Define	- Define a network element
Link	- Define the linkage of network elements
Enable	- Request startup of a physical line
Run	- Request execution of a handler (MS or DQ)
Stop	- Unload a handler
Dump	- Perform a snap dump of a portion of the message system tables
AWKP	- Initiate the TWIN Allocation task
DWKP	- Disable allocation of additional TWIN tasks
ABRT	- Abort all existing TWIN tasks

Back-End Network Resource Controller

The BENRC performs several functions concerned with network resources:

1. It coordinates with the host NRC to allocate resources which are Back-End Tasks (BET's) or TWIN's (refer to page 34).

2. It controls the BET's in that the BENRC starts them and, in the event of an operator ABORT directive, stops them.

3. It reports the status of all back-end network resources to the operator on request.

The BENRC runs as a real-time task under the IAS/RSX-11 operating system and consequently may be resident for only short bursts of processing. These bursts are initiated by the message system. When NRC begins processing, it immediately issues a RCVD$ IAS directive to receive any waiting data. The first word of the received buffer indicates the function to be performed.

AWKP:	Causes NRC to identify the buffer as a well-known PORT.
DWKP:	Causes NRC to terminate the buffer well-known PORT.
ABRT:	Causes NRC to abort all currently active BET's and to terminate itself as a well-known task.
DACT:	Causes NRC to pass current back-end network status on to the unit.

Data waiting for the RCVD$ IAS directive contains a minus one in the first word and is assumed to be a message from the message system indicating that a CONNECT has been issued for the NRC from some host NRC. The BENRC then issues a QIO$ directive to the message system (MS) unit for a receive. The message received must be an "ALLOCATE BET" request from the host NRC. The BENRC then finds the first available TWIN and issues a RQST$ IAS directive to run it. The BENRC then constructs a BET ALLOCATED message for the host NRC and issues a QIO$ IAS directive to the MS unit to send the message. The BENRC then issues a QIO$ to the MS unit to disconnect from the host NRC and exits.

TWIN

The BE TWIN performs the actual data base calls for the user program on the host. The TWIN is started by the BENRC task but is responsible for establishing communication with the user task in the host. On startup, the TWIN inspects its entry in the SGA call TWINST. The entry contains both the task identification that the TWIN is to use when referring to itself and the task ID of the user task on the host machine. The TWIN uses these values to issue a QIO$ to the message system (MS) logical unit to attach to the user task. When this is complete, the TWIN issues a QIO$ to the MS unit for a RECEIVE.

The TWIN waits for the RECEIVE to be satisfied. When the message arrives, the function code in the first byte is used to access the processing control table entry for the Data Manipulation Language (DML) command. The processing table directs the construction of the DML call from the rest of the received message, and the DML call is executed. A response message is then constructed (still under the auspices of the processing table). At this point, the TWIN is ready to issue a SEND directive to the message system. The TWIN issues the QIO$ to the MS unit to send the response it has constructed. The TWIN now restarts the processing loop by waiting for the receive to be completed, etc.

There are only two ways to exit from the loop. The first occurs when the received command is the DML FINISH command. In this case, the response is sent without the preceding RECEIVE being issued and the SEND is followed by a DETACH (DISCONNECT) and exit. The second way out of the loop occurs whenever a message system directive returns an error code after completion. In this case, DETACH and EXIT are executed to clear the task.

TEST PROGRAMS

Three test programs and two different data bases were used in the tests. These programs exercised all the functional capabilities of the DBMS but did not test every possible command. For example, only six of the possible 27 FIND/OBTAIN DML verbs were used.

The first and primary test program was a customer order program that added new records to the data base and then processed the linked records in order and displayed them. This program had enough data to allow multiple areas to be examined in one run.

The other two programs used an airline reservation data base. One program added flights and the other connected the flights to cities.

These three programs provided an adequate base to test the back-end software for all allowable input. No processing of data was done by any of the programs, and the elapsed time was spent primarily in processing DML verbs.

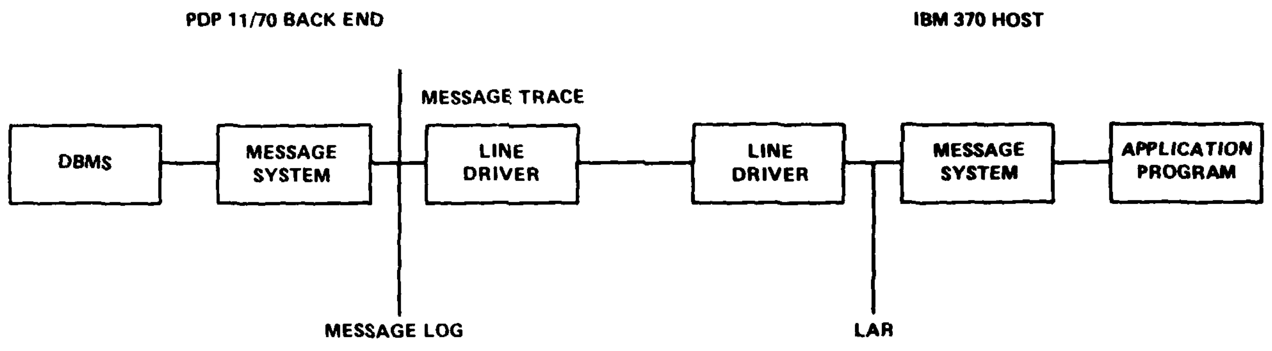
PERFORMANCE MEASURE METHODOLOGY

The host/back-end software is symmetric: application program, HINT, message system, and line driver on the host; and line driver, message system, TWIN, and DBMS on the back-end. Therefore, the back-end was chosen as the primary test site because it is a stand-alone system and any test modifications would not impact other users. Also system statistics available on the IBM would give sufficient data if a problem developed there.

The parameter of critical importance was time. If the back-end could reduce CPU time on the host and not add too much elapsed time, the concept would be validated. It was apparent from the start that elapsed time was excessive and that it was necessary to determine whether the time was uniformly distributed and irreducible or whether enhancements/modifications could make significant improvements.

Of the three software components on the back-end (message system, TWIN, and DBMS) the TWIN and DBMS were excluded from detailed testing--the TWIN due to its simplicity and the DBMS because existing test results had shown it to be capable of between 10 and 60 "CALC's" (calculation of data base address and retrieval) per second. Thus the message system was the only unknown, and it was therefore subjected to intensive testing.

Several testing tools were already available on both the host and the back-end. These and a back-end message system histogram were used to identify any bottlenecks on the back-end side. The difficulty with this approach was that the analyst had to coordinate four separate lists of test data. Figure 10 shows the various zones tested.



LAR-TIME MESSAGE SENT TO ANSWER RECEIVED.
-TIME MESSAGE RECEIVED UNTIL NEXT MESSAGE.

MESSAGE LOG-TIME DML VERB RECEIVED.

MESSAGE TRACE-TIME EACH MESSAGE SENT/RECEIVED.

HISTOGRAM-TIME SPENT IN EACH QUEUE IN MESSAGE SYSTEM.

Figure 10 - Test Timing Zones

MESSAGE TRANSMISSION/RECEPTION LOGGING

The IBM Line Activity Recorder (LAR) is a general purpose utility that permits external monitoring of a communications link. This utility permitted time stamping of all incoming and outgoing messages in units of microseconds, although this precision was greater than needed, since the PDP clock has a resolution of only one sixtieth of a second. The LAR can measure on either side, and thus the total time of the application program, HINT, and host message system was measured as was that of the host line driver, communications channel, and all back-end software.

The back-end message system provided two traces: a time stamp (hour, minute, second) and number of each received DML verb, and a listing of each transmitted and received message (including idle packets). The time stamp provided the time between DML verbs and reflected the time spent in all components of both the host and the back-end.

These transmission logging tools, which already existed, indicated the time used by several components on one machine or the other but did not isolate any component.

TIME TAGGING STRATEGY

To obtain a reasonable approximation of the throughput and performance figures of the message systems, a sufficient number of timing samples had to be obtained on well defined I/O paths. To accomplish this, the various I/O paths were first defined and then a common measure of elapsed time (centi-second, clock tick (1/60th of a second), etc.) was developed. In addition, data structures for statistical data were maintained.

Performance and throughput measurement statistics were obtained from the back-end message system and stored in a local histogram. This histogram contained data extracted from timing samples obtained from the internal message system activity (input and output of commands/data).

The performance logic enabled us to view isolated elapsed times spent in various queues in route to DBMS (input) or to the line driver (output). In addition to the histogram, the following performance variables were maintained:
(TDELTA is the difference between the message origin time and the message transmission/receive time)

Total Elapsed Time - The total time spent collecting data for a given monitoring session (expressed in wall clock unit of measure).

Total Number of Samples - The total number of messages received.

Minimum TDELTA - The lowest TDELTA computed for a given monitoring session.

Maximum TDELTA - The largest TDELTA computed for a given monitoring session.

TDELTA Overflow - The total number of times that the specified upper limit was exceeded.

TDELTA Underflow - The total number of times that a negative result was obtained from a TDELTA computation.

Scaled TDELTA Range Accumulator (Histogram) - The dynamic range of scaled TDELTA values for a monitoring session (from 0 to 2.8 seconds).

For the purpose of simplicity, the systems clock was used to provide a source for time tagging of message system nodes, but conform to existing conventions, all TDELTA values were expressed in centi-seconds. In addition, all MS call nodes included space for time stamps, which were posted whenever an incoming or outgoing message was transferred from one intermediate processing stage to another.

Since there were two transmission paths within the message system, provisions were made to collect performance data in both directions. To obtain the most reasonable approximation of the total systems message throughput, all variables had to be considered, which meant that even the abnormal or invalid message handling paths had to be included in the overall systems timing analysis.

TESTING STRATEGY

Parameters external to the executing code were varied to determine their effect on throughput. Parameters considered were load on the host, line speed, and number of DBMS users. The first two affected minimum response time, and the third the amount of host resources that could be saved.

Loaded Host versus Unloaded Host

The tests were conducted twice daily to observe the relationship between the system's state (loaded or unloaded) and performance figures. Approximately half the test was conducted before seven o'clock in the morning with an unloaded host and the other half at mid-afternoon with a fully loaded host. All tests were run with a dedicated back-end machine.

4800 versus 9600 Baud Line Speed

The communications line speed was varied to note the effects of line speed on the throughput rate. The line speed variation was run on both a loaded and unloaded host. Because of the hardware configuration (one dialup 4800 baud modem and one dedicated 9600 baud modem), speed variation was very limited.

Single User versus Multiple Users

An attempt was made to collect performance data on both single user and multiple user jobs. These tests were unsuccessful because of a software problem in the host system.

Stand-Alone

Two tests were run on the IBM, using the same DBMS without the back-end. These tests were designed to show the difference in the required resources of the two modes; stand-alone and back-end.

TEST ENVIRONMENT

Whenever a test was to be run, one person was required to be present at each site. This was a minor inconvenience and to be expected on a prototype system. However, a production system should not expect anything of the programmer except instructions. This is not a minor problem and its correction would greatly simplify the running procedures.

PERFORMANCE MEASUREMENT RESULTS

Since the test results from the three programs showed a high degree of uniformity, no distinction as to program is made in the reported results. The test results are shown in one of three forms:

- Exact - results as they were printed with no editing (one test run),
- Extract - results edited to discard all unnecessary and confusing data but shown in the sequence printed (one test run),
- Summary - totals and averages of results from specified test areas of all runs.

For the specific (exact and extract) results a portion of one test run was chosen (24 April), the run with the fastest response per DML verb. The same portion is shown for PDP message logging, PDP message trace, and IBM LAR. The DML verb numbers used by the programs are defined in Table 1.

TABLE 1 - DEFINITION OF TESTED VERBS

DML Verb Number	Syntax
2	Close all areas
10	Find next Record-Name Record of Set-Name Set
11	Find next R-N Record of A-N Area
19	Find first R-N Record of A-N Area
31	Find Owner Record of Set-Name Set
32	Find Record-Name Record
36	Open Area A-N Usage-Mode is update
41	Open area A-N Usage-Mode is exclusive update
42	Store Record-Name Record
43	Obtain (any find format)
44	Insert R-N Record into S-N Set
48	Bind R-N Record
59	Bind currency for subschema name

Fifteen test runs were made on the host and back-end machines and two on the host stand-alone.

PDP MESSAGE LOGGING

The time at which a packet containing a DML verb is released to the message system and the number of the verb are shown in Figure 11. The tabulation of data extracted from the message logging for selected runs is shown in Table 2 and summarized for all runs in Table 3.

```
07:06:18 TWN101 -- FUNCTION = 10 0000000
07:06:20 TWN101 -- FUNCTION = 43 0000000
07:06:23 TWN101 -- FUNCTION = 31 0000000
07:06:25 TWN101 -- FUNCTION = 43.0000000
07:06:28 TWN101 -- FUNCTION = 10 0502100
07:06:31 TWN101 -- FUNCTION = 10 0502100
07:06:33 TWN101 -- FUNCTION = 11 0000000
07:06:35 TWN101 -- FUNCTION = 19 0000000
07:06:38 TWN101 -- FUNCTION = 43 0000000
```

Figure 11 - Sample PDP Message Logging

TABLE 2 - SELECTED PDP MESSAGE LOGGING

Date Run	Number of Functions	ET (M.S.)	MIN (sec)	MAX (sec)	SEC/FUN	BAUD
4/24	598	27.37	2	5	2.775	4800
5/1	598	30.45	2	8	3.090	4800
5/4	598	29.54	2	6	3.005	9600
5/10	598	30.18	2	5	3.045	9600
5/23	284	14.19	2	5	3.035	4800
6/1	284	14.32	2	4	3.081	4800

TABLE 3 - PDP MESSAGE LOGGING AVERAGES

	4700(11)	9600(4)
Min -	2	2
Max -	8	7
T # functions -	3733	1860
T ET	11,349	5977
T Sec/function -	3.049	5977/1856=3.220
	7L	
		2L
	4u	
		2u
Min Sec/f	2.775	3.005
Max Sec/f	3.366	3.599
# Runs	11	4

PDP MESSAGE TRACE

The first part of each packet, including idle packets, sent and received, by the PDP was recorded. Figure 12 shows the partial packets and all of the idle packets, and delineates the fields in the packets.

```
SND 2E 2C 5C
RCV 2E 5B 2D 01 06 00
SND 2E 2C 5C
RCV 2E 5B 2D 01 06 00
DEQ 00 00 00 01 0A 00 00 02 01 05 00 01 01 65 00 F0
SND 2E 2C 5C 01 0A 00 00 02 01 05 00 01 01 65 00 F0
RCV 2E 5C 2D 01 0B 00 00 01 01 65 00 02 01 05 00 0A
SND 2E 2D 5D
RCV 2E 5C 2E 00 07 00
SND 2E 2D 5D
RCV 2E 5C 2E 00 07 00
DEQ 00 00 00 01 0B 00 00 02 01 05 00 01 01 65 00 F0
SND 2E 2D 5D 01 0B 00 00 02 01 05 00 01 01 65 00 F0
RCV 2E 5D 2E 01 0C 00 00 01 01 65 00 02 01 05 00 3B
SND 2E 2E 5E
RCV 2E 5D 2F 01 08 00
SND 2E 2E 5E
RCV 2E 5D 2F 01 08 00
DEQ 00 00 00 01 0C 00 00 02 01 05 00 01 01 65 00 F0
SND 2E 2E 5E 01 0C 00 00 02 01 05 00 01 01 65 00 F0
RCV 2E 5E 2F 01 0D 00 00 01 01 65 00 02 01 05 00 13
SND 2E 2F 5F
RCV 2E 5E 30 01 09 00
SND 2E 2F 5F
RCV 2E 5E 30 01 09 00
DEQ 00 00 00 01 0D 00 00 02 01 05 00 01 01 65 00 F0
SND 2E 2F 5F 01 0D 00 00 02 01 05 00 01 01 65 00 F0
RCV 2E 5F 30 01 0E 00 00 01 01 65 00 02 01 05 00 3B
```

Figure 12 - Sample PDP Message Trace

PDP SYSTEM HISTOGRAM

The histograms show the time spent by messages in the twelve message system queues. The average number of seconds per queue for selected runs is given in Table 4 and the overall averages in Table 5. The percentage of queue elements processed per time interval for selected test runs is given in Table 6 and the overall percentages in Table 7. The averages in Figure 12 and Table 4 are computed by dividing the total time all messages spent in a queue by the number of messages.

TABLE 4 - SELECTED AVERAGE QUEUE TIMES
(Seconds/Queue)

Queue	5/1 9600 Baud	5/4 9600 Baud	5/10 9600 Baud	5/23 4800 Baud	6/1 4800 Baud	Total
ALLQUES	.55833	.47849	.54023	.49139	.51044	.52952
MESQUES	.46509	.46809			.44457	.46184
COMQUES	.64387	.47019	.54932		.44050	.53142
MQSTART		.45980	.52551	.44510	.45044	.46894
MQWAIT		.52957	.46477	.44000	.44951	.45862
MQREC			.47789	.44060	.55819	.48640
MQSEND			.47277	.44839	.45388	.46310
MQDONE			.47130		.43417	.45847
INDEL		.46471		.44878	.43837	.45244
INSPOOL	.54648	.54149		.45463	.44511	.49598
INWAIT	.48210	.54073			.40000	.49715
INTEXT	.47150	.55639				.49772
INCOM	.46542	.54365				.50772
Total	.51686	.49554	.50384	.44833	.45880	.49071

TABLE 5 - AVERAGE QUEUE TIMES
(Seconds/Queue)

Queue	4800 Baud	9600 Baud	Total
ALLQUES	.47967	.53186	.50162
MESQUES	.45951	.46711	.46364
COMQUES	.44278	.54482	.47088
MQSTART	.44681	.48961	.46894
MQWAIT	.45225	.47212	.46141
MQREC	.47412	.47789	.47597
MQSEND	.43681	.47277	.45227
MQDONE	.41987	.47130	.44405
INDEL	.44532	.46471	.45244
INSPOOL	.45071	.54334	.49598
INWAIT	.40000	.49862	.49715
INTEXT	.48391	.49772	.49205
INCOM	.43974	.50772	.47879
Total	.45650	.50519	.47840

TABLE 6 - SELECTED QUEUE PROCESS TIMES

Queue	Date	No. of Samples	.1	.2	.3	.4	.5	.6	.7	.8	.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0
INWAIT	5/4	410	332																								38	2	4			34
INSPOOL	5/4	629	526																							2	54	10	2	2		33
COMQUES	6/7 #1	630	462				84																					2	31	29	2	20
ALLQUE	5/10	1613	1353					92	2			2																50	8	4	11	73
MQSTART	5/10	686	568					20																		2		26	6		5	35
MQSEND	6/7 #2	450	330				58					2																	25	29	4	2
INCOM	6/7 #2	87	65				10																		2				4	4	2	
MQREC	6/7 #2	331	243				42	2																					15	24	3	2
MQDONE	6/7 #2	495	363				66																					2	26	38		

TABLE 7 - OVERALL PERCENTAGE OF QUEUE PROCESS TIME

Queue	Total No. of Samples	0.1		0.5		2.5	
		No. of Samples	Percentage	No. of Samples	Percentage	No. of Samples	Percentage
ALLQUES	11095	8831	79.594	937	8.455	1327	11.960
MESQUES	3174	2500	78.764	394	12.413	280	8.821
COMQUES	9860	7419	75.243	1141	11.572	1300	13.184
MQSTART	2924	2285	78.146	286	9.781	353	12.072
MQWAIT	2716	2120	78.055	310	11.413	286	10.530
MQREC	2302	1798	78.105	278	12.076	226	9.817
MQSEND	2068	1588	76.789	248	11.992	232	11.218
MQDONE	1757	1353	77.006	212	12.066	192	10.927
INDEL	2046	1568	76.637	244	11.925	234	11.436
INSPOOL	2044	1591	77.837	134	6.555	319	15.606
INWAIT	1477	1159	78.469	96	6.499	222	15.030
INTEXT	3349	2605	77.784	322	9.614	422	12.600
INCOM	2028	1567	77.268	166	8.185	295	14.546
Total	46840	36384	77.677	4768	10.179	5688	12.143

The percentages shown in Tables 6 and 7 are computed by dividing the number of messages processed in a specified time interval ($t<.1$, $.1\leq t<2.0$, $t>2.0$) by the number of messages. Table 7 shows the data clustered about three time values: 0.1, 0.6, and 2.6 seconds. The samples at 0.1, 0.5, 0.6, and over 2.2 seconds represent all but 165 samples out of 48,840 (99.648 percent).

IBM LINE ACTIVITY RECORDER

The data extracted from the LAR listings included the time a packet was transmitted/received, the same part of the packet shown in the PDP message trace, and whether it was an IBM read or write operation. In addition, the information contained in each packet (except idle packets) is described. The LAR results are shown in Table 8.

IBM SYSTEM STATISTICS

Any program (application or message system) that runs on the IBM produces as part of its output a minimum set of system usage statistics. The exception (EXCP) entry is the number of I/O operations. The data for the selected back-end runs are shown in Table 9 and for the stand-alone runs in Table 10.

ANALYSIS OF RESULTS

The data gathered were sufficient to indicate generally where the critical bottlenecks in the back-end system were. The results of the time and code analysis are used to discuss the perceived benefits as stated in the Introduction.

TIMING ANALYSIS

Tables 2 and 3 (message logging) define the basic problem: each DML verb takes three seconds to process. If the back-end is to be useful, this time must be dramatically reduced.

The problem will be approached from both ends: host and back-end. Table 9 shows that, for the best case (lowest elapsed time/DML verb) on the host, two ACK pairs are exchanged before the answer is sent in response to a DML verb. On receiving a verb MSBYSN (the BSC code) immediately sends an acknowledgment to the host. Thus the time between sending the verb and receiving the acknowledgment is

TABLE 8 - LAR RESULTS

24 April 1979
LAR DATA

Time(07)	Packet	R/W		Packet Information
07:51.153	2E2755...FO	R	<----	PDP ERROR STATUS
07:51.178	2E2728...0A (10)	W	---->	FIND NEXT OF SET VERB
07:51.611	2E5528	R	<----	
07:51.628	2E5529	W	---->	
07:52.025	2E5529	R	<----	
07:53.042	2E5529	W	---->	
07:53.507	2E5529...FO	R	<----	ERROR STATUS
07:53.537	2E5629...2B (43)	W	---->	OBTAIN VERB
07:53.970	2E2957	R	<----	
07:53.983	2E562A	W	---->	
07:54.380	2E2957	R	<----	
07:55.395	2E562A	W	---->	
07:55.871	2E2957...F2	R	<----	DATA RETURNED
07:55.885	2E572A	W	---->	
07:56.342	2E2958...FO	R	<----	ERROR STATUS
07:56.370	2E582A...1F (31)	W	---->	FIND OWNER OF SET VERB
07:56.803	2E2A59	R	<----	
07:56.818	2E582B	W	---->	
07:57.216	2E2A59	R	<----	
07:58.235	2E582B	W	---->	
07:58.699	2E2A59...FO	R	<----	ERROR STATUS
07:58.726	2E592B...2B (43)	W	---->	OBTAIN VERB
07:59.157	2E2B5A	R	<----	
07:59.172	2E592C	W	---->	
07:59.568	2E2B5A	R	<----	
08:00.585	2E592C	W	---->	
08:01.069	2E2B5A...F2	R	<----	DATA RETURNED
08:01.093	2E5A2C	W	---->	
08:01.551	2E2B5B...FO	R	<----	ERROR STATUS
08:01.614	2E5B2C...0A (10)	W	---->	FIND NEXT OF SET VERB
08:02.047	2E2C5C	R	<----	
08:02.065	2E5B2D	W	---->	
08:02.462	2E2C5C	R	<----	
08:03.492	2E5B2D	W	---->	
08:03.957	2E2C5C...FO	R	<----	ERROR STATUS
08:03.981	2E5C2D...0A (10)	W	---->	FIND NEXT OF SET VERB
08:04.415	2E2D5D	R	<----	
08:04.438	2E5C2E	W	---->	
08:04.834	2E2D5D	R	<----	
08:05.852	2E5C2E	W	---->	
08:06.317	2E2D5D...FO	R	<----	ERROR STATUS
08:06.368	2E5D2E...0B (11)	W	---->	FIND NEXT OF AREA VERB
08:06.800	2E2E5E	R	<----	
08:07.832	2E5D2F	W	---->	
08:07.229	2E2E5E	R	<----	
08:07.257	2E5D2F	W	---->	
08:08.721	2E2E5E...FO	R	<----	ERROR STATUS
08:08.779	2E5E2F...13 (19)	W	---->	FIND FIRST OF AREA VERB
08:09.211	2E2F5F	R	<----	
08:09.224	2E5E30	W	---->	
08:09.620	2E2F5F	R	<----	
08:10.633	2E5E30..	W	---->	
08:11.098	2E2F5F...FO	R	<---	
08:11.135	2E5F30...2B (43)	W	---->	OBTAIN VERB

TABLE 9 - IBM STATISTICS FOR BACK-END

Date	Program	ET H.M.S.MS	CPU H.M.S.MS	Core	Core (MS)	EXCP (MS)	ET (MS) H.M.S.MS	CPU (MS) H.M.S.MS
4/24	DMSUBS	0.27.50.95	0.0.03.45	44K	70K	2619	0.0.39.87	0.0.39.87
4/27	DMSUBS	0.36.39.80	0.0.04.17	44K	70K	2712	0.51.06.21	0.0.52.21
5/1	DMSUBS	0.31.09.73	0.0.04.40	44K	70K	2656	0.57.17.20	0.0.59.16
5/4	DMSUBS	0.30.11.23	0.0.03.49	44K	70K	2342	0.35.26.05	0.0.36.30
5/10	DMSUBS	0.43.01.63	0.0.03.60	44K	70K	2989	0.52.13.12	0.0.43.68
5/23	FLOAD	0.14.41.20	0.0.01.52	24K	70K	1715	0.36.39.90	0.0.33.30
6/1	FLOAD	0.14.56.18	0.0.01.36	24K	70K	1344	0.21.57.70	0.0.27.89
6/7 #1	PLANE	0.06.24.11	0.0.00.61	24K	70K	1089	0.35.43.02	0.0.16.14
6/7 #2	PLANE	0.18.09.12	0.0.00.03	26K	70K	934	0.18.15.82	0.0.14.83

TABLE 10 - IBM STATISTICS FOR STAND-ALONE

Date	Program	ET	CPU	Core	Excp
1 May	DMSUBS	0.14.38.35	0.0.4.78	92K	276
3 May	DMSUBS	0.15.47.36	0.0.4.30	92K	276

the time required to transmit forty characters (1/12 second at 4800 bps), thirty to the PDP and ten in response, plus the time spent in the IBM line driver, plus the time on the back-end, plus the time to turn the line around. The time between sending the verb and receiving the acknowledgment is .433 $\pm$ .0001 seconds from Table 8. If we ignore the IBM line driver and assume 150 milliseconds line turn around time (high for a local line), the back-end spends about .19 seconds receiving, identifying, and acknowledging a DML verb. Since the verb has not been released to the message system, only MSBYSN and the line driver are executing. This problem, which is not a major one, will be examined in the next section on code analysis.

When a verb is received, the packet is split into a command packet and a text packet. On their way to and from a TWIN these packets pass through the various queues. Tables 4 and 5 show that a packet spends an average of almost half a second in each queue, which suggests that the problem is evenly distributed throughout the message system. Tables 6 and 7, however, show that the message system is capable of processing the packets rapidly (77 percent in 0.1 second). The discrepancy between the ability to process packets and the average time to process packets can be explained only by an asynchronous process interrupting the message system and thereby freezing packets for a short time. This freezing of packets would explain the clustering of queue times around 0.6 and 2.6 seconds. The only processes executing when the message system is processing packets are the IAS operating system, MSBYSN, and the line driver.

CODE EXAMINATION

Code examination began with MSBYSN and the line driver. MSBYSN analysis showed two problem areas: protocol specification and interprocess communication.

The protocol, as specified and implemented, requires that an incoming message is not to be released to the message system until an ACK is received from the host. The sequence of actions is:

- DML verb received from host
- ACK sent to host
- ACK received from host
- packet containing DML verb released to the message system

This sequence forces a delay of about 0.4 second on the response time for each DML verb. This delay is a serious problem and requires using a single input buffer instead of a double buffer. Double buffering would release all valid packets immediately and save the 0.4 second delay time. The protocol will not allow command and text packets to be transmitted together. Thus for an OBTAIN verb the error code and the data are transmitted separately, separated by an ACK from the host. This procedure is both time consuming (almost 0.5 second) and unnecessary. Integrating the messages or having a full duplex protocol would solve the problem.

MSBYSN communicates with three other tasks; the line driver, the message system, and the message trace and logging print routine. Communication to/from the line driver and the message system is by read/write IAS system directives, since the line driver and message system are device handler tasks to IAS. Communication to the trace and logging task is by send data/receive data IAS system directives an intertask communication facility. Use of this facility results in considerable overhead for the simple transfer of data but is required by the size of the tasks. This method of communication is not serious by itself, but will impact another area as will be seen shortly.

Analysis of the line driver code revealed several problems, although none critical. The problems are:

- the cyclic redundancy check is computed in software
- character stuffing is done in software
- the line driver handles both the DQ and DU synchronous devices

It is clear from the second problem (character stuffing) that functions belonging in MSBYSN are being performed in the line driver for convenience. This integration instead of separation of functions makes changes (e.g., to a different device) much more difficult.

The problems discussed so far do not account for all the time required to process a DML verb. Examining the code did, however, show the size of the tasks and indicated a generally critical problem area: size. Size includes partition size, task size, and system size.

All the back-end software, except the IAS operating system, executes in one partition, (A partition is a block of memory reserved at sysgen time). The DMBS, TWIN, BENRC, and message system are all in the DBMS partition. It was because of partition size that only one TWIN could be resident at a time. This limitation made it impossible to test multiple jobs on the host.

An IAS handler is limited to a size of 24,576 bytes. The message subsystem packet subsystem module had to use memory management to control two in-core overlays because of the general purpose design of the message system and the memory constraint of the IAS operating system.

The message system places a heavy intra-system data transportation burden on the IAS operating system. The message system demands for buffers and nodes could be solved by sysgening a larger system.

It is the author's opinion that the conflict for system resources caused by these three size problems and their interaction constitutes the most important back-end DBMS problem. The partition and system sizes are easy to change; the task size limitation cannot be changed under IAS. Fortunately, changing the partition and system sizes should resolve the problem, since there could then be no interaction and the in-core overlays would not be a significant problem in themselves.

PROBLEM AREAS

Of the following several problems identified by the analysis, the first two are the most serious:

- The protocol specification does not allow the DML verb to be released until a succeeeding ACK/ACK pair is complete. It also will not allow command packets to be sent with text packets. Use of one of the new full duplex protocols would solve these problems.
- The partition size, task limit, and system size interact in a very detrimental manner. Rebuilding the system would solve the size problem.
- The CRC calculation now done in software should be done in hardware which is about two orders of magnitude faster.
- Character stuffing is done in software and would not have to be done at all if a new protocol were used.
- There was no clear division of functions. Part of the BSC protocol was done in the line driver, which handled two different devices. The message and packet

subsystems were integrated into one module. New networking standards and implementations, however, have made use of the layered approach with well-defined interfaces and separation of functions. This approach would make testing and modification much easier.

The question naturally arises of the response time per DML verb if all the problems were corrected. The answer is speculative. Sending 30 characters of result using a full duplex protocol at 9600 bps would require 1/16 second. Adding about 1/5 second for the DBMS provides a basis for estimating. On the basis of these figures the back-end should be able to respond to a DML verb in 1/2 to 3/4 second. Although these figures are for a single user environment, increasing the number of users would only incrementally increase the response time because the I/O time needed to process additional verbs would be small.

BACK-END VERSUS STAND-ALONE

Tables 9 and 10 show quite clearly the differences in the two systems. A single user back-end system is not as efficient in the use of memory and CPU time as a stand-alone one. When two simultaneous tasks are performed, the back-end is more efficient and becomes increasingly so for additional users.

The one area in which the stand-alone system is far superior is in the number of I/O operations (EXCP's). This is an important area and the time used could be reduced significantly on a back-end system by using a DBMS language more powerful than CODASYL. This would force more work to the back-end, but would reduce CPU time, EXCP's, and memory space on the host.

BENEFITS COMPARISON

The Introduction included a list of perceived benefits for the host when a back-end system is added. These benefits can be described for the current test back-end system and projected to that system with its problems resolved, as follows:

- The amount of disk space used definitely goes down. The space needed for addition of the message system is more than offset by the removal of the DBMS software and data base.

- The CPU time and memory required to process concurrent jobs demands that better performance be realized than when they are run separately. The use of a powerful DBMS language to reduce I/O operations would also reduce CPU time and memory space.
- Data security was analyzed in an earlier section and the back-end was shown to have much superior capabilities.
- Removing the DBMS from the host makes them independent.
- The increased host utilization which was expected to occur by linking several hosts was not explicitly tested. There was no evidence, however, that it would not be realized.

Although a stand-alone system has, in most cases, a superior response time, a back-end could, with the previously mentioned modifications and a more powerful DBMS language, provide a response time that would be more than adequate for most applications.

CONCLUSIONS AND RECOMMENDATIONS

This section will address the functions a production system should provide and the design of that system. This test implementation demonstrated the feasibility of the concept and will strongly influence the design of a production system. A properly designed and implemented back-end system could play a significant role in extending the useful life of existing machines and in fully utilizing existing resources.

Since a Back-End Data Base Management System is intended to be coupled to one or more existing machines, it is difficult to describe the best system configuration in terms of either hardware or software. It will be assumed that there are few system configuration constraints. The recommendations will therefore be oriented toward an approach involving a powerful back-end system and addressing security and off loading data management. The system will also have potential for multiple connectivity, distributed data bases, and distributed processing. Although this study did not investigate multiple connections, the potential for using them is important and they should be allowed for.

TESTING FUNCTIONS

It is important to be able to test and analyze the system both when it is delivered and later when changes are to be made. The following test functions should be available at delivery:

- A LAR-like facility to show packet arrival and transmit times.
- A vertical packet trace, i.e., a means of following a packet through the system with time stamps at functional boundaries.
- A horizontal packet trace, i.e., a time trace of all packets during processing by any function (equivalent to the histogram data used in this report).
- The ability to determine the time necessary to pass packets between functions (e.g., the time to "write" a packet from MSBYSN to the message system).
- The ability to determine the CPU time used by any function.
- A means for recording any task swapping done by the operating system.
- A means for recording the number of times and by whom a task is interrupted, and the duration of the interruption.

The time stamp should have a resolution of at least 10^{-4} seconds. The test functions should be easy to turn on or off, and the results should be printed in a readable format.

BACK-END DBMS FUNCTIONS

The back-end should allow for three types of verbs: low level (such as CODASYL), system library functions, and user defined functions. The low level provides the general capabilities upon which the other two types are constructed. The system library allows general functions, such as query or multiple retrievals, to be used by all application programs. These functions could be incorporated into the preprocessor as English-like commands for easy use. The user functions are specific to an application and are added to or deleted from a user library by the Data Base Administrator. All library functions would execute on the back-end processor.

The addition of the system and user function libraries is designed to push data management functions onto the back-end. These functions require many calls to the data base and, when executed on the back-end, result in a significant reduction in data transmission, host I/O, and CPU times because the message system is called much less frequently and fewer characters are transmitted between machines. This arrangement reduces message processing and increases data management processing on the back-end, which is exactly the trade-off desired.

Utilities should be provided that enable the addition/deletion of verb processing routines to user and system libraries. The ability to easily modify the preprocessor to accommodate new verbs should be required.

SYSTEM FUNCTIONS AND CHARACTERISTICS

The host/back-end system should be designed with the following system functions and characteristics in mind:

- Full duplex DLC protocol.
- Implementation of functions in hardware that lend themselves to such implementation, such as CRC calculation.
- Hierarchical peer level implementation of the message system, i.e., hierarchial design on each machine with each level having a corresponding (or peer) level on the other machine.
- Well documented and easy to use test functions.
- Implementation where possible of functions in dedicated CPU's. This approach allows for expansion and updating as new technology becomes available. Dedicated CPU's require little overhead for operating systems or task switching and provide efficient concurrent processing.
- DBMS utilities to provide for maintenance and control of the data base.
- General purpose message system to handle future expansion and connectivity.
- The message system as an operating system utility and not as another user program. The message system would be most efficiently implemented if designed from the start as a privileged task.
- A constantly running message system. It should be quiescent when no host job is communicating with the back-end and should automatically restart on call by an application program (for output) or the line driver (for input). The protocol could allow an active line to be inactive for long periods of time.

USER ENVIRONMENT

The back-end system must be transparent to a running application program to allow conversion of existing programs with a minimum of effort. If the DBMS utilities run on the host, then the program development cycle looks unchanged to the user. There are some drawbacks, however:

- Precompilers must be developed for each type of host machine.
- The schema and subschemas must be kept on the host or transferred to the host for each precompile. If kept on the host, they must be updated whenever they are modified on the data base, which is obviously more complicated in a multiply-connected system.
- Security is lessened.
- Only the DBMS is off-loaded from the host, which defeats one of the basic objectives of the back-end approach.

If the DBMS utilities are not to be run on the host, they could be run on a system with access to both the host and the data base. The concept of a Program Development Machine (PDM) was developed for this purpose as a separate processor that contains:

- a text editor
- FORTRAN and COBOL compilers
- a message system
- a rudimentary file transfer capability

With these capabilities a user can enter the program, precompile, compile and debug--all on the same PDM. The precompiled FORTRAN/COBOL source can then be transferred to the host for compilation and production runs. A prototype PDM configuration is shown in Figure 13.

In addition to its benefits, the PDM approach has four obvious drawbacks:

- increased hardware costs
- increased software costs. The editor and compilers must be purchased and the file transfer capability, using the message system, developed. If the PDM and back-end are the same type of machine, the precompilers can be purchased.
- more is required of the user. No matter how easy it is to use, the machine is different.

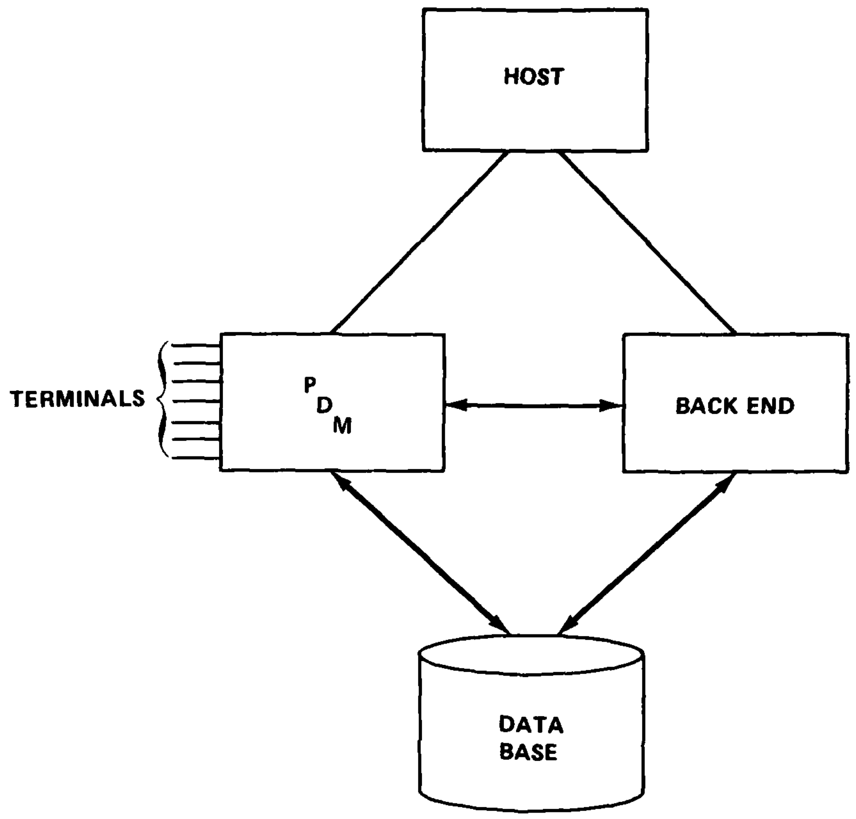


Figure 13 - Back-End with Program Development Machine

- increased line use when the precompiled source is transferred to the host.

The benefits, however, are impressive:

- Much of the program development effort is not done on the host.
- There are no DBMS utilities on the host.
- All data base values and structure are centrally located and access is controlled.
- Disk storage space on the host is reduced.
- Sharing a data base among multiple hosts is made easier.
- The PDM provides a friendly development environment and not a controlled production system.

• The host and back-end are independent of each other and are coupled only by the message system. Adding or changing one machine has no effect on the other machine.

ADDITIONAL BENEFITS

These additional benefits are received because of the generality of the message system. Although this particular message system need not be used, the functions it contains should be provided. The additional benefits are:

• intertask communication on the host. This feature is incorporated in the design of modern high level languages, such as ADA.

• basis for file transfer between host and PDM or other hosts.

• foundation upon which networks, distributed processing, and distributed data bases may be developed.

APPLICATION PROGRAM REQUIREMENTS

A stand-alone program should look the same as a back-end program. The DML verb statements are the same and any additional functions performed on the back-end are just subroutine calls. The additional functions are, however, off-loaded from the host and the resultant application program is smaller. This off-loading also reduces the overhead of the message system on the host.

BACK-END SYSTEM DESIGN

The essential elements of a complete system design are dedicated CPU's and a method for interconnection. Since the dedicated CPU's will motivate the interconnection issue, they will be discussed first.

The message system can be functionally divided into six parts: the message system operators (MSO) interface, the message subsystem, the packet subsystem, the routing subsystem, and protocol subsystem, and the line drivers. These functions are then grouped for CPU assignment as follows:

• the MSO and message subsystem execute in the PDM and back-end processors.

• the packet subsystem and routing sybsystem execute in the front-end processor (FEP).

- the protocol subsystem and line driver execute in a protocol processor (PP). The PP, FEP, PDM, and back-end processors are connected via high speed data links (e.g., bus interface or DMA device). To minimize communication costs and repetition of software only an FEP, through a PP, will connect to a host. A single processor configuration is shown in Figure 14. If a significant reduction in the amount of data is achieved, then a 9600-bps connection between the host and the PP would be sufficient. If this cannot be done, a higher speed connection will be needed. For co-located machines connection costs are not high and lines of at least 50 Kilubytes per second could be used.

A site may contain multiple hosts all of which are saturated. A single back-end system could handle this situation as shown in Figure 15. The use of the PDM requires only the message system to be developed for each host.

It may not be feasible or desirable to have two data bases active on the same back-end machine. Thus two back-end machines would be needed, but only PDM as shown in Figure 16. Again, the separation of functions and use of multiple CPU's make this implementation a straight-forward extension of the one shown in Figure 14.

The system described here can be easily expanded to accommodate the most general case, multiple hosts and multiple back-ends, as shown in Figure 17. The level of sophistication that can be built into the back-end system depends only on the implementor's requirements. The PP's shown in Figure 17, for example, can link dissimilar back-end systems. Such a system design allows for dynamic growth with a minimum of effort.

SUMMARY

The test back-end system showed the feasibility of the concept but suffered from slow response time. Some significant problems with the system were identified and remedies were suggested.

Use of the system, the problems encountered, and system analysis led to the proposal of two basic additions for a follow-on system:

- Use of multiple CPU's
- Use of the PDM

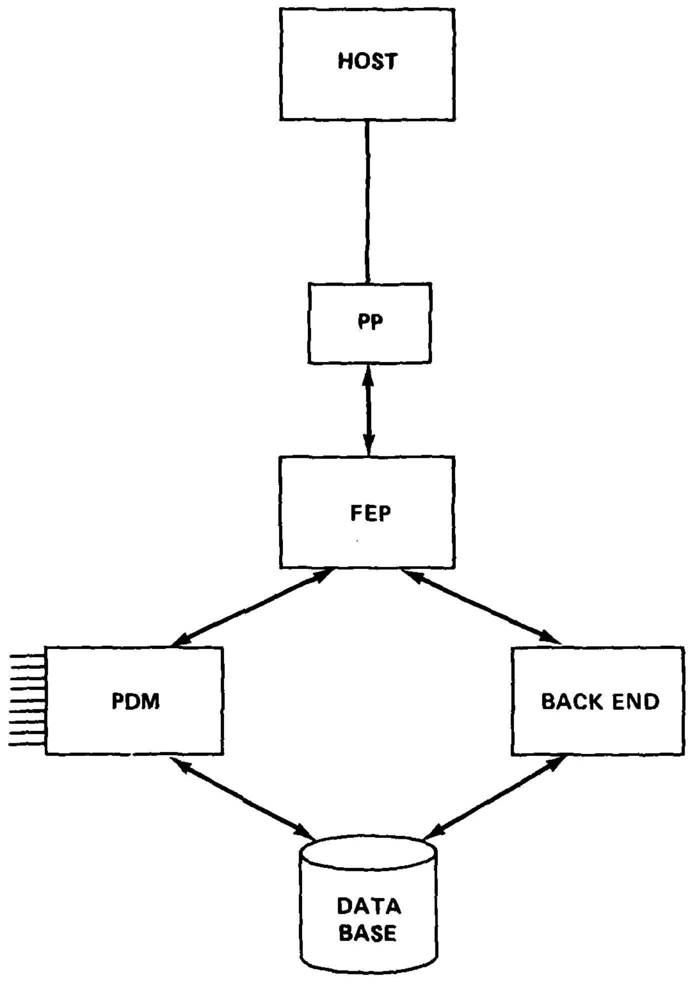


Figure 14 - Singly Connected Back-End System Configuration

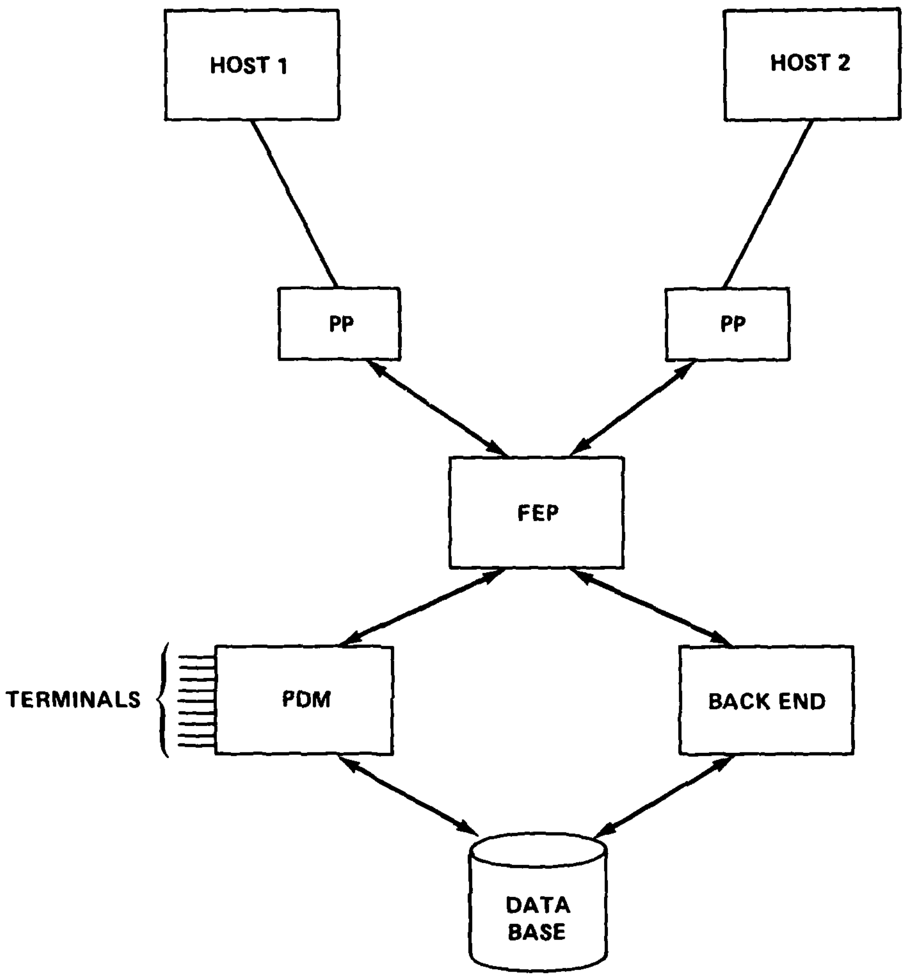


Figure 15 - Multiple Host Back-End System Configuration

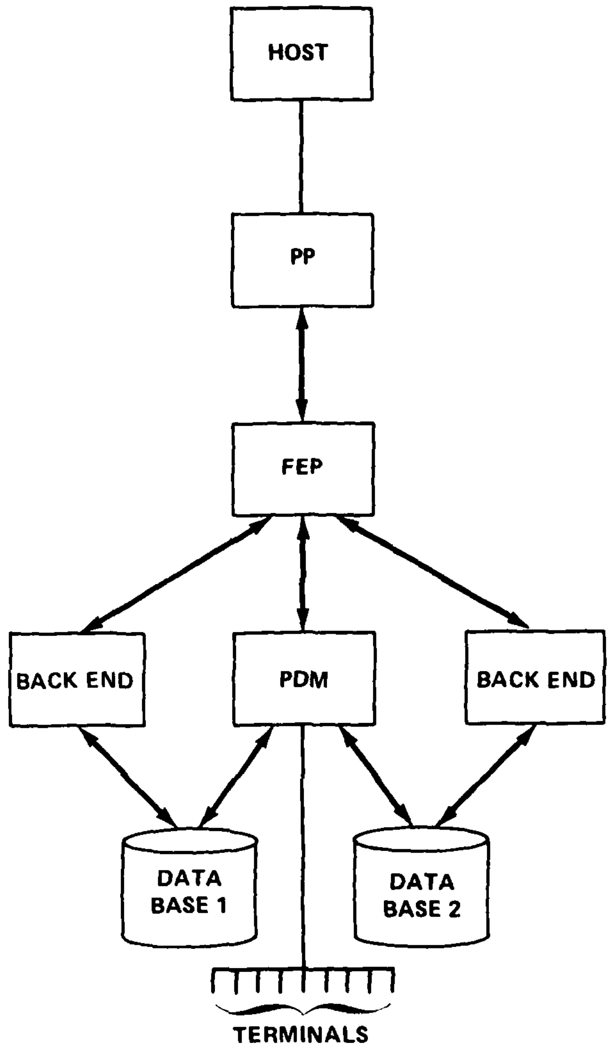


Figure 16 - Single Host Multiple Back-End System Configuration

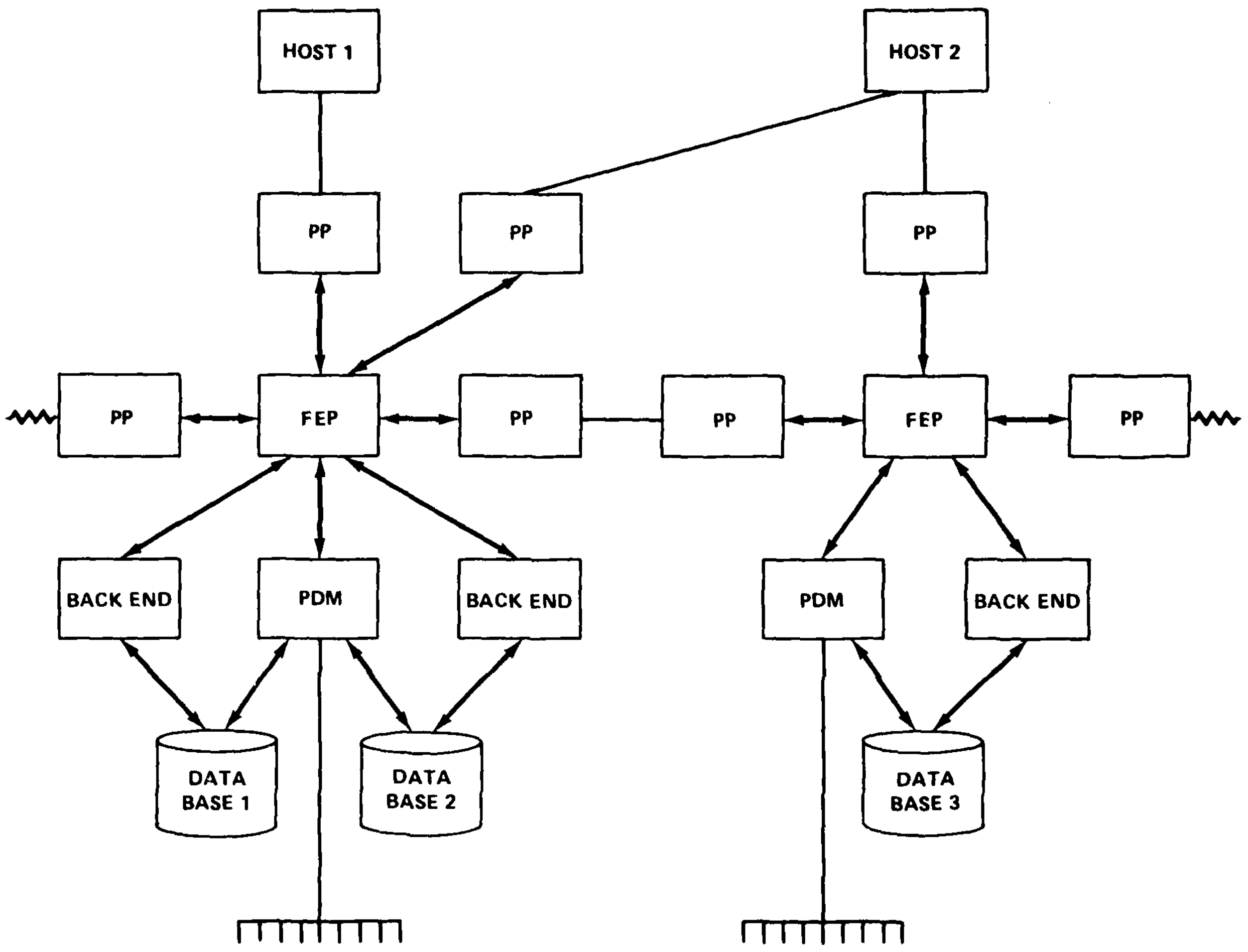


Figure 17 - Multiply-Connected Back-End System Configuration

The use of multiple CPU's has four basic benefits:

- Efficient function performance through use of a dedicated CPU.
- Reduced hardware costs if the system is expanded as shown in Figure 15.
- Easy growth of the system including increasing system power or connecting to other systems.
- Isolated functions and increased security.

The use of the PDM has the following benefits:

- Off-loading much of the program development work from the host.
- Increased security.
- Reduced host disk usage.
- Off-loading all of the DBMS from the host.
- Reduced dependence between the back-end and the host.

The proposed system configuration has the potential to meet the present and future needs of any site.

The back-end system is a viable means of reducing host resource usage, increasing data security, providing an inter- and intra-computer communication system, and providing a powerful program development tool to the users at a reasonable cost.

ACKNOWLEDGMENTS

This work was performed under the supervision of Mr. Clarence Godfrey of DTNSRDC who also provided the coordination among the various agencies involved.

REFERENCES

1. Diffie, W. and M. E. Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory, Vol. IT-22, pp. 644-654 (Nov 1976).

2. COBOL Journal of Development, Canadian Government (Supply and Services, Canada) (1978).

INITIAL DISTRIBUTION

Copies	
12	NAVSUP
	1 SUP 01B (J. Schanzenbach)
	1 SUP 09H (J. Roberts)
	10 SUP 0431 (G. Bernstein)
12	DTIC

CENTER DISTRIBUTION

Copies	Code	Name
1	18/1808	G. Gleissner
2	1809.3	D. Harris
1	182	A. Camara
1	1824	J. Carlberg
1	1826	L. M. Culpepper
1	1826	L. K. Meals
20	1828	M. Wallace
1	1828	C. Godfrey
1	184	J. Schot
1	185	T. Corin
1	187	M. Zubkoff
1	189	G. Gray
10	5211.1	Reports Distribution
1	522.1	Library (C)
1	522.2	Library (A)