Virtual Machine/
Enterprise Systems Architecture

IBM

TCP/IP Function Level 320
Diagnosis Guide

Version 2 Release 4.0

GC24-5851-01

Virtual Machine/
Enterprise Systems Architecture

IBM

TCP/IP Function Level 320
Diagnosis Guide

Version 2 Release 4.0

GC24-5851-01

Note!

Before using this information and the product it supports, read the information under "Notices" on page vii.

| **Second Edition (July 1999)**

| This edition applies to the IBM® Transmission Control Protocol/Internet Protocol Feature for VM/ESA® (TCP/IP Function Level 320),
| program number 5654-030 and to all subsequent releases and modifications until otherwise indicated in new editions.

| This edition replaces GC24-5851-00.

Contents

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

```
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.
```

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

```
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan
```

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes to the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

```
IBM Corporation
Mail Station P300,
522 South Road
Poughkeepsie, NY 12601-5400
U.S.A.
Attention: Information Request
```

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities on non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX	AS/400
BookManager	Common User Access
C/370	DB2
ES/9000	Enterprise System/9000
ES/9370	ESCON
GDDM	IBM
Language Environment	Micro Channel
NetView	Office Vision/VM
SP	OpenEdition
OS/2	PC Network
PROFS	RACF
S/390	SAA
VM/ESA	VM/XA
Database 2	OS/390

Other company, product, and service names may be trademarks or service marks of others.

Preface

This *TCP/IP Diagnosis Guide* is intended to provide information for diagnosing problems occurring in IBM's* Transmission Control Protocol/Internet Protocol (TCP/IP) networks. This book can also help you determine whether a specific problem is a result of the TCP/IP Feature for VM/ESA implementation.

Programming Interface Information

This book documents information NOT intended to be used as Programming Interfaces of VM/ESA.

Warning: Do not use this Diagnosis, Modification, or Tuning Information as a programming interface.

Who Should Read This Book

This book is intended to be used by system programmers or TCP/IP administrators for diagnosing problems. You should use this book to:

- Analyze a problem in a VM TCP/IP implementation
- Classify the problem as a specific type

You should be familiar with TCP/IP and the protocol commands to use this book.

How To Use This Book

You should read this book when you want to diagnose and report problems that can occur in TCP/IP networks.

How This Book is Organized

Chapter 1, "Diagnosis Overview," describes basic problem determination steps. A flow diagram shows the process to follow when determining problems.

Chapter 2, "Problem Identification," describes problem categories and the structure of service support to help you solve your problems.

Chapter 3, "TCP/IP VM Structures and Internetworking Overview," describes the structures of the TCP/IP implementation for VM and an overview of Internetworking.

Chapter 4, "Server Initialization," describes the mechanism used to start each TCP/IP server.

Chapter 5, "TCP/IP Procedures," describes TCPIP internal procedures, queues, and activities and input/output functions.

Chapter 6, "Diagnosing the Problem," provides information about diagnosing TCP/IP problems. The chapter also provides a systematic approach to solving TCP/IP problems.

Chapter 7, “TCP/IP Traces,” describes how to activate traces and direct trace output. The chapter also describes single and group processes.

Chapter 8, “FTP Traces,” describes how to activate and interpret File Transfer Protocol (FTP) traces.

Chapter 9, “Simple Mail Transfer Protocol Traces,” describes how to activate and interpret Simple Mail Transfer Protocol (SMTP) traces.

Chapter 10, “RPC Programs,” describes how to activate and interpret Remote Procedure Call (RPC) traces.

| Chapter 11, “RouteD Diagnosis,” describes how to activate, debug, and interpret
| RouteD traces, and diagnose problems.

Chapter 12, “Network File System,” describes how to debug NFS Server problems plus interpret NFS traces.

Chapter 13, “Remote Printing Traces,” describes the tracing capabilities available in the client and server functions provided with the Remote Printing implementation in TCP/IP for VM.

Chapter 14, “Remote Execution Protocol Traces,” describes the tracing capabilities available in the client and server functions provided with the Remote Printing implementation in TCP/IP for VM.

Chapter 15, “TFTP Client Traces,” describes how to activate and interpret TFTP client traces.

Chapter 16, “TFTPD Traces,” describes how to activate and interpret TFTPD traces.

Chapter 17, “BOOT Protocol Daemon (BOOTPD) Traces,” describes how to activate and interpret BOOTPD traces.

Chapter 18, “Dynamic Host Configuration Protocol Daemon (DHCPD) Traces,” describes how to activate and interpret DHCPD traces.

Chapter 19, “Hardware Trace Functions,” describes how to activate and interpret traces on PCCA and CETI devices. The chapter also provides samples of Channel Control Word (CCW) traces.

Appendix A, “Return Codes,” describes TCP/IP return codes.

Appendix B, “Related Protocol Specifications,” describes the TCP/IP RFCs.

This book also includes a glossary, a bibliography, and an index.

How Numbers Are Used in This Book

In this book, numbers over four digits are represented in metric style. A space is used rather than a comma to separate groups of three digits. For example, the number sixteen thousand, one hundred forty-seven is written 16 147.

Where to Find More Information

The "Glossary" on page 219, defines terms used throughout this book associated with TCP/IP communication in an internet environment.

For more information about related publications, see "Bibliography" on page 237.

Table 1 shows where to find specific information about TCP/IP for VM applications, functions, and protocols.

Table 1 (Page 1 of 2). Usage of TCP/IP Version 2 for VM Applications, Functions, and Protocols

Applications, Functions, and Protocols	**Topic**	**Book**
BOOTP Daemon (BOOTPD)	Setting up the Server	*Planning and Customization* *Program Directory*
	Usage	*Planning and Customization*
	Commands	*Planning and Customization*
DHCP Daemon (DHCPD)	Setting up the Server	*Planning and Customization* *Program Directory*
	Usage	*Planning and Customization*
	Commands	*Planning and Customization*
eXternal Data Representation (XDR)	Usage	*Programmer's Reference*
File Transfer Protocol (FTP)	Setting Up the Server	*Planning and Customization* *Program Directory*
	Usage	*User's Guide*
	Commands	*User's Guide*
Kerberos	Setting Up the Server	*Planning and Customization* *Program Directory*
	Usage	*Programmer's Reference*
	Commands	*User's Guide*
NETSTAT	Usage	*User's Guide*
Network Computing System (NCS)	Setting Up NCS	*Planning and Customization* *Program Directory*
	Usage	*Programmer's Reference* *User's Guide*
Network File System (NFS)	Setting Up the Server	*Planning and Customization* *Program Directory*
	Usage	*User's Guide*
OSF/Motif**	Usage	*Programmer's Reference*
PING	Usage	*Planning and Customization* *Program Directory* *User's Guide*
Portmapper**	Setting Up the Server	*Planning and Customization* *Program Directory*
	Usage	*Programmer's Reference* *User's Guide*

Table 1 (Page 2 of 2). Usage of TCP/IP Version 2 for VM Applications, Functions, and Protocols

Applications, Functions, and Protocols	**Topic**	**Book**
Remote Execution Protocol (REXEC)	Setting Up the Server	*Planning and Customization* *Program Directory*
	Usage	*User's Guide*
Remote Printing	Setting Up the Server	*Planning and Customization* *Program Directory*
	Usage	*User's Guide*
Remote Procedure Calls (RPC)	Usage	*Programmer's Reference*
Resolver	CMS Program Interface	*Programmer's Reference*
	Configuration Parameters	*Planning and Customization* *Program Directory*
RouteD	Setting Up the Server	*Planning and Customization* *Program Directory*
RPCGEN command	Usage	*Programmer's Reference*
Simple Mail Transfer Protocol (SMTP)	Setting Up the Server	*Planning and Customization* *Program Directory*
	Usage	*User's Guide*
	Interface to SMTP	*Programmer's Reference*
Simple Network Management Protocol (SNMP)	Setting Up the Server and Agent	*Planning and Customization* *Program Directory*
	Usage	*Planning and Customization* *Program Directory* *User's Guide*
SNMP Distributed Program Interface (DPI)	Usage	*Programmer's Reference*
Socket Calls	Usage	*Programmer's Reference*
Telnet	Setting Up the Server	*Planning and Customization* *Program Directory*
	Usage	*User's Guide*
	Commands	*User's Guide*
Trivial File Transfer Protocol (TFTP)	Usage	*User's Guide*
	Commands	*User's Guide*
Trivial File Transfer Protocol Daemon (TFTPD)	Setting up the Server	*Planning and Customization* *Program Directory*
	Usage	*Planning and Customization*
	Commands	*Planning and Customization*
X Window System	Usage	*Programmer's Reference*
X Window System GDDM® Support	Setting Up the Interface	*Planning and Customization* *Program Directory*
	Usage	*User's Guide*

How the Term “internet” Is Used in This Book

In this book, an internet is a logical collection of networks supported by routers, gateways, bridges, hosts, and various layers of protocols, which permit the network to function as a large, virtual network.

Note: The term “internet” is used as a generic term for a TCP/IP network, and should not be confused with the Internet, which consists of large national backbone networks (such as MILNET, NSFNet, and CREN) and a myriad of regional and local campus networks worldwide.

Sending Comments for This Publication

You can send comments using the form in the back of this publication, or use electronic mail or FAX. Please include:

- Publication title
- Order number (including suffix)
- Page, section title, or topic

The electronic addresses are:

Format	Address
Internet	pubrcf.vnet.ibm.com
IBM Mail	USIB2L8Z@IBMMAIL
IBMLink™	GDLVME(PUBRCF)
FAX	(607)752-2327

Your comments will help improve our information. Thank you.

Understanding Syntax Diagrams

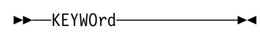
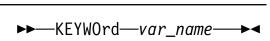
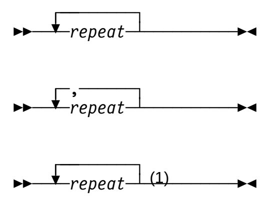
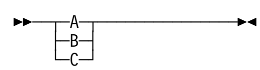
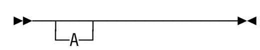
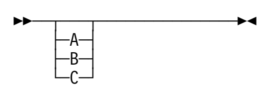
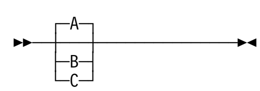
This section describes how to read the syntax diagrams in this book.

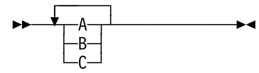
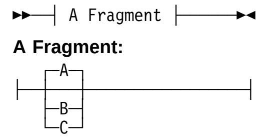
Getting Started: To read a syntax diagram, follow the path of the line. Read from left to right and top to bottom.

- The ►►── symbol indicates the beginning of a syntax diagram.
- The ──► symbol, at the end of a line, indicates that the syntax diagram continues on the next line.
- The ►── symbol, at the beginning of a line, indicates that a syntax diagram continues from the previous line.
- The ──►◄ symbol indicates the end of a syntax diagram.

Syntax items (for example, a keyword or variable) may be:

- Directly on the line (required)
- Above the line (default)
- Below the line (optional).

Syntax Diagram Description	Example
Abbreviations: Uppercase letters denote the shortest acceptable abbreviation. If an item appears entirely in uppercase letters, it cannot be abbreviated. You can type the item in uppercase letters, lowercase letters, or any combination. In this example, you can enter KEYWO, KEYWOR, or KEYWORD in any combination of uppercase and lowercase letters.	
Symbols: You must code these symbols exactly as they appear in the syntax diagram.	* Asterisk : Colon , Comma = Equal Sign - Hyphen () Parentheses . Period
Variables: Highlighted lowercase items (*like this*) denote variables. In this example, *var_name* represents a variable you must specify when you code the KEYWORD command.	
Repetition: An arrow returning to the left means that the item can be repeated. A character within the arrow means you must separate repeated items with that character. A footnote (1) by the arrow references a limit that tells how many times the item can be repeated.	**Note:** 1 Specify *repeat* up to 5 times.
Required Choices: When two or more items are in a stack and one of them is on the line, you *must* specify one item. In this example, you must choose A, B, or C.	
Optional Choice: When an item is below the line, the item is optional. In this example, you can choose A or nothing at all. When two or more items are in a stack below the line, all of them are optional. In this example, you can choose A, B, C, or nothing at all.	
Defaults: Defaults are above the line. The system uses the default unless you override it. You can override the default by coding an option from the stack below the line. In this example, A is the default. You can override A by choosing B or C.	

Syntax Diagram Description	Example
Repeatable Choices: A stack of items followed by an arrow returning to the left means that you can select more than one item or, in some cases, repeat a single item. In this example, you can choose any combination of A, B, or C.	
Syntax Fragments: Some diagrams, because of their length, must fragment the syntax. The fragment name appears between vertical bars in the diagram. The expanded fragment appears in the diagram after a heading with the same fragment name. In this example, the fragment is named “A Fragment.”	

Summary of Changes

This section describes the technical changes made in this edition of the book and in previous editions. For your convenience, the changes made in this edition are identified in the text by a vertical bar (|) in the left margin. This edition may also include minor corrections and editorial changes that are not identified.

How to Obtain Previous Editions of This Book

Previous editions of this book and other books in the VM/ESA library can be ordered using the order numbers listed in the *VM/ESA: General Information* manual. That book lists the order numbers and suffixes for VM/ESA books, as well as certain related books, for currently supported VM/ESA releases. When ordering a previous edition of any book, it is important to specify the correct order number suffix.

Summary of Changes for TCP/IP Function Level 320

The following enhancements have been made to the book:

- Chapter 11, “RouteD Diagnosis,” has been improved to include information on incoming and outgoing datagram processing and generation. Also more detailed problem diagnosis, tracing output, and debug information, has been included.
- Miscellaneous service updates were added since the previous release.

Chapter 1. Diagnosis Overview

To diagnose a problem suspected to be caused by TCP/IP for VM, you first identify the problem, then determine if it is a problem with TCP/IP, and, finally, if it is a problem with TCP/IP, gather information about the problem so that you can report the source of the problem to the appropriate IBM service support group. With this information available, you can work with service support representatives to solve the problem. The object of this book is to help you identify the source of the problem.

Figure 1 on page 2 summarizes the procedure to follow to diagnose a problem. The text following the figure provides more information about this procedure.

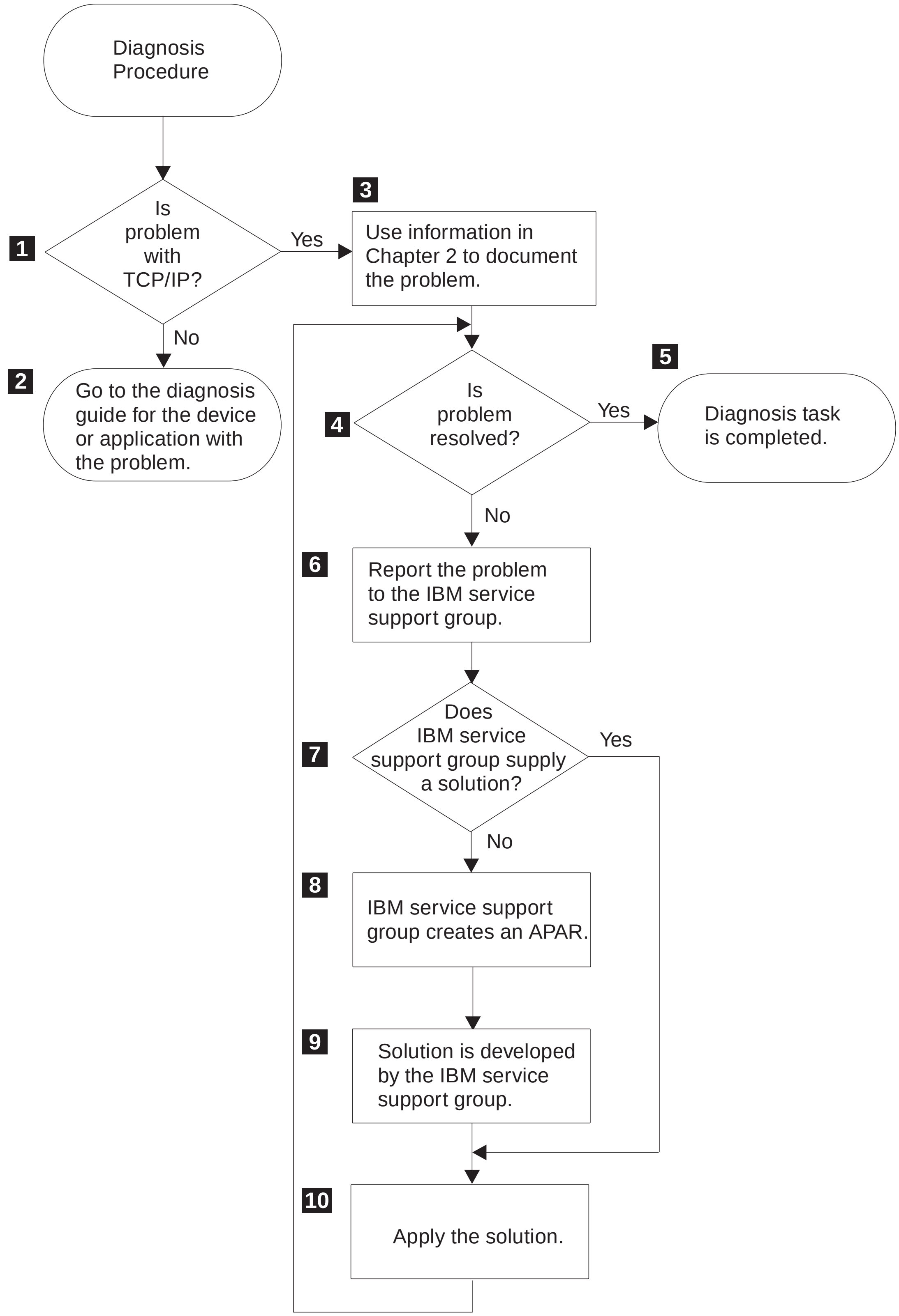


Figure 1. Overview of the Diagnosis Procedure

1 Determine if the source of the problem is TCP/IP.

Various messages outputed to the console, together with alerts and some diagnostic aids provide information that helps you to find the source of a problem. If the problem is with TCP/IP, go to Step **3**; otherwise, go to Step **2**.

2 **Check appropriate books.**

Refer to the diagnosis guide of the hardware device or software application that has the problem.

3 **Gather information.**

Refer to Chapter 2, "Problem Identification," for a detailed explanation of diagnostic procedures and how to collect information relevant to the problem.

4 **Try to solve the problem.**

If you can solve the problem, go to Step **5**; otherwise, go to Step **6**.

5 **The diagnosis task is completed.**

The problem has been solved.

6 **Report the problem to service support.**

After you have gathered the information that describes the problem, report it to service support. If you are an IBMLINK user, you can perform your own RETAIN® searches to help identify problems. Otherwise, a representative uses your information to build keywords to search the RETAIN database for a solution to the problem. The object of this keyword search using RETAIN is to find a solution by matching the problem with a previously reported problem.

| You can also visit the VM TCP/IP homepage to view PSP as well as FAQ
| information at *http://www.ibm.com/s390/vm/related/tcpip/*

7 **Work with support representatives.**

If a keyword search matches a previously reported problem, its solution might also correct the problem. If so, go to Step **10**. If a solution to the problem is not found in the RETAIN database, the service support representatives will continue to work with you to solve the problem. Go to Step **8**.

8 **Create an APAR.**

If service support does not find a solution, they may create an authorized program analysis report (APAR) on the RETAIN database.

9 **A solution is developed by the support personnel.**

Using information supplied in the APAR, service support representatives determine the cause of the problem and develop a solution for it.

10 **Apply the solution.**

Apply the corrective procedure supplied by the support personnel to correct the problem. Go to Step **4** to verify that the problem is corrected.

Chapter 2. Problem Identification

This chapter explains the categories that best describe a problem you might have with TCP/IP. This chapter also describes how you can use Service Support and it's indexed database (RETAIN) to find the solution to your problem. You should review this chapter before contacting any service support to help expedite a solution to your problem.

Categories that Help Identify the Problem

There are seven general problem categories:

- Abend
- Message
- Loop
- Wait State
- Incorrect Output
- Performance
- Documentation.

For each category, this section provides you with:

- A description of the category
- A list of the documentation to be gathered
- Directions for preparing your findings and providing them for further service support.

Problems that are related to installation, configuration, and general performance should first be pursued through your marketing branch office. They have access to facilities such as HONE, EQUAL, and the regional area Systems Centers, which may be able to provide a resolution to the problem. The Program Directory and the Preventive Service Planning (PSP) facility are also valuable sources of information for these types of problems. PSP bucket information can be viewed on the TCP/IP for VM home page at *http://www.ibm.coms390/vm/related/tcpip/*

In addition to the general categories previously listed, the following keywords can be used to describe problems associated with TCP/IP. These keywords are used to perform inquiries in RETAIN and in the licensed program, INFO/SYS:

- CLEAR/RESET
- DIAG/DIAGNOSTIC
- LAN
- LOCKED/HANG/HUNG
- RECFMS
- REJECT/FRMR
- SENSE
- INOP
- ETHERNET
- TOKEN-RING
- User ID names of server virtual machines

Abend

An abend occurs when TCP/IP unexpectedly terminates execution. In addition to TCP/IP abends, Pascal and C runtime routines can abend.

An execution error in the Pascal runtime produces output similar to that shown in Figure 2 on page 6. The compile module is TCQUEUE and AMPX messages are Pascal runtime errors.

```
AMPX036I Assertion failure checking error
      TRACE BACK OF CALLED ROUTINES
   ROUTINE                  STMT AT ADDRESS IN MODULE
   PREPENDENVELOPE            7    000AAC02   QUEUES
   FROM1822                  88    000EA58A   FROM1822
   SCHEDULER                 49    000BB5FC   SCHEDULER
   <MAIN-PROGRAM>             5    00020130   TCPIP
   VSPASCAL                        001103E2
```

Figure 2. Pascal Execution Error

For more information about Pascal execution errors, see the following books:

- *VS Pascal Applications Programming Guide*
- *VS Pascal Language Reference*.

Gather the Information

Gather the following documentation for your abend problem:

- TCP/IP dump (see guidelines below)
- Client or server dump, if applicable.

You might also need to gather the following documentation:

- TCP/IP configuration
- Console listing
- TCPIP DATA file
- TCPIP PROFILE file
- Channel control word (CCW) trace with data
- TCP/IP trace.
- Customized DTCPARMS file
- RSU Service Level

Document the Problem

To determine if the abend is related to TCP/IP, look at your TCP/IP dump or console log.

Message

The message problem category describes a problem identified by a message. If the message starts with AMPX, the error is caused by an abend in the Pascal runtime. For more information about Pascal execution errors, see “Abend” on page 6.

Gather the Information

Gather the following documentation for your message problem:

- Console log

You might also need to gather the following documentation:

- Host CCW trace
- Virtual Machine TCP/IP dump
- TCP/IP trace.

Document the Problem

To prepare a message problem report, follow these steps:

1. Write down the following:
   - The operation you tried to perform
   - The results you expected
   - The results you received.
2. Write down the entire content of the message or messages, including the message identifier.
3. Give this information to your service support person when reporting your problem.

Loop

If an operation, such as a message or printed output, repeats endlessly, TCP/IP could be in a loop. Some indicators of a loop problem are:

- Slow response time
- No response at all
- Inordinately high CPU utilization by TCP/IP.

Gather the Information

Gather the following documentation for your loop problem:

- TCP/IP dump (see guidelines below)
- Branch Trace if appropriate.

You might also need to gather the following documentation:

- Configuration files for TCP/IP
- TCPIP DATA file
- TCPIP PROFILE file
- CCW trace
- TCP/IP trace.

Document the Problem

To prepare the loop problem report, complete the following steps:

1. Record the circumstances of the loop that indicate you have a problem.
2. Use the addresses obtained from the branch trace to locate routine name or names, so you can determine where the loop occurs.
3. Contact the IBM service support group to report your problem. Provide the following information:
   - The symptoms that indicate you have a loop problem

- The maintenance level of your TCP/IP
- The contents of the branch trace
- The routine name or names where the loop occurs. This may be obtained from a formatted dump.

Wait State

If TCP/IP applications appear to hang and connected hosts report link time-outs on their end, TCP/IP could be in a wait state. Some indicators of a wait state problem are:

- Application programs cannot function or terminate
- Link time-outs are observed on connected hosts
- No communication with system console is possible
- No CPU utilization by TCP/IP is observed
- No response at all
- Traffic ceases through the network connections.

Gather the Information

Gather the following documentation for your wait state problem:

- TCP/IP dump (see guidelines below)
- Dump of the client or server virtual machine if appropriate.

You might also need to gather the following documentation:

- Configuration files for TCP/IP
- TCPIP DATA file
- TCPIP PROFILE file
- Virtual PSW value for the TCP/IP virtual machine
- Console log
- TCP/IP trace of events prior to the wait state occurring.

Document the Problem

To prepare the wait state problem report, complete the following steps:

1. Record the circumstances leading up to the wait state condition.
2. Use the module loadmap or the address portion of the virtual PSW value to determine the routine name where the wait state is occurring.
3. Contact the IBM Support Center to report your problem. Provide the following information:
   - The symptoms that indicate you have a wait state problem
   - The program levels where the wait state occurs
   - The contents of any traces activated at the time the problem occurred
   - The routine name indicated by the address portion of the PSW.

Incorrect Output

A TCP/IP incorrect output problem, such as missing, repeated, or incorrect data, is an unexpected result received during regular network operation. Incorrect output is the broadest problem category, and includes some of the following problems:

Problem	Description
Activate Failure	The inability to establish a connection with the device.
Deactivate Failure	The inability to end a connection that was established with the device.
Load Failure	Any problem that occurs during initialization.
Dump Failure	Any problem that causes the storage contents of TCP/IP to be dumped or a Pascal trace back.
Device Failure	The inability of a device to continue communication using TCP/IP.

Gather the Information

Gather the following documentation for your incorrect output problem:

- The operation you tried to perform
- The results you expected
- The results you received.

You might also need to gather the following documentation:

- TCP/IP dump (see guidelines below)
- CCW trace
- The contents of any traces activated at the time of problem
- Console log

Document the Problem

Incorrect output problems are often caused by definition errors during TCP/IP generation. Before you contact the IBM Support Center to report your problem, check that all statements and their keywords were correctly specified for your system during the generation process. After you confirm that all generation definitions were correctly specified:

1. Prepare a description of the following:
   - The operation you tried to perform
   - The results you expected
   - The results you received.
2. Give this information to the IBM Support Center when you call to report your problem.

Performance

A performance problem is characterized by slow response time or slow throughput, which can be caused by congestion in the network or a malfunction of an interface. When you suspect that you have a performance problem, gather as much information as possible about your system before and during the poor performance times.

Performance problems are normally caused by:

- Over-utilization of the host
- Inappropriate prioritization of an application program within the host
- Over-utilization of the communication interface
- Malfunction in the host, communication controller, or network.

Gather the Information

Gather the following documentation for your performance problem:

- The operation you tried to perform
- The results you expected
- The results you received
- TCP/IP configuration files

You might also need to gather the following documentation:

- TCP/IP dump (see guidelines below)
- Console log
- CCW trace
- TCP/IP trace.

Document the Problem

To prepare a performance problem report:

1. Write a description of the following:
   - The operation you tried to perform
   - The results you expected
   - The results you received.
2. Record any other characteristics about your operating environment during the time of the performance problem. Some examples of these characteristics are:
   - The time of day that the poor performance occurred.
   - Any unique application programs that were running at the time of the problem.
   - The physical configuration of your network, especially the LAN interfaces or the number of virtual circuits, such as X.25, involved.
   - Any modifications made to your operating system, input/output (I/O) generation, or the connection interface, such as virtual circuits for X.25 or the local area network (LAN) configuration for LANs.
3. Check the console for messages.

Documentation

A TCP/IP documentation problem is defined as incorrect or missing information in any of the TCP/IP books.

If the error interferes with TCP/IP operation, report the problem to your service support. However, for comments or suggestions on the content of a TCP/IP book, use the Readers' Comment Form located at the back of the book. An e-mail address is also provided for your convenience.

Gather the Information

Gather the following information for your documentation problem:

- The name and order number of the IBM publication in error
- The page number of the error
- The description of the problem caused by the error.

Document the Problem

Give the following information to your service support personnel when you report your problem:

- The order and revision number of the book that contains the error.

  The order and revision number appear on the front cover and title page of the book in the form *xxxx-xxxx-n*. The *xxxx-xxxx* is the order number and *n* is the revision number.

- Page numbers, figure numbers, chapter titles, section headings, and any other information that pinpoints the location of the text that contains the error.
- A description of the problem caused by the documentation error.

Guidelines for Machine Readable Documentation

If, after talking to the Level 2 Support Center representative about a problem, it is decided that documentation should be submitted to the TCP/IP support team, it may be more convenient for the customer and/or the TCP/IP support team that documentation be submitted in machine readable form (that is, on tape) or else sent over the network. Machine readable documentation can be handled most efficiently by the IBM support person if it conforms to the following guidelines when creating the tape (or tapes).

When preparing machine readable documentation for submission in a VM environment, the following guidelines should be followed:

1. Dumps and traces should be submitted on tape.

   - For dumps:

     The generation of dumps for the TCP/IP virtual machine (for program checks) is controlled by a parameter on the ASSORTEDPARMS statement in the PROFILE TCPIP file. Two possible formats are supported:

     – **CPDUMP** - tells TCP/IP to generate a dump using the CP DUMP command.

     – **VMDUMP** - tells TCP/IP to generate a dump using the CP VMDUMP command.

     If neither of these parameters is specified on the ASSORTEDPARMS statement, TCP/IP suppresses the dump generation for program checks. Use of the VMDUMP parameter presumes the availability of the Dump Viewing Facility (DVF) at your installation. Refer to the *CP Command Reference* for additional information on the two dump formats.

     Dumps generated for other error conditions will have a format specified by the error processing routine that intercepted the error (such as the C run-time library). These dumps will be in either the DUMP or VMDUMP format.

     Dumps generated in the VMDUMP format must be processed by the Dump Viewing Facility prior to submission. Refer to the *Dump Viewing Facility Operation Guide* for information on processing VMDMP formatted dumps. When submitting dumps processed by the applicable facility, be sure to include all of the files produced by the processing of the dump (DUMP, REPORT, etc.). Dumps generated in the DUMP format must be read from the system spool to disk (using the RECEIVE command) prior to submission.

Dump files may be transferred to tape using the VMFPLC2 command. Refer to the *Service Guide* for VM for details on using VMFPLC2. Each file dumped to tape should constitute a single tape file (that is, a tape mark should be written after each file is dumped to tape).

- For TCP/IP Traces:

  TCP/IP trace files should be transferred to tape using the VMFPLC2 command. If multiple traces are being submitted, each trace file dumped to tape should constitute a single tape file (that is, a tape mark should be written after each file is dumped to tape).

Note: Use of any other utility (IBM or non-IBM) to transfer dumps or traces to tape may result in a processing delay and could result in the APAR being returned to the customer (closed RET) due to the inability of the change team to process the tape.

2. Submit other types of information (such as server virtual machine traces, configuration files, console logs, etc.) on paper or tape. If submitted on tape, the data should be written to tape using VMFPLC2 only, adhering to the requirement that each file dumped to tape is followed by a tape mark.
3. Write at least ten tape marks after the last file to ensure the load processing correctly recognizes the end of the tape and does not spin off the end off the reel (for 3420 tapes).
4. Tapes that are submitted to the TCP/IP support team must be non-label (nl). Cartridge (3490) or reel tapes may be used. Each tape should contain an external label to identify the tape and its contents in some way. The problem number/apar number should appear on the label. If multiple tapes are used, a separate explanation should be included itemizing the contents of each tape.
5. Generate a map of the tape (or tapes) to be submitted using the VMFPLC2 SCAN command and include the hard copy output of that scan with the tapes.

Necessary Documentation

Before you call for IBM service support, have the following information available:

Information	Description
Customer Number	The authorization code that allows you to use service support. Your account name, and other customer identification should also be available.
Problem Number	The problem number previously assigned to the problem. If this is your first call about the problem, the support center representative assigns a number to the problem.
Operating System	The operating system and level that controls the execution of programs.
Component ID	A number that is used to search the database for information specific to TCP/IP. If you do not give this number to the support center representative, the amount of time taken to find a solution to your problem increases.

Release Number An identification number that is on each TCP/IP release.

Table 2. TCP/IP Component ID Number

Licensed IBM Program Product	Component ID Number
TCP/IP (VM)	5735FAL00

A complex problem might require you to talk to a number of people when you report your problem to service support. Therefore, you should keep all the information that you have gathered readily available.

Note: You might want to keep the items that are constantly required, such as the TCP/IP component ID, or VM operating system release level in a file for easy access.

Additional Documentation

The service support representative might ask you to furnish the following additional items:

- The failing CPU type
- The communication interface, such as X.25 using NPSI or a LAN bridge
- The system fixes and changes.

  Have a list of all program temporary fixes (PTFs) and authorized program analysis report (APAR) fixes that have been applied to your system. You should also have a list of any recent changes made to your system, such as user program modifications, redefinition of statements in system generation, or a change of parameters used to start the system.
- Documentation list

  Prepare a list of all documentation that you use to operate your system and any documentation used to locate or fix the problem.
- System configuration

  System configuration information includes:

  – TCPIP DATA file
  – TCPIP PROFILE file
  – Configuration statements for clients or servers
  – Problem type

    TCP/IP problems are described by one or more of the following categories:

    - Abend
    - Message
    - Loop
    - Wait State
    - Incorrect Output
    - Performance
    - Documentation.

    “Categories that Help Identify the Problem” on page 5 explains how to use these categories when reporting your problem.

Problem Resolution

The service support representative uses the information that you provide to create a list of categories describing your problem.

The program specialist examines all the information that has been compiled, refines your problem definition, and attempts to solve the problem. If a solution is not found in RETAIN or through other sources, the program specialist writes an APAR. A number is assigned to the APAR. The APAR allows the support group to examine your problem more closely and develop a solution. Once the solution is developed and tested, it is entered into RETAIN and sent to you. RETAIN is kept current with new solutions and error descriptions so that future similar problems can be resolved through a problem category search.

Severe Problem Resolution

If your problem is so severe that it must be resolved immediately, you should work closely with a program specialist to help develop a quick solution.

You need to provide the specialist with detailed problem information. Answer questions and follow procedures directed by the program specialist so that a possible quick temporary fix can be developed for your problem.

Customer Worksheet

You, the customer may wish to fill out an informal worksheet to use as a reference before calling for support. By completing this worksheet before calling for support, you will save time and help expedite your fix.

The following Problem Category topic along with the references in Chapter 2, “Problem Identification,” should be reviewed before you call for service support.

Problem Category

Determine within which of the following categories your problem falls:

Category	Description
Abend	An abend occurs when TCP/IP unexpectedly stops processing. These problems are explained in “Abend” on page 6.
Message	The message problem category describes a problem identified by a message. These problems are explained in “Message” on page 6.
Loop	Loop problems refer to an operation that repeats endlessly. These problems are explained in “Loop” on page 7.
Wait State	Wait state problems refer to situations where TCP/IP (or possibly specific servers) fail to respond to requests for service and no activity takes place in the address space or virtual machine of the affected server. These problems are explained in “Wait State” on page 8.

Incorrect Output	An incorrect output problem, such as missing, repeated, or incorrect data, is an unexpected result received during regular network operation. These problems are explained in "Incorrect Output" on page 8.
Performance	A performance problem is characterized by slow response time or slow throughput. These problems are explained in "Performance" on page 9.
Documentation	A documentation problem is defined as incorrect, missing, or ambiguous information in any of the TCP/IP books. These problems are explained in "Documentation" on page 10.

Background Information

After determining the problem category and reviewing the section referring to that category, you must gather the required information regarding your problem. Each problem category detailed in this chapter contains a section called "Gather the Information." See this section to determine the appropriate information you will need to obtain.

Additional Information

Some additional information may be required. See "Additional Documentation" on page 13, to determine if you need more information.

Chapter 3. TCP/IP VM Structures and Internetworking Overview

This chapter describes the TCP/IP implementation for VM. It also provides an overview of networking or internetworking as background information.

VM Structure

Figure 3 represents the TCP/IP layered architecture for the VM environment.

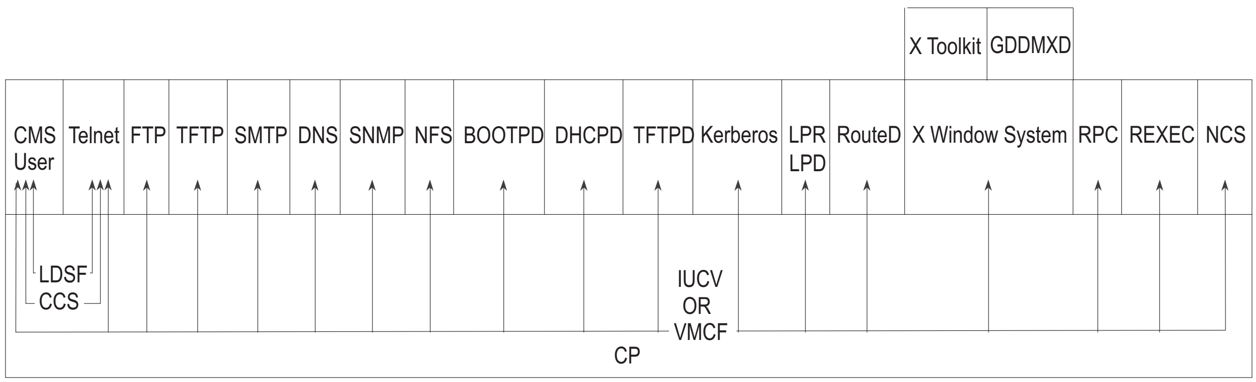


Figure 3. The TCP/IP Layered Architecture for VM

Virtual Machines

In VM, most TCP/IP servers and clients are virtual machines. Each server and client is implemented as an independent virtual machine.

A request for service is sent to the appropriate virtual machine for processing and then forwarded to the appropriate destination. The destination can be the TCP/IP virtual machine if the request is outgoing, or a user's CMS virtual machine if the request is incoming.

The configuration and initialization steps for typical CMS type servers is shown in figure Figure 4.

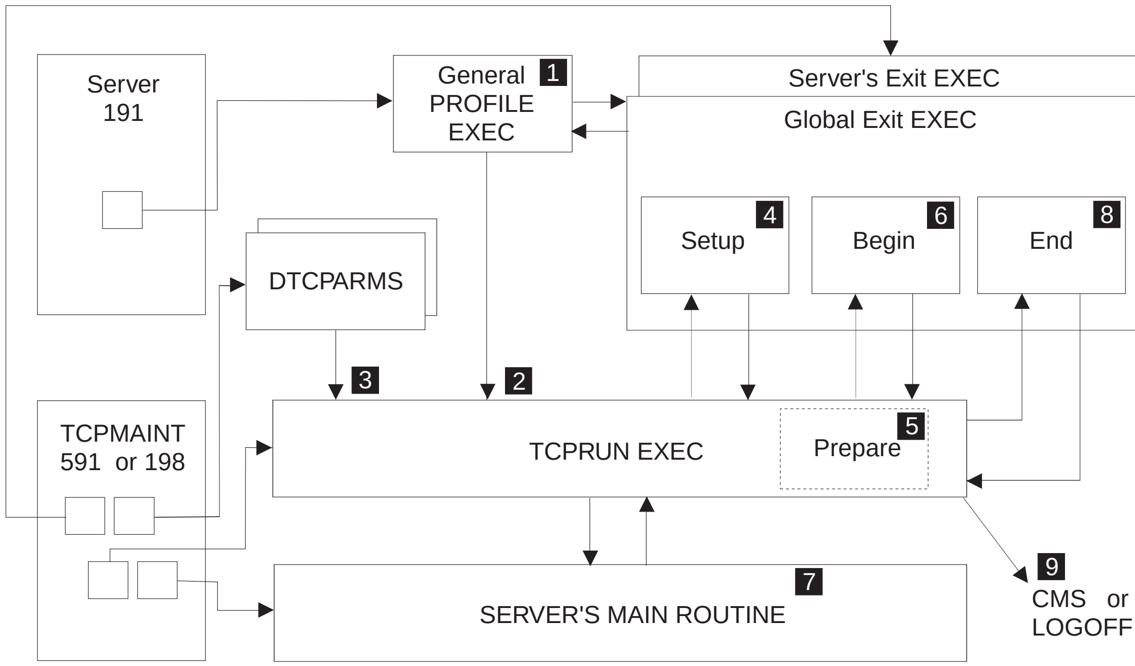


Figure 4. The sequence of a Server Startup

where :

- **1** PROFILE EXEC on 191 accesses required disks
- **2** PROFILE EXEC calls TCPRUN EXEC
- **3** Locate server and server class definitions in DTCPARMS files
- **4** Call any server and global exits with SETUP parameters
- **5** Prepare the execution environment, issuing any needed CP and CMS commands
- **6** Calls any server and global exits with BEGIN parameters
- **7** Run the server
- **8** Call any server and global exits with END parameters
- **9** Return to CMS or logoff

TCPRUN EXEC may also call the exits with the ADMIN or ERROR parameters if the server cannot be started due to administration or problems.

Virtual Machine Communication Facility

The Virtual Machine Communication Facility (VMCF) is used by virtual machines for communication. Because the TCPIP virtual machine has all of the physical interfaces, all communication input/output (I/O) requests are sent to TCPIP for execution.

Inbound data comes into the TCPIP virtual machine and is sent through VMCF to the destination virtual machine. The routing for inbound data is chosen on the basis of the virtual machine that is communicating with the destination.

Inter-User Communication Vehicle

All communication that uses the current socket interface uses the Inter-User Communication Vehicle (IUCV) interface. For example, the Remote Procedure Call (RPC) uses the socket interface and, therefore, RPC communication uses IUCV to communicate with virtual machines.

*CCS and Logical Device Service Facility

*CCS is used for communication between Telnet and a user's CMS virtual machine. This line-mode interface permits requests to be passed between the user and Telnet virtual machines.

When a user requires a full-screen interface, the Logical Device Service Facility (LDSF) is used. This interface simulates a 3270 device on the user's virtual machine, thereby relieving TCP/IP of the need to create a full-screen interface.

Overview of Internetworking

Networking in the TCP/IP world consists of connecting different networks so that they form one logical interconnected network. This large overall network is called an *internetwork*, or more commonly, an *internet*. Each network uses its own physical layer, and the different networks are connected to each other by means of machines that are called *internet gateways* or simply *gateways*.

Note: This definition of a gateway is very different from the one used in general network terms where it is used to describe the function of a machine that links different network architectures. For example, a machine that connects an OSI network to an SNA network would be described as a *gateway*. Throughout this chapter, the TCP/IP definition of a gateway is used.

Figure 5 shows a simple internet with a gateway.

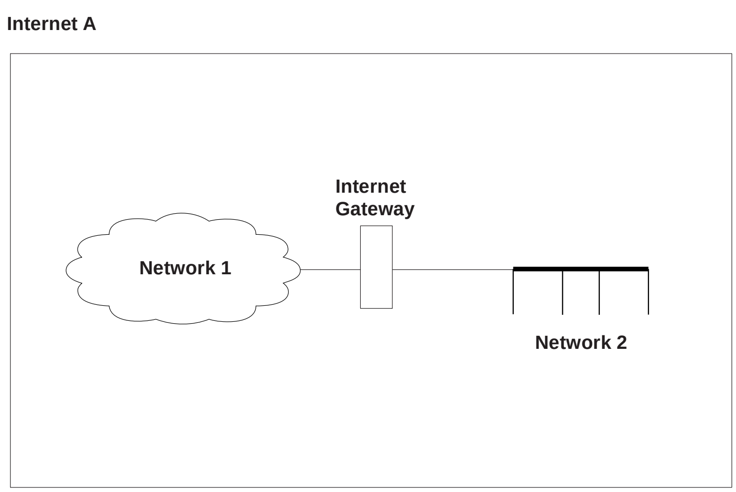


Figure 5. Networks with a Gateway Forming an Internet

The function provided by these gateways is to transfer IP datagrams between the 2 networks. This function is called *routing* and because of this the internet gateways are often called *routers*. Within this chapter, the terms router and gateway are synonymous; both refer to a machine that transfers IP datagrams between different networks.

The linking of the networks in this way takes place at the International Organization for Standardization (ISO) network level. It is possible to link networks at a lower layer level using *bridges*. Bridges link networks at the ISO data link layer. Bridges pass packets or frames between different physical networks regardless of the protocols contained within them. An example of a bridge is the IBM 8209, which can interconnect an Ethernet network and a Token-Ring network.

Note: A bridge does ***not*** connect TCP/IP networks together. It connects physical networks together that will still form the same TCP/IP network. (A bridge does ***not*** do IP routing.)

Figure 6 depicts a router and a bridge. The router connects Network 1 to Network 2 to form an internet.

Internet A

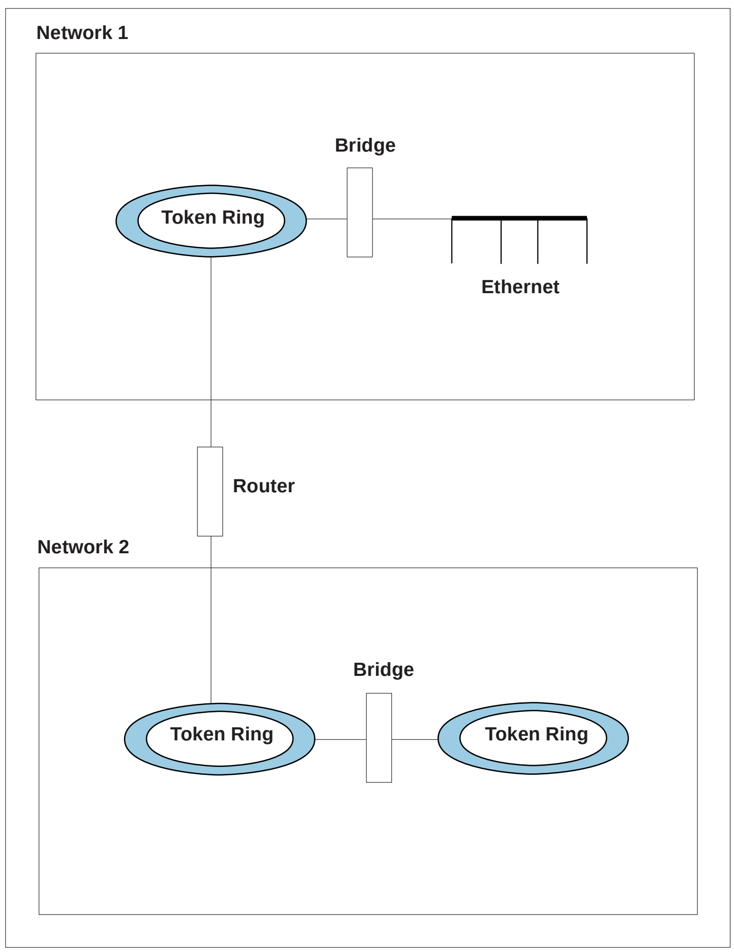


Figure 6. Routers and Bridges within an Internet

Bridges

Bridges are not within the scope of this document; however, there are some aspects of bridging that have a direct effect on TCP/IP networks, particularly in the area of IP routing. This is very important because if IP datagrams are not passed properly over a bridge, none of the higher TCP/IP protocols or applications will work correctly.

Maximum Transmission Unit (MTU)

Different physical networks have different maximum frame sizes. Within the different frames, there is a maximum size for the data field. This value is called the *maximum transmission unit* (MTU), or maximum packet size in TCP/IP terms.

Figure 7 shows the relationship of MTU to frame size.

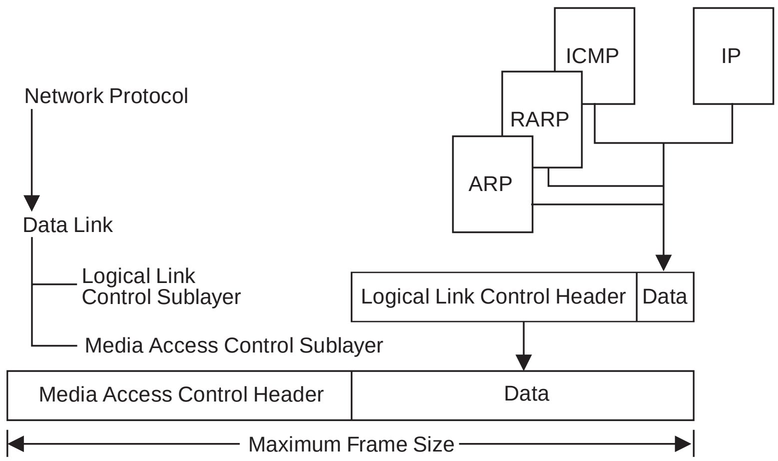


Figure 7. Relationship of MTU to Frame Size

If an IP datagram is to be sent out onto the network and the size of the datagram is bigger than the MTU, IP will fragment the datagram, so that it will fit within the data field of the frame. If the MTU is larger than the network can support, then the data is lost.

The value of MTU is especially important when bridging is used because of the different network limits. *RFC 791 - Internet Protocols* states that all IP hosts must be prepared to accept datagrams of up to 576 bytes. Because of this, it is recommended that an MTU of 576 bytes be used if bridging (or routing) problems are suspected.

Note: MTU is equivalent to the PACKET SIZE value on the GATEWAY statement, or the MAXMTU value when using BSDROUTINGPARMS in the TCPIP PROFILE file.

Token Ring IEEE 802.5

When a token-ring frame passes through a bridge, the bridge adds information to the routing information field (RIF) of the frame (assuming that the bridge supports source route bridging). The RIF contains information concerning the route taken by the frame and, more importantly, the maximum amount of data that the frame can contain within its data field. This is called the maximum information field (I-field). The value specified for the maximum I-field is sometimes referred to as the largest frame size, but this means the largest frame size ***excluding*** headers. See Figure 8 on page 22 for details on the relationship of the I-field to the header fields.

Note: It is important to be aware that IBM's implementation limits the number of bridges through which a frame can be passed to 7. An attempt to pass a frame through an eighth bridge will fail.

The maximum I-field is always decreased by a bridge when it cannot handle the value specified. So, for a given path through several token-ring bridges, the maximum I-field is the largest value that ***all*** of the bridges will support. This value is specified in the Routing Control (RC) field within the RIF as shown in Figure 8 on page 22.

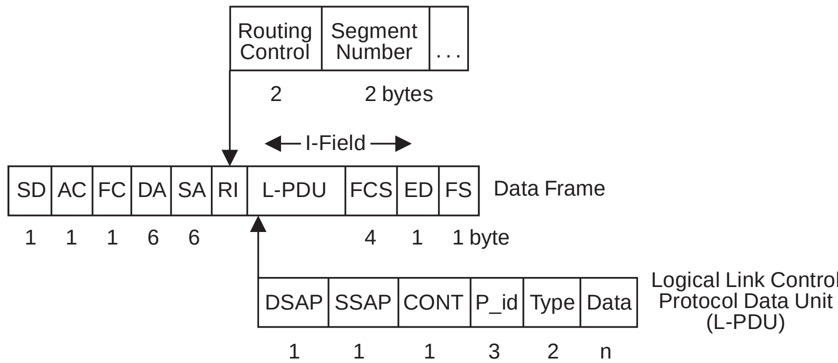


Figure 8. Format of an IEEE 802.5 Token-Ring Frame

The size of the MTU is the maximum amount of data that is allowed within a frame. The token-ring architecture specifies the maximum value of the I-field in the data frame, which corresponds to the maximum size of the L-PDU. The maximum I-field is determined by the bit configuration in the RC field, and is present in all routed frames.

Table 3 shows the relationship between the RC field and the maximum I-field values.

Table 3. Relationship between RC Field and Maximum I-Field Value

Routing Control Field	**Maximum I-Field in Bytes**
x000 xxxx xxxx xxxx	516
x001 xxxx xxxx xxxx	1500
x010 xxxx xxxx xxxx	2052
x011 xxxx xxxx xxxx	4472
x100 xxxx xxxx xxxx	8144
x101 xxxx xxxx xxxx	11407
x110 xxxx xxxx xxxx	17800

In Figure 8, we can see that, within the L-PDU, the Logical Link Control (LLC) header uses 8 bytes, and so the MTU value is 8 bytes less that the maximum I-field. (Note that the L-PDU contains a SNAP header, as described in “Sub-Network Access Protocol (SNAP)” on page 23.) This is how to calculate the MTU for a token ring. The token-ring bridges always adjust the value of the maximum I-field to that of the smallest one in the path. You should always ensure that the MTU value is less than the value specified by the bridge.

Typically, within a 4Mbps token-ring network, the value of maximum I-field will be 2052 bytes, and so the MTU would be set to 2044 bytes (2052 minus 8 bytes for the LLC header).

IEEE 802.3

The frame used in IEEE 802.3 Ethernet networks is shown in Figure 9.

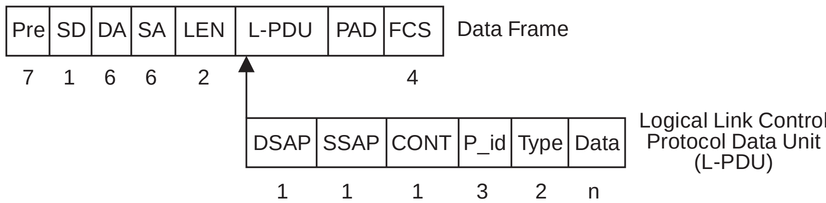


Figure 9. Format of an IEEE 802.3 Frame

The maximum size of the L-PDU for a 10Mbps network is 1500 bytes. Because 8 bytes are used within the L-PDU for the LLC header, this means that the maximum size of the data field is 1492 bytes. Therefore, the MTU for IEEE 802.3 networks should be set to 1492 bytes.

Ethernet - DIX V2

The frame used in DIX Ethernet networks is shown in Figure 10.

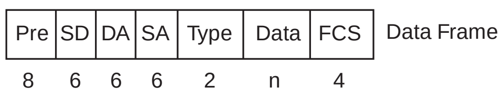


Figure 10. Format of an Ethernet V2 Frame

There is no LLC data in an Ethernet V2 frame. The maximum size for the frame is 1526 bytes. This means that the data field can be 1500 bytes maximum. The MTU for Ethernet V2 can be set to 1500 bytes.

It is possible to bridge Ethernet V2 frames to either IEEE 802.3 or IEEE 802.5 networks; a LLC header is added or removed from the frame, as required, as part of the conversion when bridging.

Sub-Network Access Protocol (SNAP)

The TCP/IP software provides protocol support down to the ISO network layer. Below this layer is the data link layer, which can be separated into two sublayers. These are the *Logical Link Control* (LLC) and the *Media Access Control* (MAC) layers.

The IEEE 802.2 standard defines the LLC sublayer, and the MAC sublayer is defined in IEEE 802.3, IEEE 802.4, and IEEE 802.5.

The format of an IEEE 802.2 LLC header with the SNAP header is shown in Figure 11.

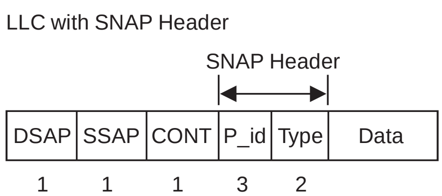


Figure 11. SNAP Header

The values of the fields in the LLC header when a SNAP header is used are specified in *RFC 1042 - Standard for Transmission of IP Datagrams over IEEE 802 Networks*. The values specified are:

Field	Value
DSAP	X'AA'
SSAP	X'AA'
CONT	X'03' Specifies unnumbered information (UI)
P_id	X'00 00 00'
Type	X'08 00' - IP X'08 06' - ARP X'08 35' - RARP

IP Routing

IP routing is based on routing tables held within a router or internet host. These tables can either be *static* or *dynamic*. Typically, static routes are predefined within a configuration file, and dynamic routes are “learned” from the network, using a *routing* protocol.

Internet Addressing

Hosts on an internet are identified by their *IP address*. *Internet Protocol* (IP) is the protocol that is used to deliver datagrams between these hosts. It is assumed the reader is familiar with the TCP/IP protocols. Specific information relating to the Internet Protocol can be found in RFC 791.

An IP address is a 32-bit address that is usually represented in dotted decimal notation, with a decimal value representing each of the 4 octets (bytes) that make up the address. For example:

```
00001001010000110110000100000010        32-bit address
00001001 01000011 01100001 00000010     4 octets
   9        67       97       2         dotted decimal notation (9.67.97.2)
```

The IP address consists of a *network address* and a *host address*. Within the internet, the network addresses are assigned by a central authority, the *Network Information Center* (NIC). The portion of the IP address that is used for each of these addresses is determined by the *class of address*. There are three commonly used classes of IP address (see Figure 12). (A fourth class, Class D, which is used for multicast, will not be discussed).

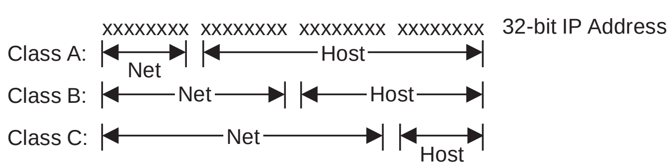


Figure 12. Classes of IP Addresses

The class of address of the IP network is determined from the first 4 bits in the first octet of the IP address. Figure 13 on page 25 shows how the class of address is determined.

```
32-bit address          xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

Class A                 0xxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
              min       00000000
              max       01111111
              range     0 - 127    (decimal notation)

Class B                 10xxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
              min       10000000
              max       10111111
              range     128 - 191  (decimal notation)

Class C                 110xxxxx xxxxxxxx xxxxxxxx xxxxxxxx
              min       11000000
              max       11011111
              range     192 - 223  (decimal notation)
```

Figure 13. Determining the Class of an IP Address

As shown in Figure 13, the value of the bits in the first octet determine the class of address, and the class of address determines the range of values for the network and host segment of the IP address. For example, the IP address 9.67.97.2 would be a class A address, since the first 2 bits in the first octet contain B'00'. The network part of the IP address is “9” and the host part of the IP address is “67.97.2.”

Refer to *RFC 1166 - Internet Numbers* for more information about IP addresses. Refer to *RFC 1060 - Assigned Numbers* for more information about reserved network and host IP addresses, such as a *network broadcast address*.

Figure 14 on page 26 shows a simple network with a bridge and a router.

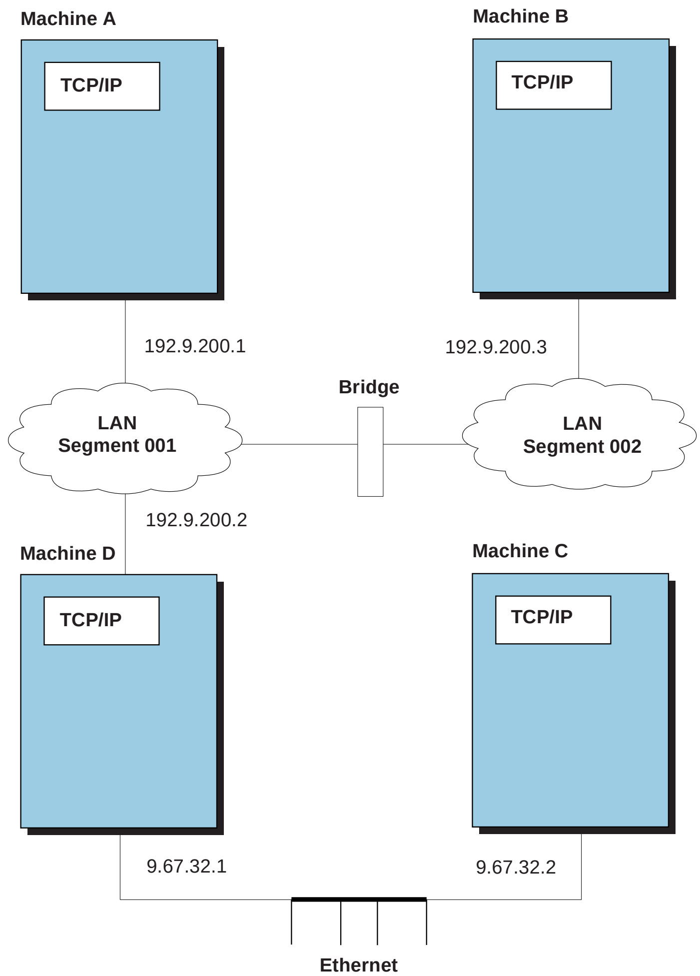


Figure 14. Routing and Bridging

Machine D is acting as an IP router and will transfer IP datagrams between the class C, 192.9.200, network and the class A, 9, network. It is important to note that for Machine B to communicate with Machine C using TCP/IP, both Machine D and the bridge have to be correctly configured and working.

Direct Routing

Direct routing can take place when two hosts are directly connected to the same physical network. This can be a bridged token-ring network, a bridged Ethernet, or a bridged token-ring network and Ethernet. The distinction between direct routing and indirect routing is that with direct routing an IP datagram can be delivered to the remote host without subsequent interpretation of the IP address, by an intermediate host or router.

In Figure 14, a datagram travelling from Machine A to Machine B would be using direct routing, although it would be traveling through a bridge.

Indirect Routing

Indirect routing takes place when the destination is ***not*** on a directly attached IP network, forcing the sender to forward the datagram to a router for delivery.

In Figure 14 on page 26, a datagram from Machine A being delivered to Machine C would be using indirect routing, with Machine D acting as the router (or gateway).

Simplified IP Datagram Routing Algorithm

To route an IP datagram on the network, the algorithm shown in Figure 15 is used.

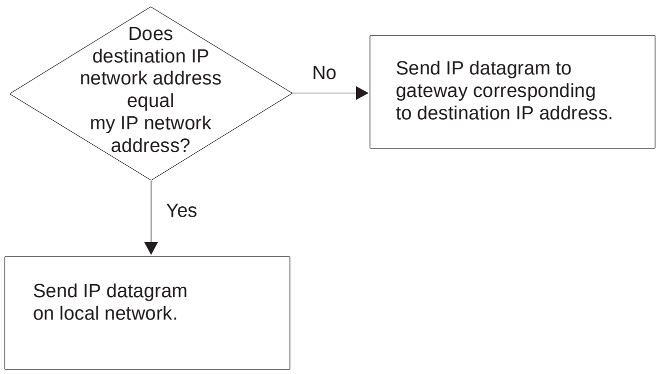


Figure 15. General IP Routing Algorithm

Using this general routing algorithm, it is very easy to determine where an IP datagram will be routed. Following is a simple example based on the configuration shown in Figure 14 on page 26.

```
Machine A   IP Address = 192.9.200.1

Routing Table

Destination     Gateway

192.9.200.1     192.9.200.1   (Machine A's network interface)

9.0.0.0         192.9.200.2   (Route to the 9.n.n.n address is
                               via Machine D, 192.9.200.2)
```

Machine A sends a datagram to host 192.6.200.3 (Machine B), using the direct route, 192.9.200.1 (its own network interface). Machine A sends a datagram to host 9.67.32.2 (Machine C), using the indirect route, 192.9.200.2 (Machine D), and Machine D then forwards the datagram to Machine C.

Subnetting

A variation of the network and host segments of an IP address, known as *subnetting*, can be used to physically and logically design a network. For example, an organization can have a single internet network address (NETID) that is known to users outside the organization, yet configure its internal network into different departmental subnets. Subnetwork addresses enhance local routing capabilities, while reducing the number of network addresses required.

To illustrate this, let us consider a simple example. Assume that we have an assigned class C network address of 192.9.200 for our site. This would mean that we could have host addresses from 192.9.200.1 to 192.9.200.254. If we did not use subnetting, then we could only implement a single IP network with 254 hosts. To split our site into two logical subnetworks, we could implement the following network scheme:

```
Without Subnetting:
                                          Network        Host Address
                                          Address        Range
       192      9        200       host
    11000000 00001001 11001000 xxxxxxxx   192.9.200      1 - 254


With Subnetting:
                                          Subnet         Host Address  Subnet
                                          Address        Range         Value
       192      9        200   64  host
    11000000 00001001 11001000 01xxxxxx   192.9.200.64   65 - 126      01


                                          Subnet         Host Address  Subnet
                                          Address        Range         Value
       192      9        200   128 host
    11000000 00001001 11001000 10xxxxxx   192.9.200.128  129 - 190     10

The subnet mask would be

       255      255      255     192
    11111111 11111111 11111111 11000000
```

Notice that there are only two subnets available, because subnets B'00' and B'11' are both reserved. All 0's and all 1's have a special significance in internet addressing and should be used with care. Also notice that the total number of host addresses that we can use is reduced for the same reason. For instance, we cannot have a host address of 16 because this would mean that the subnet/host segment of the address would be B'0001000', which with the subnet mask we are using, would mean a subnet value of B'00', which is reserved.

The same is true for the host segment of the fourth octet. A fourth octet value of B'01111111' is reserved because, although the subnet of B'01' is valid, the host value of B'1' is reserved.

Each bit of the network segment of the subnet mask is always assumed to be 1, so each octet has a decimal value of 255. For example, with a class B address, the first 2 octets are assumed to be 255.255.

Simplified IP Datagram Routing Algorithm with Subnets

The algorithm to find a route for an IP datagram, when subnetting is used, is similar to the one for general routing with the exception that the addresses being compared are the result of a logical AND of the subnet mask and the IP address.

For example:

```
IP address:       9.67.32.18   00001001 01000011 00100000 00010010
                                                  <AND>
Subnet Mask: 255.255.255.240   11111111 11111111 11111111 11110000

Result of
Logical AND:      9.67.32.16   00001001 01000011 00100000 00010000
```

The subnet address is 9.67.32.16, and it is this value that is used to determine the route used.

Figure 16 shows the routing algorithm used with subnets and Figure 17 shows how a subnet route is resolved.

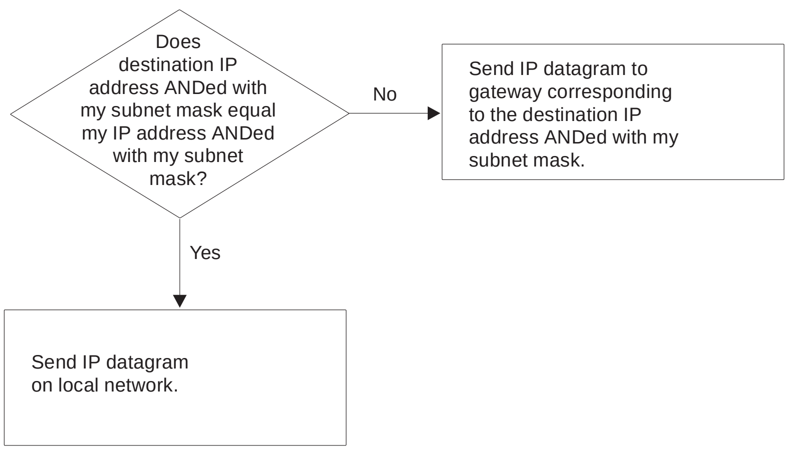


Figure 16. Routing Algorithm with Subnets

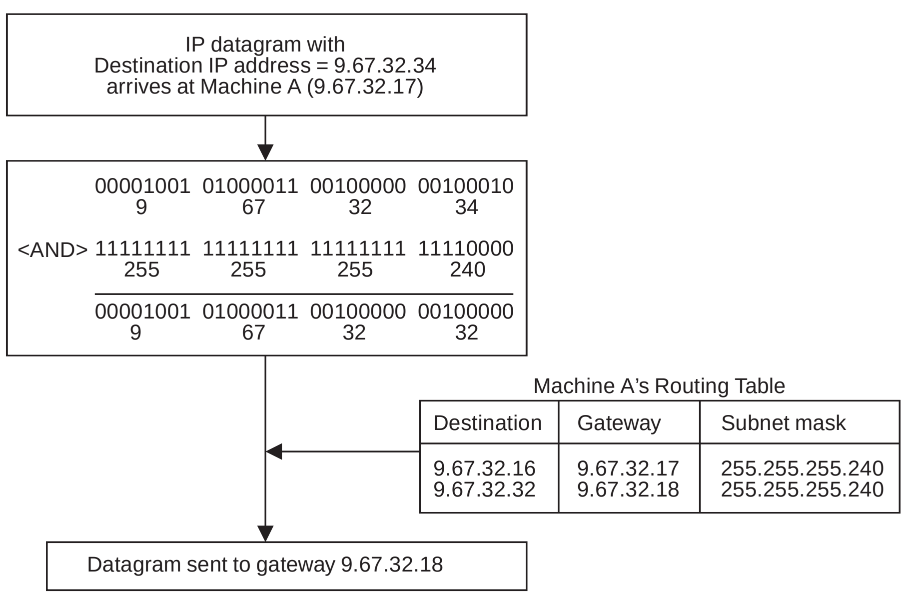


Figure 17. Example of Resolving a Subnet Route

Static Routing

Static routing, as the name implies, is defined within the local host, and as changes to the network occur, must be manually changed. Typically, a configuration file will contain the definitions for directly-attached networks, routes for specific hosts, and a possible default route that directs packets to a destination for networks that are not previously defined.

TCP/IP uses the GATEWAY statements, defined in the TCPIP PROFILE file, to configure the internal routing tables. The internal routing tables for TCP/IP can be modified by using the OBEYFILE command. Refer to the *TCP/IP Planning and Customization* for details about defining the GATEWAY statements and using the OBEYFILE command.

Note: When the GATEWAY statements are updated using OBEYFILE, all previously-defined routes are discarded and replaced by the new GATEWAY definitions.

Dynamic Routing

Dynamic routing is the inverse of static routing. A TCP/IP protocol is used to dynamically update the internal routing tables when changes to the network occur. TCP/IP uses the Routing Information Protocol (RIP) and the RouteD virtual machine to monitor network changes. The *TCP/IP Planning and Customization* contains more details about RouteD.

Note: When you use RouteD, the GATEWAY statements ***must*** be commented out of the TCPIP PROFILE file, and the BSDROUTINGPARMS statements should be used to configure the initial network definitions.

Dynamic Routing Tables

When TCP/IP is configured to use RouteD, there are actually two routing tables. The first routing table is managed by RouteD, and is updated dynamically based on the RIP protocol. RouteD will then update the internal routing table of the RouteD virtual machine. The two routing tables ***might not be identical*** for the following reasons:

- ICMP redirects are received by the TCPIP address space. TCPIP updates its internal routing table, but these changes are not propagated to RouteD. To prevent this situation, the parameter IGNOREREDIRECTS, should be coded in the TCPIP PROFILE file.
- The GATEWAY statements are not commented out in the TCPIP PROFILE file. In this situation, TCPIP will route packets based on the GATEWAY statements, and then based on the updates by RouteD. This is similar to a condition in UNIX** environments known as “kernel” routes.

Customizing both the GATEWAY and BSDROUTINGPARMS statements should only be attempted by network programmers familiar with IP routing, RIP, and the ramifications of having distinct routing tables.

Example of Network Connectivity

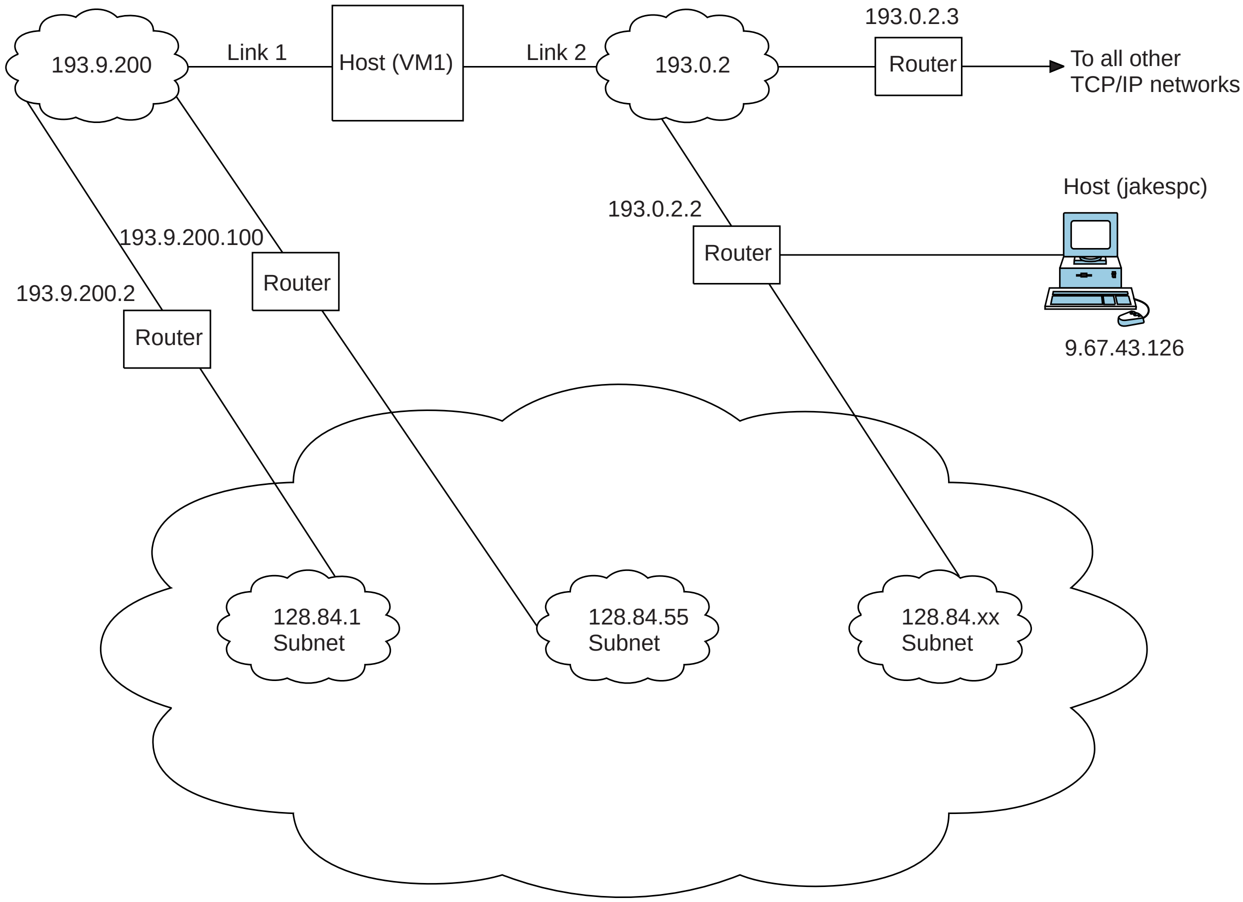


Figure 18. Example of Network Connectivity

Figure 18 shows a host, `VM1`, directly connected to networks 193.9.200 and 193.0.2. Neither network has subnets. `VM1` is indirectly connected to network 128.84, which has subnets using the high-order byte of the host number as the subnet field. The subnet 128.84.1 is accessible through 193.9.200.2; the subnet 128.84.55 is accessible through 193.9.200.100; and the other subnets of 128.84 are accessible through 193.0.2.2. All packets destined for a network that has no entry in the routing table should be routed to 193.0.2.3. All packets to the host `jakespc` should be routed through 193.0.2.2.

Notes:

1. Directly-attached networks must be defined in the GATEWAY table before default networks (DEFAULTNET) or first-hop networks (FIRSTHOP) are defined.
2. Verification of the TCPIP virtual machine is recommended for connectivity issues, regardless of whether overt internal or external changes have been made to the system.

Chapter 4. Server Initialization

This chapter describes the mechanism used to start each TCP/IP server.

CMS Servers

Servers that run under CMS share a common profile, TCPROFIL EXEC. It is copied by TCP2PROD to each server's 191 disk as PROFILE EXEC. You should never modify this file as it may be replaced by TCP/IP service procedures.

The profile accesses the common disks (198, 591, and 592) and then calls TCPRUN EXEC. TCPRUN determines what kind of server is running and invokes the appropriate server function. The kind of server is referred to as the *server class*. It is obtained from the *userid*, *nodeid*, SYSTEM, or IBM DTCPARMS file.

The DTCPARMS file contains all of the information needed to establish the necessary runtime environment and to start the server. Exits can be defined to override any value set by a DTCPARMS file. A complete description of the DTCPARMS file and the server initialization process can be found in the *TCP/IP Planning and Customization*.

Because the various tags in the DTCPARMS file are used to determine what special environments should be created, as well as the options or parameters that will be passed to the server, it may become necessary to determine the precise commands that are issued.

A trace of TCPRUN EXEC can be obtained using one of the following procedures.

Diagnosis Method 1

1. Logon to the server and indicate that you do not want the server to start.
2. Enter the command `TCPRUN (DEBUG`.
3. Stop the server (#CP EXT or HX)
4. Examine the trace file, `TCPRUN DEBUG A`.

Diagnosis Method 2

If a problem only occurs when the server is disconnected, an alternate trace method is provided.

1. Logon to the server and indicate that you do not want the server to start.
2. Enter the command `GLOBALV SELECT DTCRUN SETLP DEBUG 1`.
3. Logoff.
4. Autolog the server.
5. Logon to the server and stop it (#CP EXT or HX)
6. Examine the trace file, `TCPRUN DEBUG A`.
7. Enter the command `GLOBALV SELECT DTCRUN SETLP DEBUG` (set DEBUG to null) to turn off the trace.

GCS Servers

Servers that run under the GCS operating system share a common profile, TCPROFIL GCS. It is copied by TCP2PROD to each server's 191 disk as PROFILE GCS. You should never modify this file as it may be replaced by TCP/IP service procedures.

The profile will then search for and run *userid* GCS. The DTCPARMS file is not used by the GCS servers.

Due to the simple nature of the relationship between the common profile and the server-specific GCS exec, no debug facility is provided.

Chapter 5. TCP/IP Procedures

This chapter describes some of the internal procedures that occur in the TCP/IP server and the types of input/output (I/O) supported by TCP/IP.

You should collect the messages, console logs, and system and user dumps pertaining to TCP/IP server protocols and procedures. You should also trace TCP/IP protocols or procedures to determine TCP/IP suite problems, such as TCP requests from remote and local clients or servers.

TCP/IP Internals

The following sections describe the internal procedures, queues, and activities for TCP/IP.

Internal Procedures

Table 4 describes the major internal Pascal procedures. These procedures are external declarations of processes invoked by the scheduler.

Table 4 (Page 1 of 3). TCP/IP Internal Procedures

Procedure	Description
ArpProcess	Processes Address Resolution Protocol requests.
CallProcRtn	Calls the appropriate processing routine for Activity Control Blocks (ACBs) with a ProcessName of DEVICEdriverNAME.
ClientTimer	Converts an INTERNALclientTIMER ACB to a notification to the internal client.
ConsistencyChecker	Schedules itself at regular intervals to perform various tests of the TCP/IP machine's internal consistency. The ConsistencyChecker maintains various statistics about recent resource usage. It tries to restart well-known clients that appear to be inactive and attempts to collect infrequently used, but active, TCBs.
From1822	Receives incoming datagrams and IMP messages from the Series/1 on the Defense Data Network (DDN). Processes incoming IMP messages and passes the incoming datagrams to IpUp.
IntCliProc	Processes notifications for the internal client.
IpDown	Processes outgoing IP datagrams received from TcpDown. It selects the network to use for the first hop, and the address within that network to employ. It passes datagrams to ToGlue to send to the Series/1. IpDown also processes table-driven gateway selections for IpDown's routings (except for the internal loopback routes used for debugging, which are hard coded into DispatchDatagram). All routines are placed in IpDown and other processes, such as IpUp (for ICMP redirect messages) and TcpIpInitialize. You can access the routing information using these routines.

Table 4 (Page 2 of 3). TCP/IP Internal Procedures

Procedure	Description
IpUp	Processes incoming IP datagrams. If necessary, it reassembles fragmented datagrams. IpUp sends completed datagrams to TcpUp, UdpUp, or RawIpUp.
IucvApiGreeter	Processes new IUCV paths from clients using IUCV APIs.
Monitor	Maintains internal performance records. It receives status requests from clients and information on the Series/1 through StatusIn. The Monitor collects run-time performance statistics and responds to requests from clients to execute commands that alter internal routing and addressing information, write out performance records, control run-time debug tracing, and indicate the clients that are authorized to make these special requests. The Monitor also handles some unusual situations, such as recording errors detected by the interrupt handlers (which cannot simply write out tracing, because they function with interrupts disabled) and attempting to autolog well-known clients.
Notify	Sends asynchronous notifications to clients. It processes ACB, CCB, and TCB bufferpools to manage the notifications sent to clients through VMCF.
PingProcess	Processes PING requests, responses, and time-outs.
RawIpRequest	Processes incoming RAWIP requests. It passes outgoing datagrams to IpDown.
Scheduler	The scheduler checks the queues of executable activities, removes the first item of the highest priority, nonempty queue, and invokes the indicated process. If all of the executable job queues are empty, it is inactive until an interrupt arrives and schedules some activity. If the consistency checker is not currently scheduled to execute and there is activity scheduled on the main job queue (the ToDoQueue), the scheduler establishes a time-out, so that the consistency checker can be invoked.
ShutDown	Shuts down the TCPIP server gracefully. The DoShutDown parameter returns a true value, and then a return from the scheduler to main program shutdown is used to call the halt procedure. You need to return to main to print profile statistics.
SnmpDpiProcess	Processes SNMP DPI requests from an SNMP agent.
SockRequest	Processes BSD-style socket requests.
StatusOut	Receives requests for information on the status of Glue from the Monitor, which it passes to ToGlue on the Series/1.
TcpDown	Creates outgoing TCP segments based on the client requests handled by TcpRequest and the remote socket responses handled by TcpUp. TcpDown packages these segments into IP datagrams, which it passes to IpDown.
TcpRequest	Processes client's requests for TCP service and for handling asynchronous notifications. Buffers outgoing client TCP data, updates the state of TCP connections, and signals TcpDown to send TCP segments.

Table 4 (Page 3 of 3). TCP/IP Internal Procedures

Procedure	Description
TcpUp	Processes incoming TCP segments received from IpUp. If necessary, TcpUp signals Notify to generate asynchronous notifications about TCP connections. It also processes window and acknowledgment information from the remote socket.
Timer	Checks the TimerQueue for any time-outs that may be due and places them in the ToDoQueue. Then Timer resets the external timer to awaken it later if future time-outs are pending. Timer also encapsulates all operations involving time-outs, including the Timer process that transforms time-outs into active signals. The TimerQueue is referenced in the TCQueue segment.
ToA220	Sends the outgoing datagrams supplied by IpDown to A220. See "HYPERchannel Driver" on page 43 for more information.
ToCeti	Sends the outgoing datagrams supplied by IpDown to CETI. See "CETI Driver" on page 42 for more information.
ToGlue	Sends outgoing datagrams supplied by IpDown to the Series/1.
ToIUCV	Sends the outgoing datagrams supplied by IpDown to PVM IUCV. See "IUCV Links" on page 45 for more information.
ToPCCA3	Sends the outgoing datagrams supplied by IpDown to PCCA3. PCCA is the name for LAN channel-attached units.
To1822	Sends outgoing datagrams supplied by IpDown to the Series/1 on DDN.
UdpRequest	Processes incoming UDP requests. Gives outgoing datagrams to IpDown.
1822Status	Receives status information from the 1822 interrupt handlers about the IMP and the Series/1, and passes that information to the clients.
1822Timer	Controls OutHost table cleanup, and brings down and reinitializes IMP.

Queues

Table 5 describes the queues TCP/IP uses to control events that occur during run-time.

Table 5 (Page 1 of 2). TCPIP Queues

Queue	Description
InDatagram	The various device drivers place incoming IP datagrams in this queue for IpUp to process.

Table 5 (Page 2 of 2). TCPIP Queues

Queue	Description
QueueOfCcbsForTcpResources	This queue contains a list of ACBs pointing to CCBs that have tried to perform TcpOpen, but failed because of a lack of TCBs, data buffers, or SCBs. As resources become available, they are assigned to the first CCB on this list. When all resources (a TCB and two data buffers) are available, a RESOURCESavailable notice is sent to the client, who reissues the open.
QueueOfCcbsForUdpResources	This queue contains a list of ACBs pointing to CCBs that have tried to perform UdpOpen, but failed because of a lack of UCBs or SCBs. Processing is similar to QueueOfCcbsForTcpResources.
QueueOfRcbFrustrated	This queue contains raw-IP client-level requests to send datagrams that cannot be processed, because of a shortage of buffer space. When buffer space becomes available internally, the RAWIPspaceAVAILABLE notice is sent to the appropriate clients, and the requests are removed from this queue.
QueueOfTcbFrustrated	This queue contains client-level TCP send-requests that cannot be satisfied, because of a shortage of internal TCP buffer space. When buffer space becomes available internally, the BUFFERspaceAVAILABLE notice is sent to the appropriate clients, and the requests are removed from this queue.
QueueOfUcbFrustrated	This queue contains UDP client-level requests to send datagrams that cannot be satisfied, because of a shortage of buffer space. When buffer space becomes available internally, the UDPdatagramSPACEavailable notice is sent to the appropriate clients, and the requests are removed from this queue.
Segment: EnvelopePointerType	IpUp places incoming TCP segments in this queue for TcpUp to process.
ToDoPullQueue, ToDoPushQueue	This is the primary queue for executable activities. Activities are placed in this queue directly by Signal and indirectly by SetTimeout. The scheduler removes these activities from the queue and invokes the corresponding process.

Internal Activities

Table 6 describes TCP/IP internal activities performed by TCP/IP processes. An example of called internal activities is shown in Figure 44 on page 79. Activities, which are found in most TCP/IP internal traces, explain why the process has been called.

Table 6 (Page 1 of 4). TCP/IP Internal Activities

Activity	Description
ACCEPTipREQUEST	Sent by the external interrupt handler to the IPrequestor informing it of an incoming IP-level request from a local client.
ACCEPTmonitorREQUEST	Sent by the external interrupt handler to the Monitor informing it of an incoming monitor request from a client.
ACCEPTpingREQUEST	Sent by the external interrupt handler to the PING process informing it of an incoming PING request from a client.
ACCEPTrawipREQUEST	Sent by the external interrupt handler to the RAWIPrequestor informing it of an incoming RAWIP-level request.
ACCEPTtcpREQUEST	Sent by the external interrupt handler to the TCPrequestor informing it of an incoming TCP-level request (or a request that belongs to both IP and TCP, such as Handle) from a local client.
ACCEPTudpREQUEST	Sent by the external interrupt handler to the UDPrequestor to inform it of an incoming UDP-level request.
ACKtimeoutFAILS	Sent by the Timer to TcpDown when an ACK time-out fails.
ARPtimeoutEXPIRES	Sent by the Timer to the ARP process when it is time to scan the queue for packets that are waiting for an ARP reply. Outdated packets are discarded.
CCBwantsTCB	This is not an activity. ACBs with this activity value point to CCBs that attempted to perform TcpOpen, but failed because of a lack of TCBs or data buffers. These ACBs are located in QueueOfCcbsForTcpResources.
CCBwantsUCB	This is not an activity. ACBs with this activity value point to CCBs that attempted to perform UpdOpen, but failed because of a lack of UCBs or data buffers. These ACBs are located in QueueOfCcbsForUpdResources.
CHECKconsistency	Sent by any process to check the ConsistencyChecker for the internal data structures.
DELETEtcb	Sent by the Timer to the TCPrequestor, signifying that enough time has elapsed since the connection was closed to free the TCB without endangering later sequence numbers or allowing internal dangling pointers.
DEVICEspecificACTIVITY	Sent by a device driver to itself for a driver-specific purpose.
DISPOSEsockTCB	Sent by various processes to SockRequest to delete a TCB owned by a BSD socket-style client.
EXAMINEincomingDATAGRAM	Sent by IP-down or a network driver (such as From-r) to IpUp when it places an incoming datagram in the global InDatagram Queue.

Table 6 (Page 2 of 4). TCP/IP Internal Activities

Activity	**Description**
EXAMINEincomingSEGMENT	Sent by IpUp to TcpUp when an incoming datagram contains a TCP segment. It places these datagrams in the global InSegment Queue.
FINISHdatagram	Sent by IP-request, TcpDown, and IpUp signifying the presence of outgoing datagrams in the global OutDatagram Queue. These datagrams are available for IpDown, which completes the IP header and sends it out.
FROMlineSENSE	Sent by the 1822 interrupt handler when a unit check ending status is given by the channel to an I/O command.
HAVEcompletedIO	Sent by the I/O interrupt to a network driver when the most recent I/O operation has been completed.
HOSTtimeout	Sent by the 1822 initialization routine to the 1822-Timer routine to check the OutHost table for entries whose idle time limit has been exceeded.
IMPdown	Sent by the From-1822 routine to the 1822-Timer routine when there is an indication that the IMP is about to go down.
IMPinit	Sent by the From-1822 routine to the 1822-Timer routine when there is an indication from the Series/1 that the IMP is down and needs to be reinitialized.
INFORMmonitor	Sent by any internal process informing the Monitor of some noteworthy situation or event.
INTERNALclientTMOUT	Sent to the INTERNALclientTIMERname process, which converts it to an internal client notification.
INTERNALldsfNOTIFICATION	Sent to the internal client, which passes a notification of an LDSF interrupt.
INTERNALnotification	Sent to the internal client, which passes a notification.
IUCVrupt	Sent to the ToIUCV process when an IUCV interrupt occurs.
KILLdetachedTCB	Sent by the Timer to TcpRequest indicating that a TCB, which was detached from a BSD-style socket, has not disappeared. TcpRequest then deletes it.
LOOKatTIMERqueue	Sent by the external interrupt handler to awaken the Timer process. The Timer process then removes the appropriate items from the TimerQueue and places them in the ToDoQueue.
NOactivity	Sent when someone does not initialize an ACB.
OPENtimeoutFAILS	Sent by the Timer to TcpRequest when an open time-out fails.
PENDINGpingREQUEST	This is not an activity. ACBs with this activity value contain information on PING requests awaiting a response or time-out. These ACBs are in a local queue within TCPING.
PINGtimeoutFAILS	Sent by the Timer to the PING process when a Ping request times out.

Table 6 (Page 3 of 4). TCP/IP Internal Activities

Activity	Description
PROBEtimeoutFAILS	Sent by the Timer to TcpDown when a window probe should be sent to a given connection.
PROCESSsnmpAGENTrequest	Sent by Sock-request to the SNMP DPI process when a write() is performed on a special SNMPDPI socket.
QUITwaiting	Sent by the Timer to TcpRequest when a connection in a time-wait state should be closed.
READdatagram	Sent by the I/O interrupt handler to FromGlue or StatusIn indicating that a message from the Series/1 should be read.
REASSEMBLYfails	Sent by the Timer to IpUp when a datagram reassembly times out.
REJECTunimplementedREQUEST	Sent by the external interrupt handler to Monitor instructing it to reject an unrecognized request.
RESETconnection	Sent by TcpRequest to TcpDown in response to a client's abort or by TcpUp in response to an unacceptable segment. It instructs TcpDown to send an RST to the foreign socket, appearing as though it came from the local socket with the necessary values for RCV.NXT and SND.NXT.
RETRANSMITdata	Sent by the Timer to TcpDown when TCP data should be retransmitted.
RETRYread	Some drivers set a time-out for this activity if they are unable to start a read channel program. When the time-out expires, the drivers try the read again.
RETRYwrite	Some drivers set a time-out for this activity if they are unable to start a write channel program. When the time-out expires, the drivers try the write again.
SELECTtimeoutFAILS	Sent by the Timer to Sock-request indicating that a select() time-out has expired.
SENDdatagram	Sent by IpDown to the network driver of its choice indicating the availability of one or more datagrams to send on that network. At present, the supported drivers are ToGlue, ToPronet, and ToEthernet, and the supported networks are Telenet, Pronet, and Ethernet, respectively.
SENDnotice	Sent to Notify by any process that has discovered information that warrants sending an asynchronous notification to a client.
SendOFFControl	Sent by any internal process informing the Series/1 that the host is not ready.
SendONControl	Sent by any internal process informing the Series/1 that the host is up and ready.
SENDreadDIAG	Sent by any internal process conducting a test of the Series/1 read channel.
SENDtcpDATA	Sent by TcpRequest to TcpDown when data is available to send to the specified connection.

Table 6 (Page 4 of 4). TCP/IP Internal Activities

Activity	Description
SENDwriteDIAG	Sent by any internal process before it tests the Series/1 write channel.
SEND1822noops	Sent by any internal process before it sends three 1822 NOOP messages to the IMP.
SHUTdownTCPipSERVICE	Sent to the shutdown process instructing it to terminate the TCPIP server gracefully.
STOPlingering	Sent by the Timer to TcpRequest indicating that the lingering time-out for a socket-style connection has expired. TcpRequest then releases the client.
TERMINATEnotice	Sent by the external interrupt handler to Notify when a final response has been received for an outstanding VMCF message that Notify has sent.
TOlineSENSE	Sent by the 1822 interrupt handler when a unit check ending status is given by the channel to an I/O command.
TRYautologging	Sent to the monitor by the timer when an autologged client has been forced off the network. An attempt is then made to log on the client.
TRYiucvCONNECT	Sent by the IUCV driver to itself (meout)ll, so that it can retry an IUCV connect that has failed.

Input/Output

The following sections describe the types of I/O supported by TCP/IP. These I/O types include CETI, HYPERchannel, and IUCV.

CETI Driver

The CETI driver controls the 9370 internal adapters supported by TCP/IP for VM implementations (token-ring, Ethernet, and X.25 adapters).

The TCCETI1_ceti1 segment is the primary CETI driver. This segment prepares channel control words (CCWs), controls memory requirements for CETI I/Os, and processes initializations and communication with the 9370 internal adapters. Table 7 describes some of this segment's procedures.

Table 7. TCPIP Internal Activities

Procedure	Description
SendCetiMessage	This is the output routine for a CETI data port. It sends data passed to it over the addressed CETI port to a given destination. The message data and an appropriate header are placed in the correct locations in the data port buffer area. Transfer of the message to the CETI controller is signaled by updating the outbound control block. This process has four steps: 1. Checks for sufficient space; a return code is issued if there is insufficient space. 2. Moves the message and appropriate header to the proper location in the buffer. 3. Updates the outbound control block, which notifies the controller of the outbound message. 4. Updates host-specific information, CurrentBuff, and statistics.
GetCetiMessage	This is the primary interface for receiving messages sent to an inbound CETI port. It retrieves the next message from the port indicated by the portindex parameter and places it in the specified memory. After the message has been transferred, the control block is updated to indicate that the buffer has been freed. Data length is set by the caller to the maximum size message that it can accept and reset by GetCetiMessage to the actual number of bytes transferred. This procedure updates INDX after the message has been transferred, so that INDX remains NumBuffs ahead of the next inbound message. It also updates SLPT, so that SLPT refers to the same buffer as CurrentBuff. **Note:** This procedure assumes that CheckCetiIo has verified the presence of a message.
ToCeti	This is the primary external task entry point for activating the CETI driver. During device initialization, this procedure is scheduled by the interrupt handler after each I/O completion. During normal operation, it is activated by higher levels using the SENDdatagram event or by the interrupt handler if attention interrupts have occurred. During initialization, interrupts cause the next I/O operation in the startup sequence to be initiated, and send Datagram events are ignored. During normal operation, both attention interrupts and send datagram events cause a process Ceti event to occur, which places any output into the correct buffers and passes any input to the proper protocol software.

HYPERchannel Driver

The HYPERchannel driver was taken from TCTOPC3 PASCAL (Version 1.1) and modified to support HYPERchannel. There are several major differences between the IBM 8232 (supported by TCTOPC3) and A22x (370 HYPERchannel adapter).

The IBM 8232 operates like a gateway supporting various LAN attachments, such as Ethernet, token-ring, and Proteon. TCTOPC3 implements multiple packet blocking and media interface headers for the IBM 8232. Packets to and from the

IBM 8232 have the IP-packets encapsulated in the media-interface protocol packets. Multiple packets can also be transferred in an IBM 8232 block transfer. An IBM 8232 block consists of one or more media-interface encapsulated packets, prefixed by a halfword index field. The end of the block is indicated by a halfword index field of zero.

TCTOA22 PASCAL implements the following modifications to TCTOPC3.

- An A220 block starts with the HYPERchannel basic 16-bit message encapsulation. For more information, see Figure 19.
- Media-interface encapsulation is not performed; however, HYPERchannel-block encapsulation is performed.
- Only a single packet is transferred for each block.

TCTOA22 implements basic message encapsulation with an IP packet starting at displacement +16 (for example, message header +9 = X'10' and +11 = X'04'). Figure 19 shows the basic 16-bit message encapsulation.

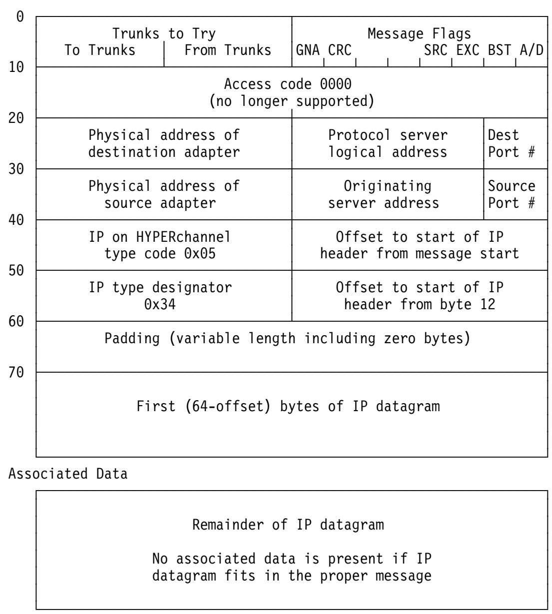


Figure 19. Basic HYPERchannel 16-bit Message Encapsulation

Two enhancements to the basic 16-bit HYPERchannel encapsulated information are supported, depending on schedules and assignment of the MESSAGETYPE field.

- Packet-blocking

  Assuming that you assigned a unique MESSAGE TYPE field, TCTOA22 conditionally implements packet blocking using the IBM 8232 model (halfword length fields terminated with a length field of zero). This enhancement requires your installation to specify block or nonblocking mode.

- SLS/720 Datagram Mode

  SLS/720 implements an extended message header for datagram mode that is not directly interoperable with either the 16-bit or 32-bit mode encapsulation standard described in RFC1044. It incorporates some aspects of both 16-bit and 32-bit modes, using the first 16 bits to address the SLS/720 in the local domain and the second 16 bits to address the adapter in the remote domain (with a pair of SLS/720s connecting the two domains using an RS449/TELCO link).

IUCV Links

At present, IUCV links support two types of IUCV communication: Passthrough Virtual Machine (PVM) IUCV and System Network Architecture (SNA) IUCV. They differ only in the connect procedure.

PVM IUCV

There are two types of PVM IUCV connections:

- Remote
- Local.

Remote PVM IUCV: The CONNECT request for Remote PVM IUCV contains the following two fields:

Field	Description
VM ID	The *VM ID* of the CONNECT request is the ID of a local virtual machine.
user	The *user* of the CONNECT request is the user of a local virtual machine. The format of the user field in the CONNECT request is shown in Figure 20.

```
                      1                 8
                     ┌───────────────────┐
1st doubleword       │ Remote PVM node   │
                     ├───────────────────┤
2nd doubleword       │ Remote TCPIP VM ID│
                     └───────────────────┘
```

Figure 20. Format of the User Field for a CONNECT Request

The time-out is set for one minute, because a response (COMPCONN or SERVER) should occur within that time. If the time-out expires, the connection is disconnected and retried later.

If a PENDCONN interrupt is received while waiting for a response to a CONNECT, a conflict can occur. The conflict is resolved by using the IucvOurPvmNode field. If the PVM node name is lower in the collating sequence than the remote node, the CONNECT request is abandoned, and the pending incoming connection request is served. If the PVM node name is higher in the collating sequence than the remote node, the pending incoming connection request is abandoned, and the CONNECT request is served.

Local PVM IUCV: The CONNECT request for Local PVM IUCV contains the following two fields:

Field	Description
VM ID	The *VM ID* of the CONNECT request is the ID of another TCP/IP user.
user	The *user* of the CONNECT request is the user of a local virtual machine. The format of the user field in the CONNECT request is shown in Figure 21.

	1 ... 8
1st doubleword	XYZZY
2nd doubleword	XYZZY

Figure 21. Format of the User Field for a Local IUCV CONNECT Request

Local IUCV links are considered to be a PVM IUCV link.

SNA IUCV

The CONNECT request for SNA IUCV contains the following two fields:

Field **Description**

VM ID The *VM ID* of the CONNECT request is the ID of another TCP/IP user.

user The *user* of the CONNECT request is the user of a local virtual machine.

The format of the user field in the CONNECT request is shown in Figure 22.

	1 ... 8
1st doubleword	SNALINK Program Name (usually SNALINK)
2nd doubleword	Remote LU Name

Figure 22. Format of the User Field for an SNA IUCV CONNECT Request

If the local SNALINK machine is the SNA PLU, there should be a short response time. If it is the SNA SLU, then the SNALINK machine does not respond until it receives a BIND from the SNA PLU. Therefore, do not set a time-out while waiting for a response to your CONNECT, because the SNALINK machine does not initiate a connect in this case.

When communicating over an established path, blocks up to 32K are sent and received. The blocks contain packets prefixed by block headers. Each packet is preceded by a halfword block header that contains the offset within the block of the next block header. A zero block header indicates the end of the block. Figure 23 shows a block containing a 10-byte packet followed by a 20-byte packet.

	12	10-byte packet	34	20-byte packet	00
Offset (dec):	0	2	12	14	34

Figure 23. IUCV Block Header

The PVM and SNALINK machines do not look at individual packets. They send the block as a unit to the peer TCP/IP machine using the PVM or SNA network. This driver only issues one IUCV SEND, and waits for the COMPMSG interrupt before issuing the next SEND. The PVM or SNALINK machine can have more than one outstanding SEND through SNALINK.

Chapter 6. Diagnosing the Problem

This chapter describes how to diagnose problems associated with TCP/IP and its interfaces. Different scenarios are used to illustrate a systematic approach to solving TCP/IP problems, although it is unlikely that they will duplicate exactly the problems you encounter.

The scenarios presented in this chapter include the inability to connect to a TCP/IP node, and failure of the HYPERchannel interface and the SNA IUCV connection. Each scenario describes the problem, explains the symptoms associated with the problem, outlines the steps necessary to determine the nature of the problem, and suggests recovery procedures for you to implement.

For each scenario, the following configuration is used:

Nodes and Addresses	Configuration Setting
Local node name	LOCAL1
Local node IP address	1.2.3.4
Remote node name	REMOTE1
Remote node IP address	1.2.4.1

Unable to Connect to TCP/IP Node

This section describes a failure to establish a Telnet connection to a TCP/IP node.

Description of the Problem

You attempt to activate a Telnet connection to a remote node, REMOTE1, but the system returns an “Invalid or unknown node” message.

Symptom

When you execute the following TELNET command, the system returns the following message:

```
TELNET REMOTE1

Host 'REMOTE1' Unknown.
```

Problem Determination

The system returns the `Host host_name Unknown` message, because the node is not defined in the HOST LOCAL file in VM, the node is not defined in the Domain Name System (DNS), or the host resides in a domain other than that specified in the TCPIP DATA file.

If you are unsure whether the REMOTE1 host resides in your domain, try specifying the fully-qualified name, including both the host name and domain name.

If you use Domain Name Server (DNS) at your site, check the DNS database for REMOTE1 and verify that the IP address is correct.

Another method of narrowing down the possible problem areas is to use the PING command to see if any communications with the remote system are possible. The PING command sends a string to the given destination and informs you of the message's status. It provides an efficient method for determining whether your configuration is correct. The destination may be specified by its name or by its IP address. The command is issued as follows:

```
PING 1.2.4.1
     or
PING REMOTE1
```

The possible errors from the PING command invocation and the probable causes of these errors are:

- **HOST UNKNOWN** - Name server problem (if host name was used) or problem with the HOSTS LOCAL file.
- **DESTINATION UNREACHABLE** - This indicates that the name (if specified) was successfully resolved, but there is no route that will allow access to that host or network.

  Use the NETSTAT GATE command to verify that the 1.2.4 subnet is readable. If not, check the GATEWAY statements in the PROFILE TCPIP file in VM. The GATEWAY statement defines how to connect to an external network. In this scenario, you should find the entry 1.2.4.

  If you are using dynamic routing (RouteD), verify that all routing daemons are operating, and that BSDrouting parms are correct in the PROFILE TCPIP.
- **TIMEOUT** - Numerous error conditions are possible in this case. It could be that the remote host is down, network congestion prevented the return of the PING reply, or the reply came back after the timeout period. Further analysis is required, focusing on the possible conditions.

PING—Sending an Echo Request to a Foreign Host

The PING command sends an echo request to a foreign host to determine if the system is accessible. PING uses ICMP as its underlying protocol.

PING Command

The TCP/IP User’s Guide has the complete PING command format.

Resolving the PING Command Problems

The echo request sent by the PING command does not guarantee delivery. More than one PING command should be sent before you assume that a communication failure has occurred.

A foreign host can fail to respond even after several PING commands. This can be caused by one of the following situations:

- The foreign host may not be listening to the network.
- The foreign host may be inoperative, or some network or gateway leading from the user to the foreign host may be inoperative.
- The foreign host may be slow because of activity.
- The packet may be too large for the foreign host
- The routing table on the local host may not have an entry for the foreign host.

Use additional PING commands to communicate with other foreign hosts in the network to determine the condition that is causing the communication failure.

However, you need to know the network topology to determine the location of the failure. Issue the PING commands in the following order, until the failure is located:

1. Send a PING command to your local host.
2. Send a PING command to a host (other than your local host) on your local network.
3. Send a PING command to each intermediate node that leads from your local host to the foreign host, starting with the node closest to your local host.

A successful PING command, sent to a different host on the same network as the original host, suggests that the original host is down, or that it is not listening to the network.

If you cannot get echoes from any host on that network, the trouble is usually somewhere along the path to the foreign hosts. Direct a PING command to the gateway leading to the network in question. If the PING command fails, continue to test along the network from the target, until you find the point of the communication breakdown.

Failure of the HYPERchannel Interface

This scenario describes the failure of a HYPERchannel driver, during which disruption of the channel interface stops data transmittal between processors.

Description of the Problem

HYPERchannel is a high-speed extension of a channel interface between physically distinct processors. This interface is similar to Ethernet or token-ring LANs, defined according to 802 IEEE standards.

A HYPERchannel failure is difficult to diagnose, because it can result from problems with software or hardware developed by different companies.

Symptom

When a HYPERchannel interface fails, it appears as a channel failure to the host. To quickly determine if a HYPERchannel interface has failed, use a host-based channel program, such as NetView® or the Event Reporting Error Program (EREP). For example, NetView generates a real-time alert if the necessary filters are set. This alert can automatically trigger a number of actions ranging from displaying a highlighted message on the NetView screen to taking a series of automated, corrective steps.

Problem Determination

You should use EREP to analyze a hardware error and determine its source in the VM environment. Although EREP is limited in diagnosing a HYPERchannel failure, it can isolate the problem to a HYPERchannel (sub)channel.

If the HYPERchannel has failed, or if a problem is suspected, the primary diagnostic aid available for use in the VM environment is a TCP/IP level trace.

A MORETRACE HCH can be initiated to trace HYPERchannel activity. The second-level trace should be used as opposed to just TRACE since the latter traces only errors, while MORETRACE traces all activity. In analyzing the resultant trace output, it is helpful to bear in mind that HYPERchannel transmission problems on the local LAN will normally be reflected via A220 unit check and sense information. Transmission problems involving remote LANs (via link adapters, 710, 715, 720, 730, etc.) may reflect problems with fault messages, since the A220 part of the operation may have already completed.

Since the HYPERchannel hardware is dedicated to the TCPIP virtual machine, the tracing facilities present in native VM can also be used to aid in problem determination.

Recovery

If a HYPERchannel hardware problem is evident or suspected by examination of trace and/or EREP output, then the HYPERchannel driver should be stopped using the OBEYFILE interface and the device taken off-line. The trace information (particularly the sense codes) and possibly the EREP data should be made available to the hardware CE to assist in problem analysis. Once the problem has been resolved, the NETSTAT CP and OBEYFILE interfaces can be used to reactivate the HYPERchannel driver.

If the problem cannot be positively identified as hardware-related, stop and restart the HYPERchannel driver via the OBEYFILE interface, ensuring that “full” tracing is activated. If the problem does not clear, contact the IBM Support Center. Ensure that a trace of the HYPERchannel activity is available for submission as supporting documentation of the problem.

Failure of an SNA IUCV Connection

The SNA IUCV connection communicates with other SNA nodes and is useful for interfacing with a token-ring or X.25 NPSI configuration.

Description of the Problem

An SNA IUCV connection failure appears as if a device is lost, and the session between the nodes is disrupted. Use NetView or EREP to identify an SNA IUCV failure.

Symptom

An SNA IUCV connection failure signals either a hardware failure or a session error, depending on the status of the connection across the interface. If an active session is using the connection, the SNA IUCV failure is classified as a session error and a session-level failure is generated. If a connectionless data transport fails, the SNA IUCV failure is classified as a hardware failure of the data transport and a link-level failure is generated by the access method.

When an SNA IUCV connection is disrupted, it is detected by the application that is sending or receiving data, or by the communication software or hardware. For example, if you are using UDP or ICMP connectionless data transport, the datagram detects the failure. If an active session is in progress, an SNA or TCP connection detects the failure.

Problem Determination

Determining the cause of an SNA IUCV failure depends on whether it is a session error or hardware failure.

Session Error

Use a logical monitoring system, such as NetView, to determine the cause of a session error. NetView generates a real-time alert if the necessary filters are set. This alert notifies the network operator by displaying a highlighted message on the NetView console. This message lists the session partners, which allows you to determine where the failure occurred. Using NetView, you can:

- View the network and the specific interface
- Proceed through several layers of screens to pinpoint the source of the problem
- Test the interface for operability in most cases.

For more information about NetView's diagnostic capabilities, see *NetView at a Glance*.

If you are using an X.25 NPSI configuration, loss of the CTCP in your host can cause a session error. The default name for the CTCP is TCPIPX25. The CTCP operates through the X.25 NPSI GATE (Generalized Access to X.25 Transport Extension) and provides a flexible interface between the host and the simulated LU in X.25 NPSI.

You can activate an internal trace for TCPIPX25 by putting a TRACE statement in the X25IPI CONFIG file. Use the DATA option on the TRACE statement and specify debug flags to view the CTCP internally. At a minimum, specify the following debug flags:

Flag	Description
0	This flag is set to 0.
1	This flag is set to 0.
2	This flag traces the IUCV interface and is set to 1.
3	This flag traces the VTAM® interface and is set to 1.
4	This flag is set to 0.
5	This flag is set to 0.
6	This flag is set to 0.
7	This flag is set to 0.

For more information about the TRACE statement, see the *TCP/IP Planning and Customization*.

Hardware Failure

The operating system or access method can detect a hardware failure. When a hardware failure occurs, the operating system displays a message and writes it to a system error log, such as EREP. Analyze the error log to determine what hardware component failed and why.

Recovery

The steps you take to recover the SNA IUCV link depend on your network configuration and the cause of the failure. Once you have determined the cause, you can use NetView to recover the SNA IUCV link:

NetView can perform the following enhanced error recovery procedures:

- Highlights the error message on the NetView console so that it does not scroll off the screen
- Creates automated recovery procedures
- Forwards the alert to the appropriate focal point.

Chapter 7. TCP/IP Traces

This chapter describes how to activate traces and direct the output to a file or the screen. Single and group processes are also described and samples of trace output are shown.

Debugging in VM

There are no special TCP/IP options or invocation parameters that are specifically directed toward VM-specific debugging activities. Since all of the servers are implemented as virtual machines, normal VM debugging tools are available for use in problem analysis.

Executing Traces

Varying levels of tracing of virtual machine activity are available for use in the VM environment. This tracing is activated through the use of the CP TRACE command. Refer to the *CP Command Reference* publication for more information on the use of these commands. The scope of the processing that one traces by virtue of these commands should be selected judiciously. Portions of TCP/IP processing are very timing-dependent. Excessive tracing can introduce connection failures due to time-out limits being exceeded.

Activating Traces

There are two levels of detail for run-time traces: first-level and second-level traces. These levels are also referred to as basic and detailed traces. Second-level traces provide more detailed information than first-level traces. Each internal TCP/IP process can be independently selected for first-level tracing or for the additional level of detail provided by second-level tracing.

Use of the TRACEONLY statement restricts TCP/IP stack tracing to particular users, devices, or IP addresses.

Activation of tracing can be accomplished by either including a list of processes to be traced in the TCPIP profile or by using the OBEYFILE command to manipulate the trace specifications dynamically. A combination of these methods can also be used to vary the amount of tracing performed as needs dictate. Both levels of tracing are eligible for manipulation by these means. The default name of the profile is PROFILE TCPIP. For more information about OBEYFILE, see the *TCP/IP Planning and Customization*.

First-Level Trace

To activate and deactivate first-level traces, use the TRACE and NOTRACE commands, respectively.

The following is the format of the TRACE command:

```
           ┌─ALL──────────────────┐
           │ ┌──────────────────┐ │
►►──TRACE──┴─▼─process_name─────┴─┴──────────────────────────────────►◄
```

The parameters of the TRACE command are:

Parameter	Description
process_name	Is the set of new process names to be activated by TRACE. The new set replaces any previous set of selected processes.
ALL	Is the default value and activates the ALL set of process names.

The following is the format of the NOTRACE command:

```
                  ┌─ALL──────────────┐
                  │ ┌──────────────┐ │
►►──NOTRACE───────┴─▼─process_name─┴─┴──────────────────────────────────►◄
```

The parameters of the NOTRACE command are:

Parameter	Description.
process_name	Is the set of process names to be deactivated by NOTRACE. NOTRACE deactivates a set of process names previously started by a TRACE command.
ALL	Is the default value and deactivates the entire trace process, closing any active trace file.

Second-Level Trace

To activate and deactivate second-level traces, use the MORETRACE and LESSTRACE commands, respectively.

The following is the format of the MORETRACE command:

```
                  ┌─ALL──────────────┐
                  │ ┌──────────────┐ │
►►──MORETRACE─────┴─▼─process_name─┴─┴──────────────────────────────────►◄
```

The parameters of the MORETRACE command are:

Parameter	Description
process_name	Is the set of process names to be activated by MORETRACE. MORETRACE activates second-level traces.
ALL	Is the default value and activates the ALL set of process names.

The following is the format of the LESSTRACE command:

```
                  ┌─ALL──────────────┐
                  │ ┌──────────────┐ │
►►──LESSTRACE─────┴─▼─process_name─┴─┴──────────────────────────────────►◄
```

The parameters of the LESSTRACE command are:

Parameter	Description
process_name	Is the set of process names to be deactivated by LESSTRACE. LESSTRACE deactivates a set of process names previously started by a MORETRACE statement.
ALL	Is the default value and deactivates the entire second-level trace process.

Figure 41 on page 77 shows a sample trace using LESSTRACE.

Directing Output

You can send trace output either to a file or to the screen.

Output Directed to a File

The FILE command creates a file and writes the current trace output to it.

VM FILE Command: The following is the format of the FILE command.

```
                               ┌─A────────┐
►►──FILE──filename──filetype───┼──────────┼──────────────────────────────►◄
                               └─filemode─┘
```

The parameters of the FILE command are:

Parameter	Description
filename	The name of the file to which the output is written.
filetype	The file type of the file to which the output is written.
filemode	The file mode where the file is written.

Output Directed to the Screen

The SCREEN command sends trace output to the TCPIP user console, closing any active disk trace file.

```
►►──SCREEN───────────────────────────────────────────────────────────────►◄
```

The SCREEN command has no parameters.

For more information about trace activation and output statements, see the *TCP/IP Planning and Customization*.

Process Names

The process names entered in the TRACE, NOTRACE, MORETRACE, and LESSTRACE commands are used in conjunction with the internal procedures listed in “Internal Procedures” on page 35. There are single process names and group process names. A group process combines several single processes into one process name.

You should be as specific as possible when entering process names, because some process names yield voluminous output. For example, the output from the

MORETRACE ALL command can be overwhelming. Also, you should not execute traces unnecessarily, because it can adversely affect system response time.

Note: In the sample traces shown in this chapter, the home addresses could be:

- 9.67.58.233
- 9.67.58.39
- 9.67.58.193

There can be more than one name for a process. The following sections list the different forms of the process name where appropriate.

Single Process Names

Single process names involve only one event. They are usually not as helpful as entering a group process name or several single process names, because several processes can give complementary information, which in some situations, could be matched with a CCW trace, if required.

ARP

The ARP trace provides information about the ARP process, ARP table contents, ARP packets, and ARP requests.

Figure 24 shows a sample trace of the ARP process and the ARP table content using ARP and Parse-Tcp options.

Note: The event `Arp adds translation...` indicates when ARP translation information is added to the ARP table. `ARPop` is the operation field in the ARP packet. A value of 1 is an ARP request, and a value of 2 is an ARP response.

```
ScanTranslationTable: Scanning for ARP entries older than 300 seconds
ScanTranslationVisitNode: NOT deleting entry for link ETH1 address 9.67.58.39
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.226
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.233
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.234
ScanTranslationVisitNode: NOT deleting entry for link TR2 address 9.67.58.193
ScanTranslationTable: Scanning for ARP entries older than 300 seconds
ScanTranslationVisitNode: NOT deleting entry for link ETH1 address 9.67.58.39
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.226
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.233
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.234
ScanTranslationVisitNode: NOT deleting entry for link TR2 address 9.67.58.193
```

Figure 24 (Part 1 of 3). A Sample of an ARP Trace

```
Arpin: Processing Arp packet:
   ArpHardwareType: 6
   ArpProtocolType: 2048
   ArpHardwareLen: 6
   ArpProtocolLen: 4
   ArpOp:          1
   ArpSenderHardwareAddr: 10005A140138
   ArpSenderInternetAddr: 9.67.58.225
   ArpTargetHardwareAddr: C53400D7C530
   ArpTargetInternetAddr: 9.67.58.234
Arpin: Processing Arp packet:
   ArpHardwareType: 6
   ArpProtocolType: 2048
   ArpHardwareLen: 6
   ArpProtocolLen: 4
   ArpOp:          1
   ArpSenderHardwareAddr: 10005A140138
   ArpSenderInternetAddr: 9.67.58.225
   ArpTargetHardwareAddr: C49C00D7C498
   ArpTargetInternetAddr: 9.67.58.234
ScanTranslationTable: Scanning for ARP entries older than 300 seconds
ScanTranslationVisitNode: NOT deleting entry for link ETH1 address 9.67.58.39
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.226
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.233
ScanTranslationVisitNode: Deleting entry for link TR1 address 9.67.58.234, age 325 seconds
ScanTranslationVisitNode: NOT deleting entry for link TR2 address 9.67.58.193
ArpReqSent: ArpEnvelopeQueue is now:
   1 packets queued waiting for ARP reply
      First Hop 9.67.58.234, Seconds on queue 0
Arpin: Processing Arp packet:
   ArpHardwareType: 6
   ArpProtocolType: 2048
   ArpHardwareLen: 6
   ArpProtocolLen: 4
   ArpOp:          2
   ArpSenderHardwareAddr: 10005A250858
   ArpSenderInternetAddr: 9.67.58.234
   ArpTargetHardwareAddr: 10005A6BB806
   ArpTargetInternetAddr: 9.67.58.233
Arp adds translation9.67.58.234 = IBMTR: 10005A250858
ArpReplyReceived: ArpEnvelopeQueue is now:
   0 packets queued waiting for ARP reply
ScanTranslationTable: Scanning for ARP entries older than 300 seconds
ScanTranslationVisitNode: NOT deleting entry for link ETH1 address 9.67.58.39
ScanTranslationVisitNode: Deleting entry for link TR1 address 9.67.58.226, age 310 seconds
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.233
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.234
ScanTranslationVisitNode: NOT deleting entry for link TR2 address 9.67.58.193
Arpin: Processing Arp packet:
   ArpHardwareType: 6
   ArpProtocolType: 2048
   ArpHardwareLen: 6
   ArpProtocolLen: 4
   ArpOp:          1
   ArpSenderHardwareAddr: 10005A0019F5
   ArpSenderInternetAddr: 9.67.58.226
   ArpTargetHardwareAddr: F53400D7F530
   ArpTargetInternetAddr: 9.67.58.234
```

Figure 24 (Part 2 of 3). A Sample of an ARP Trace

```
Arpin: Processing Arp packet:
   ArpHardwareType: 6
   ArpProtocolType: 2048
   ArpHardwareLen: 6
   ArpProtocolLen: 4
   ArpOp:           1
   ArpSenderHardwareAddr: 10005A0019F5
   ArpSenderInternetAddr: 9.67.58.226
   ArpTargetHardwareAddr: F53400D7F530
   ArpTargetInternetAddr: 9.67.58.233
Arp adds translation9.67.58.226 = IBMTR: 10005A0019F5
ScanTranslationTable: Scanning for ARP entries older than 300 seconds
ScanTranslationVisitNode: NOT deleting entry for link ETH1 address 9.67.58.39
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.226
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.233
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.234
ScanTranslationVisitNode: NOT deleting entry for link TR2 address 9.67.58.193
```

Figure 24 (Part 3 of 3). A Sample of an ARP Trace

Figure 25 shows the MORETRACE command used in conjunction with an ARP trace.

```
ScanTranslationTable: Scanning for ARP entries older than 300 seconds
ScanTranslationVisitNode: NOT deleting entry for link ETH1 address 9.67.58.39
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.233
ScanTranslationVisitNode: NOT deleting entry for link TR2 address 9.67.58.193
ArpReqSent: ArpEnvelopeQueue is now:
   1 packets queued waiting for ARP reply
      First Hop 9.67.58.234, Seconds on queue 0
Arpin: Processing Arp packet:
   ArpHardwareType: 6
   ArpProtocolType: 2048
   ArpHardwareLen: 6
   ArpProtocolLen: 4
   ArpOp: 0
   ArpSenderHardwareAddr: 10005A250858
   ArpSenderInternetAddr: 9.67.58.234
   ArpTargetHardwareAddr: 10005A6BB806
   ArpTargetInternetAddr: 9.67.58.233
Arp adds translation9.67.58.234 = IBMTR: 10005A250858
ArpReplyReceived: ArpEnvelopeQueue is now:
   0 packets queued waiting for ARP reply
ScanTranslationTable: Scanning for ARP entries older than 300 seconds
ScanTranslationVisitNode: NOT deleting entry for link ETH1 address 9.67.58.39
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.233
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.234
ScanTranslationVisitNode: NOT deleting entry for link TR2 address 9.67.58.193
 .
 .
 .
ScanTranslationTable: Scanning for ARP entries older than 300 seconds
ScanTranslationVisitNode: NOT deleting entry for link ETH1 address 9.67.58.39
ScanTranslationVisitNode: NOT deleting entry for link TR1 address 9.67.58.233
ScanTranslationVisitNode: Deleting entry for link TR1 address 9.67.58.2  34, age 342 seconds
ScanTranslationVisitNode: NOT deleting entry for link TR2 address 9.67.58.193
```

Figure 25. A Sample of an ARP Trace Using MORETRACE

CCS

Figure 26 shows a sample of a CCS CP System Service trace. This trace indicates when a remote client has logged on using a TELNET internal client.

```
Telnet server: Conn 0:Connection opened 09/07/97 at 12:29:14
   Foreign internet address and port:  net address = 9.67.58.226, port= 1030
12:30:04 09/07/97 PCCA3 common routine KILL TCB #1000 (INTCLIEN) Foreign host aborted the
connection
    Bytes: 9313 sent, 292 received
    Segs in: 67 OK, 24 pushed
    Max use: 1 in retransmit Q
```

Figure 26. A Sample of a CCS Trace

CLAW Trace Information

The CLAW driver support includes provisions to gather trace information to assist in problem diagnosis. This tracing is supported by the CLAW process name. Activation of the CLAW trace process can be accomplished by either including the process name in the list of processes to be traced in the PROFILE TCPIP file or by using the OBEYFILE command interface.

Two levels of tracing are supported, TRACE and MORETRACE. Specifying TRACE CLAW results in the generation of the following output.

- Information about CLAW read and write channel program processing
- Start I/O and write complete notifications
- CSW information on I/O completions
- Data from Sense ID channel command execution
- Statistical information about packets (queue sizes, packet data lengths, and so forth)
- ACB information.

Figure 27 on page 60 shows an abridged section of a sample trace of the CLAW process.

```
  :
ToClaw: Acb Received:
11793672:
   Have completed I/O -> To-CLAW (from Claw interrupt handler)
      IoDevice 0A90
      Csw:
         Keys: 00, CcwAddress: 006BABB0
         Unit Status: 0C, Channel Status: 00
         Byte Count:           0
      Device AIXV3:
         Type: CLAW, Status: Sense ID on input
         Envelope queue size: 0
         Address: 0A90
         Host name: HOST
         Adapter name: PSCA
         Control task name: NONE
CLAW device AIXV3:
   Received Sense ID data: FF 30 88 61 00 00 00 on device 0A90
ToClaw: Acb Received:
11793672:
   Have completed I/O -> To-CLAW (from Claw interrupt handler)
      IoDevice 0A91
      Csw:
         Keys: 00, CcwAddress: 006BABB0
         Unit Status: 0C, Channel Status: 00
         Byte Count:           0
      Device AIXV3:
         Type: CLAW, Status: Sense ID on output
         Envelope queue size: 0
         Address: 0A90
         Host name: HOST
         Adapter name: PSCA
         Control task name: NONE
CLAW device AIXV3:
   Received Sense ID data: FF 30 88 61 00 00 00 on device 0A91
CLAW device AIXV3:
   CallSio: Starting I/O on device 0A90. First command 02
ToClaw: Acb Received:
11793984:
   Send datagram -> To-CLAW (from To-CLAW)
      Device AIXV3:
         Type: CLAW, Status: Waiting for start pkt
         Envelope queue size: 0
         Address: 0A90
         Host name: HOST
         Adapter name: PSCA
         Control task name: NONE
CLAW device AIXV3: ToClaw PackWrites: Queuesizes: 1 0
CLAW device AIXV3: ToClaw PackWrites: LengthOfData: 32
CLAW device AIXV3:
   CallSio: Starting I/O on device 0A91. First command 01
ToClaw: Acb Received:
11793984:
   Have completed I/O -> To-CLAW (from Claw interrupt handler)
```

Figure 27 (Part 1 of 2). A Sample of a CLAW Trace

```
      IoDevice 0A91
      Csw:
         Keys: 00, CcwAddress: 00001018
         Unit Status: 0C, Channel Status: 00
         Byte Count:           1
      Device AIXV3:
         Type: CLAW, Status: Waiting for start pkt
         Envelope queue size: 0
         Address: 0A90
         Host name: HOST
         Adapter name: PSCA
         Control task name: NONE
CLAW device AIXV3: ToClaw write complete.
CLAW device AIXV3: ToClaw PackWrites: Queuesizes: 0 0
ToClaw: Acb Received:
11793672:
   Have completed I/O -> To-CLAW (from Claw interrupt handler)
      IoDevice 0A90
      Csw:
         Keys: 00, CcwAddress: 00002878
         Unit Status: 00, Channel Status: 80
         Byte Count:        8208
      Device AIXV3:
         Type: CLAW, Status: Waiting for start pkt
         Envelope queue size: 0
         Address: 0A90
         Host name: HOST
         Adapter name: PSCA
         Control task name: NONE
Claw device AIXV3: System validate completed.
CLAW device AIXV3: ToClaw PackWrites: Queuesizes: 2 0
CLAW device AIXV3: ToClaw PackWrites: LengthOfData: 32
CLAW device AIXV3: ToClaw PackWrites: LengthOfData: 32
CLAW device AIXV3:
   CallSio: Starting I/O on device 0A91. First command 01
  :
```

Figure 27 (Part 2 of 2). A Sample of a CLAW Trace

Specifying MORETRACE CLAW results in the generation of the following output:

- All trace information described for TRACE CLAW, above
- Envelope and CLAW control packet information
- IP datagram information
- Read and write channel program information when I/O is started

Figure 28 on page 62 shows an abridged section of a sample trace of the CLAW process when MORETRACE is specified.

```
  :
ToClaw: Acb Received:
11777704:
   Send datagram -> To-CLAW (from To-CLAW)
      Device AIXV3:
         Type: CLAW, Status: Ready
         Envelope queue size: 0
         Address: 0A90
         Host name: HOST
         Adapter name: PSCA
         Control task name: NONE
CLAW device AIXV3: ToClaw PackWrites: Queuesizes: 0 0
ToClaw: Acb Received:
11777704:
   Have completed I/O -> To-CLAW (from Claw wait scan)
      IoDevice 0000
      Csw:
         Keys: 00, CcwAddress: 00000000
         Unit Status: 00, Channel Status: 00
         Byte Count:            0
      Device AIXV3:
         Type: CLAW, Status: Ready
         Envelope queue size: 0
         Address: 0A90
         Host name: HOST
         Adapter name: PSCA
         Control task name: NONE
CLAW device AIXV3: Received Control Packet:
   Connection Response: Version=2, Link ID=2, Correlator=0,
   Return Code=0, Work station application=TCPIP,
   Host application=TCPIP
CLAW device AIXV3: Received Control Packet:
   Disconnect: Version=2, Link ID=2, Correlator=0,
   Return Code=0, Work station application=        ,
   Host application=
CLAW device AIXV3: ToClaw PackWrites: Queuesizes: 1 0
CLAW device AIXV3: Sending envelope:
   Disconnect: Version=2, Link ID=2, Correlator=0,
   Return Code=0, Work station application=TCPIP,
   Host application=TCPIP
CLAW device AIXV3: ToClaw PackWrites: LengthOfData: 32
CLAW device AIXV3: StartClawOutputIo
CCWB at 00692D80, real address=0001FD80, data at 0069B000
OpCode=01 Address=016000 Flags=60 Length=0020
OpCode=22 Address=01FD8F Flags=60 Length=0001
OpCode=08 Address=020008 Flags=00 Length=0000
CLAW device AIXV3:
   CallSio: Starting I/O on device 0A91. First command 01
CLAW device AIXV3: ToClaw: Sio returned 0 on device 0A91
ToClaw: Acb Received:
11777600:
   Send datagram -> To-CLAW (from To-CLAW)
```

Figure 28 (Part 1 of 3). A Sample of a CLAW Trace Using MORETRACE

```
      Device AIXV3:
         Type: CLAW, Status: Ready
         Envelope queue size: 0
         Address: 0A90
         Host name: HOST
         Adapter name: PSCA
         Control task name: NONE
CLAW device AIXV3: ToClaw PackWrites: Queuesizes: 0 0
ToClaw: Acb Received:
11777704:
   Have completed I/O -> To-CLAW (from Claw interrupt handler)
      IoDevice 0A91
      Csw:
         Keys: 00, CcwAddress: 00020018
         Unit Status: 0C, Channel Status: 00
         Byte Count:            1
      Device AIXV3:
         Type: CLAW, Status: Ready
         Envelope queue size: 0
         Address: 0A90
         Host name: HOST
         Adapter name: PSCA
         Control task name: NONE
CLAW device AIXV3: ToClaw write complete.
CLAW device AIXV3: ToClaw PackWrites: Queuesizes: 0 0
ToClaw: Acb Received:
11777704:
   Have completed I/O -> To-CLAW (from Claw interrupt handler)
      IoDevice 0A90
      Csw:
         Keys: 00, CcwAddress: 0001F998
         Unit Status: 00, Channel Status: 80
         Byte Count:         8208
      Device AIXV3:
         Type: CLAW, Status: Ready
         Envelope queue size: 0
         Address: 0A90
         Host name: HOST
         Adapter name: PSCA
         Control task name: NONE
CLAW device AIXV3:
   UnpackReads: NetType 98 AdapterNumber 1 BytesToMove 156
CLAW device AIXV3: Received IP datagram:
   IP Datagram:
   version: 4
   Internet Header Length: 5 = 20 bytes
   Type of Service:Precedence = Routine
   Total Length: 156 bytes
   Identification: 3590
   Flags: May Fragment, Last Fragment
   Fragment Offset: 0
   Time To Live: 255
   Protocol: ICMP
   Header CheckSum: 42324
   Source Address: 01020301
   Destination Address: 01020302
CLAW device AIXV3: ToClaw PackWrites: Queuesizes: 0 1
```

Figure 28 (Part 2 of 3). A Sample of a CLAW Trace Using MORETRACE

```
CLAW device AIXV3: Sending envelope:
   IP Datagram:
   version: 4
   Internet Header Length: 5 = 20 bytes
   Type of Service:Precedence = Routine
   Total Length: 156 bytes
   Identification: 3590
   Flags: May Fragment, Last Fragment
   Fragment Offset: 0
   Time To Live: 60
   Protocol: ICMP
   Header CheckSum: 26709
   Source Address: 01020302
   Destination Address: 01020301
CLAW device AIXV3: ToClaw PackWrites: LengthOfData: 156
CLAW device AIXV3: StartClawOutputIo
CCWB at 00692D80, real address=0001FD80, data at 0069B000
OpCode=09 Address=016000 Flags=60 Length=009C
OpCode=22 Address=01FD8F Flags=60 Length=0001
OpCode=08 Address=020018 Flags=00 Length=0000
CLAW device AIXV3:
   CallSio: Starting I/O on device 0A91. First command 09
CLAW device AIXV3: ToClaw: Sio returned 0 on device 0A91
  :
```

Figure 28 (Part 3 of 3). A Sample of a CLAW Trace Using MORETRACE

It is not recommended that either level of CLAW tracing be activated as a normal course of business. These traces have the potential to generate large amounts of data and there is a fair amount of overhead associated with them. In sample traces of the same test traffic, MORETRACE CLAW generated more than twice as many lines of output as TRACE CLAW. Both should be used with discretion, with exploitation of MORETRACE CLAW reserved for those situations where a CLAW-related problem is evident and you wish to maximize the collection of diagnostic data.

Congestion

Figure 29 shows a sample of a TCP Congestion Control trace.

A TCP Congestion Control trace gives information about internal TCPIP congestion.

```
  .
  .
TCPUTI032I Conn 1004: TcpSlowStart: CongestionWindow now 536, was 65535. Thr
esh now 1072, was 65535. MSS 536, SndWnd 0
TCPUTI032I Conn 1004: TcpSlowStart: CongestionWindow now 536, was 536. Thres
h now 1072, was 1072. MSS 536, SndWnd 0
TCPUTI032I Conn 1004: TcpSlowStart: CongestionWindow now 536, was 536. Thres
h now 1072, was 1072. MSS 536, SndWnd 0
TCPUTI015I 11:17:49 05/28/91 TCP-request KILL TCB #1004 (USER11  ) Foreign h
ost did not respond within OPEN timeout
```

Figure 29 (Part 1 of 2). A Sample of a Congestion Trace

```
TCPUTI019I     Bytes: 1 sent, 0 acked, 0 received
TCPUTI027I     Max use: 1 in retransmit Q
TCPROU003I Conn 1004: Opening congestion win: CongestionWindow 65535, Thresh
old 65535, MSS 536, increment 536
TCPUTI032I Conn 1004: TcpSlowStart: CongestionWindow now 536, was 65535. Thr
esh now 4096, was 65535. MSS 536, SndWnd 8192
TCPROU003I Conn 1004: Opening congestion win: CongestionWindow 536, Threshol
d 4096, MSS 536, increment 536
TCPROU003I Conn 1004: Opening congestion win: CongestionWindow 1072, Thresho
ld 4096, MSS 536, increment 536
TCPDOW021I Avoiding small packet. Desired 3, Max seg 536, MaxSndWnd div 2 40
96, HowManyInUse 1
TCPDOW021I Avoiding small packet. Desired 6, Max seg 536, MaxSndWnd div 2 40
96, HowManyInUse 1
TCPDOW021I Avoiding small packet. Desired 9, Max seg 536, MaxSndWnd div 2 40
96, HowManyInUse 1
TCPROU003I Conn 1004: Opening congestion win: CongestionWindow 1608, Thresho
ld 4096, MSS 536, increment 536
TCPROU003I Conn 1004: Opening congestion win: CongestionWindow 2144, Thresho
ld 4096, MSS 536, increment 536
TCPROU003I Conn 1004: Opening congestion win: CongestionWindow 3216, Thresho
ld 4096, MSS 536, increment 536
TCPUTI015I 11:20:37 05/28/91 TCP-request KILL TCB #1004 (USER11  ) You abort
ed the connection
TCPUTI019I     Bytes: 73 sent, 2293 received
TCPUTI022I     Segs in: 19 OK
TCPUTI027I     Max use: 1 in retransmit Q
  .
  .
  .
```

Figure 29 (Part 2 of 2). A Sample of a Congestion Trace

CONSISTENCYCHECKER or CONSISTENCY_CHECKER

The Consistency Checker or Consistency_Checker trace provides information about a TCPIP user's internal consistency, including the number of buffers allocated and the number of active connections. The Consistency Checker is not enabled unless the ASSORTEDPARMS configuration statement option CHECKCONSISTENCY has been specified.

Figure 30 shows a sample of a Consistency Checker trace.

```
PCCA3 device LCS1: PCCA reports home hardware address 02608C1A73F5 for link ETH1
PCCA3 device LCS1: PCCA reports home hardware address 10005A6BB806 for link TR1
PCCA3 device LCS1: PCCA reports home hardware address 10005A6BAFDF for link TR2
```

Figure 30 (Part 1 of 3). A Sample of a CONSISTENCYCHECKER Trace

```
Maximum recent queues: Timer = 5, ToDo = 4
ToTcpBuff buffers allocated: InUse conns = 0, NotInUse conns = 0
ToCpBuff    buffers allocated: InUse conns = 0 NotInUse conns = 0
FromTcpBuff buffers allocated: InUse conns = 0 NotInUse conns = 0
FromCpBuff  buffers allocated: InUse conns = 0, NotInUse conns = 0
CheckTree traversing tree IP routing via TreeTraverse
NodeCount 5, tree head says 5
CheckTree traversing tree IP routing via NormalTraverse
NodeCount 5, tree head says 5
Height 4
Free count 295, tree count 5, total 300, expected 300

CheckTree traversing tree TCP connections via TreeTraverse
NodeCount 5, tree head says 5
CheckTree traversing tree TCP connections via NormalTraverse
NodeCount 5, tree head says 5
Height 4
Free count 251, tree count 5, total 256, expected 256

CheckTree traversing tree UDP ports via TreeTraverse
NodeCount 4, tree head says 4
CheckTree traversing tree UDP ports via NormalTraverse
NodeCount 4, tree head says 4
Height 3
Free count 26, tree count 4, total 30, expected 30

CheckTree traversing tree Address translation via TreeTraverse
NodeCount 5, tree head says 5
CheckTree traversing tree Address translation via NormalTraverse
NodeCount 5, tree head says 5
Height 4
Free count 1495, tree count 5, total 1500, expected 1500

17:36:23 10/24/97 PCCA3 common routine KILL TCB #1001 (FTPSERVE) Foreign host ab
orted the connection
    Bytes: 409 sent, 80 received
    Segs in: 18 OK
    Max use: 2 in retransmit Q
Telnet server: Conn 0:Connection opened 10/24/90 at 17:36:35
   Foreign internet address and port:  net address = 9.67.58.225, port= 1071
Telnet server: Conn 1:Connection opened 10/24/90 at 17:37:17
   Foreign internet address and port:  net address = 9.67.43.126, port= 3213
Maximum recent queues: Timer = 7, ToDo = 3
ToTcpBuff buffers allocated: InUse conns = 0, NotInUse conns = 0
ToCpBuff    buffers allocated: InUse conns = 0 NotInUse conns = 0
FromTcpBuff buffers allocated: InUse conns = 0 NotInUse conns = 0
FromCpBuff  buffers allocated: InUse conns = 0, NotInUse conns = 0
CheckTree traversing tree IP routing via TreeTraverse
NodeCount 5, tree head says 5
CheckTree traversing tree IP routing via NormalTraverse
NodeCount 5, tree head says 5
Height 4
Free count 295, tree count 5, total 300, expected 300

CheckTree traversing tree TCP connections via TreeTraverse
NodeCount 8, tree head says 8
CheckTree traversing tree TCP connections via NormalTraverse
NodeCount 8, tree head says 8
Height 6
Free count 248, tree count 8, total 256, expected 256
```

Figure 30 (Part 2 of 3). A Sample of a CONSISTENCYCHECKER Trace

```
CheckTree traversing tree UDP ports via TreeTraverse
NodeCount 4, tree head says 4
CheckTree traversing tree UDP ports via NormalTraverse
NodeCount 4, tree head says 4
Height 3
Free count 26, tree count 4, total 30, expected 30

CheckTree traversing tree Address translation via TreeTraverse
NodeCount 5, tree head says 5
CheckTree traversing tree Address translation via NormalTraverse
NodeCount 5, tree head says 5
Height 4
Free count 1495, tree count 5, total 1500, expected 1500

CcbGarbageCollect disposing of CCB for client TCPUSR13
17:38:18 10/24/97 TCP-request KILL TCB #1006 (FTPSERVE) OK
    Bytes: 11457 sent, 2 received
    Segs in: 5 OK
    Max use: 3 in retransmit Q
Telnet server: Conn 2:Connection opened 10/24/97 at 17:39:21
   Foreign internet address and port:  net address = 9.67.58.225, port= 1072
Telnet server: Conn 3:Connection opened 10/24/97 at 17:41:27
   Foreign internet address and port:  net address = 9.67.58.225, port= 1073
```

Figure 30 (Part 3 of 3). A Sample of a CONSISTENCYCHECKER Trace

ELANS

The ELANS trace provides ELANS adapter status, ARP translations, received frame type, LLC frame type, and the entering ELANS procedure name.

ICMP

The ICMP trace provides information about the ICMP packets sent from the networks, and gives the IP addresses or names if the names are in the HOST LOCAL file. Figure 31 shows a sample of an ICMP trace. ICMP was specified in the TRACE statement in the PROFILE TCPIP file.

```
PCCA3 initializing:
 Device LCS1:
    Type: LCS, Status: Not started
    Envelope queue size: 0
    Address: 0560
TCP-IP initialization complete.
PCCA3 device LCS1: Received startup packet
   IP-up sees ICMP datagram, code 3, subcode: 3, source: Loopback, dest: Loopback, len: 36
PCCA3 device LCS1: PCCA reports home hardware address 02608C1A73F5 for link ETH1
PCCA3 device LCS1: PCCA reports home hardware address 10005A6BB806 for link TR1
PCCA3 device LCS1: PCCA reports home hardware address 10005A6BAFDF for link TR2
   IP-up sees ICMP datagram, code 0, subcode: 0, source: RALVMM, dest: SA23, len: 256
   IP-up sees ICMP datagram, code 3, subcode: 3, source: Loopback, dest: Loopback, len: 36
   IP-up sees ICMP datagram, code 0, subcode: 0, source: APOLLO, dest: SA23, len: 256
```

Figure 31. A Sample of an ICMP Trace

ILANS

The ILANS trace provides ILANS adapter status, ARP translations, received frame type, LLC frame type, and the entering ILANS procedure name. If you run the ILANS trace from the PROFILE TCPIP, it shows the adapter address.

Figure 32 shows a sample of an ILANS trace.

```
CETI device initializing:
 Device ILANS1:
    Type: ILANS, Status: Not started
    Envelope queue size: 0
    Address: 0240
TCP-IP initialization complete.
ILANS device ILANS1: Entering IlansLevel2Init
ILANS device ILANS1: Sending SET_NET_PARMS.request
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS device ILANS1: SET_NET_PARMS.confirm:
Completion status 00000000, Extended Status 00000000
ILANS device ILANS1: Sending DLM_RTV_ATTRIB.request
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS device ILANS1: CIOA MAC primitive type is CC31
ILANS device ILANS1: DLM_RTV_ATTRIB.confirm: Completion status 00000000
Node address: 10005A4209A0
ILANS device ILANS1: Sending DL_ACTIVATE_SAP.request
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS device ILANS1: CIOA LLC primitive type is CC0D
ILANS device ILANS1: DL_ACTIVATE_SAP.confirm: Completion status 00000000, PSapId '0007120C'X
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS device ILANS1: Entering DispatchTr: EtherType '0806'X
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS device ILANS1: Entering DispatchTr: EtherType '0806'X
Arp adds translation9.67.58.234 = IBMTR: 10005A250858
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS device ILANS1: Entering DispatchTr: EtherType '0800'X
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS device ILANS1: Entering DispatchTr: EtherType '0800'X
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS device ILANS1: Entering DispatchTr: EtherType '0806'X
Arp adds translation9.67.58.226 = IBMTR: 10005A0019F5
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS device ILANS1: Entering DispatchTr: EtherType '0800'X
CETI shutting down:
 Device ILANS1:
    Type: ILANS, Status: Ready
    Envelope queue size: 0
    Address: 0240
Final result of DoHaltIo for device 0242: Cond code 0
Final result of DoHaltIo for device 0243: Cond code 0
```

Figure 32. A Sample of an ILANS Trace

INITIALIZE

The initialization trace provides information about TCPIP initialization. The return codes for the AUTOLOG and FORCE commands are also provided.

Figure 33 shows a sample of an INITIALIZE trace using MORETRACE. The information provided by MORETRACE includes a list of autologged clients, authorizations, and reserved ports and a table of local ports.

```
TCPIP    AT GDLVM7  VIA RSCS     09/07/97 11:09:58 EST      FRIDAY
VM TCP/IP V2R4
   Initializing...
UnlockAll issuing "CP UNLOCK TCPIP 0 DFF"
COMMAND COMPLETE
LCS devices will use diagnose 98 real channel program support
Trying to open GDLVM7 TCPIP *
Trying to open PROFILE TCPIP *
Using profile file PROFILE TCPIP  *
PCCA3 initializing:
 Device LCS1:
    Type: LCS, Status: Not started
    Envelope queue size: 0
    Address: 0560
Telnet server: Using port 23
Telnet server: No inactivity timeout
Telnet server: Every 1800 seconds a timing mark option packet will be sent.
******************************************
Log of IBM TCP/IP Telnet Server Users started on 09/07/97 at 11:10:43

State after initialization:

Client list: Queue size = 19
   13610776:
   PrevCCB: Client list
   NextCCB: 13611528
   Authorization: Monitor, Informed
   No outstanding VMCF messages
   Handled notices: none
   Last touched: 20
   Login name: OPERATOR
   Notice list: empty
   Reserved socket list: empty
   VMCF error count: 0

   13611528:
   PrevCCB: 13610776
   NextCCB: 13612280
   Authorization: Monitor, Informed
   No outstanding VMCF messages
   Handled notices: none
   Last touched: 20
   Login name: TCPMAINT
   Notice list: empty
   Reserved socket list: empty
   VMCF error count: 0
   .
   .
   .
```

Figure 33 (Part 1 of 3). A Sample of an INITIALIZE Trace Using MORETRACE

```
13600336:
PrevCCB: 13599584
NextCCB: Client list
No outstanding VMCF messages
Handled notices: Buffer space available, Connection state changed, Data deliv
ered, User-defined notification, Datagram space available, Urgent pending, UDP d
ata delivered, UDP datagram space available, Other external interrupt received,
User delivers line, User wants attention, Timer expired, FSend response, FReceiv
e error, RawIp packets delivered, RawIp packet space available, IUCV interrupt,
I/O interrupt, Resources available for TcpOpen, Resources available for UdpOpen,
Ping response or timeout, SMSG received
Last touched: 41
Login name: INTCLIEN
Notice list: empty
Reserved socket list: Queue size = 1
   5104192:
   PrevScb: 13601048
   NextScb: 13601048
   Client: INTCLIEN
   Connection list: Queue size = 1
      4671640:
      PrevTcb: 5104256
      NextTcb: 5104256
      Backoff count 0
      Client: INTCLIEN
      ClientRcvNxt: 0
      ClientSndNxt: 600188177
      CongestionWindow: 65535, SlowStartThreshold: 65535
      Local connection name: 1000
      Foreign socket: net address = *, port= Unspecified
      Sender frustration level: Contented
      Incoming segment queue: Queue size = 1
         5732096:
         PrevDataBuffer: 4672528
         NextDataBuffer: 4672528
         First Unused Sequence Number: 0
         Offset of last byte delivered: 0
         Offset of last byte received: 0
         Sequence number of first byte: 0

      Incoming window number: 0
      Initial receive sequence number: 0
      Initial send sequence number: 600188176
      Maximum segment size: 536
      Local socket: net address = *, port= TELNET (23)
      Outgoing window number: 0
      Precedence: Routine
      RcvNxt: 0
      Round-trip information:
         Smooth variance: 1.500
         ReplaceSmooth TRUE
      SndNxt: 600188176
      SndUna: 600188176
      SndWl1: 0
      SndWl2: 0
      SndWnd: 0
      MaxSndWnd: 0
      State: Listen
      No pending TCP-receive

   Local socket: net address = *, port = TELNET (23)  * permanently reserved*
      * autolog client *

VMCF error count: 0
```

Figure 33 (Part 2 of 3). A Sample of an INITIALIZE Trace Using MORETRACE

```
The local port hash table:
   20 = FTPSERVE has 0 TCBs for socket *.FTP default data (20) *Perm   21 = FTPS
ERVE has 0 TCBs for socket *.FTP control (21) *Perm *Autolog   23 = INTCLIEN has
 1 TCBs for socket *.TELNET (23) *Perm *Autolog   25 = SMTP has 0 TCBs for socke
t *.SMTP (25) *Perm *Autolog   53 = NAMESRV has 0 TCBs for socket *.DNS (53) *Pe
rm *Autolog   53 = NAMESRV has 0 TCBs for socket *.DNS (53) *Perm *Autolog   161
 = SNMP has 0 TCBs for socket *.161 *Perm *Autolog   162 = SNMPQE has 0 TCBs for
 socket *.162 *Perm *Autolog   512 = REXECD has 0 TCBs for socket *.REXEC (512)
*Perm *Autolog   514 = REXECD has 0 TCBs for socket *.RSH (514) *Perm *Autolog
 2049 = VMNFS has 0 TCBs for socket *.2049 *Perm *Autolog
```

Figure 33 (Part 3 of 3). A Sample of an INITIALIZE Trace Using MORETRACE

IPDOWN or IP-DOWN

The IPDOWN or IP-DOWN trace provides information about the IP_DOWN process and IP packets, including the link name and link type.

Figure 34 shows a sample of an IPDOWN trace.

```
Ipdown: Link: Link Name: TR1, Link Type: IBMTR, Dev Name: LCS1, Dev Type: LCS, Queuesize: 0
Ipdown: FirstHop 9.67.58.234
```

Figure 34. A Sample of an IPDOWN Trace

When you use the MORETRACE command, you receive information about the datagram such as the length, ID, protocol, TTL, addresses, and fragments. A sample of an IPDOWN trace using MORETRACE is shown in Figure 35.

```
IP-down: ShouldFragment: Datagram: 5046328 Packet size:0
     version: 0
     Internet Header Length: 5 = 20 bytes
     Type of Service:Precedence = Routine
     Total Length: 77 bytes
     Identification: 43
     Flags: May Fragment, Last Fragment
     Fragment Offset: 0
     Time To Live: 60
     Protocol: UDP
     Header CheckSum: 1443
     Source Address: 09433AE9
     Destination Address: 09432B64
```

Figure 35. A Sample of an IPDOWN Trace Using MORETRACE

IPUP or IP-UP

The IPUP or IP-UP trace provides the ID, length, protocol, and source address of incoming datagrams.

Figure 36 shows a sample of an IPUP trace.

```
   IP-up: datagram ID 52556, len 124, Protocol UDP from 9.67.43.100
DispatchDatagram: Dest 9.67.43.126, protocol 1 dispatch mode 1, PassedRoute T, DontRoute F
```

Figure 36. A Sample of an IPUP Trace

When you use the MORETRACE command, you receive additional information about the datagram, such as TTLs and fragments. A sample of an IPUP trace using MORETRACE is shown in Figure 37.

```
   IP-up examining:
      version: 0
      Internet Header Length: 5 = 20 bytes
      Type of Service:Precedence = Routine
      Total Length: 124 bytes
      Identification: 52670
      Flags: May Fragment, Last Fragment
      Fragment Offset: 0
      Time To Live: 28
      Protocol: UDP
      Header CheckSum: 22496
      Source Address: 09432B64
      Destination Address: 09433AE9
```

Figure 37. A Sample of an IPUP Trace Using MORETRACE

MONITOR

The MONITOR trace provides information about monitor requests, such as netstat, trace modifications, and drops, from authorized users.

A sample of a MONITOR trace using the MORETRACE command is shown in Figure 38. To receive more information from the details provided by MORETRACE, use the MONITORquery function. For more information about the MONITORquery function, see the *TCP/IP Programmer's Reference*.

```
Monitor cmd: UseNewFile returns
   OK
Monitor called:
   External interrupt handler->Monitor: Accept monitor request
 from TCPMAINT Monitor query

DoMonitorQuery called.
Mon Query: VMCF receive completed.
Mon Query: QueryRecord.QueryType = 12
Mon Query: reject/reply ret code is 0
    OK
DoMonitorQuery Ending!
Monitor called:
   External interrupt handler->Monitor: Accept monitor request
 from TCPMAINT Monitor query

DoMonitorQuery called.
Mon Query: VMCF receive completed.
Mon Query: QueryRecord.QueryType = 2
Mon Query: reject/reply ret code is 0
    OK
DoMonitorQuery Ending!
```

Figure 38 (Part 1 of 3). A Sample of a MONITOR Trace Using MORETRACE

```
Monitor called:
   External interrupt handler->Monitor: Accept monitor request
 from TCPMAINT Monitor query

DoMonitorQuery called.
Mon Query: VMCF receive completed.
Mon Query: QueryRecord.QueryType = 12
Mon Query: reject/reply ret code is 0
    OK
DoMonitorQuery Ending!
Monitor called:
   External interrupt handler->Monitor: Accept monitor request
 from TCPMAINT Monitor query

DoMonitorQuery called.
Mon Query: VMCF receive completed.
Mon Query: QueryRecord.QueryType = 14
Mon Query: reject/reply ret code is 0
    OK
DoMonitorQuery Ending!
Monitor called:
   External interrupt handler->Monitor: Accept monitor request
 from TCPMAINT Monitor query

DoMonitorQuery called.
Mon Query: VMCF receive completed.
Mon Query: QueryRecord.QueryType = 4
Mon Query: reject/reply ret code is 0
    OK
DoMonitorQuery Ending!
Monitor called:
   External interrupt handler->Monitor: Accept monitor request
 from TCPMAINT Monitor query

DoMonitorQuery called.
Mon Query: VMCF receive completed.
Mon Query: QueryRecord.QueryType = 12
Mon Query: reject/reply ret code is 0
    OK
DoMonitorQuery Ending!
Monitor called:
   External interrupt handler->Monitor: Accept monitor request
 from TCPMAINT Monitor query

DoMonitorQuery called.
Mon Query: VMCF receive completed.
Mon Query: QueryRecord.QueryType = 2
Mon Query: reject/reply ret code is 0
    OK
DoMonitorQuery Ending!
Monitor called:
   External interrupt handler->Monitor: Accept monitor request
 from TCPMAINT Monitor query
```

Figure 38 (Part 2 of 3). A Sample of a MONITOR Trace Using MORETRACE

```
DoMonitorQuery called.
Mon Query: VMCF receive completed.
Mon Query: QueryRecord.QueryType = 8
10:52:37 09/11/90 Monitor KILL TCB #1010 (INTCLIEN) Connection dropped by operator
    Bytes: 6469 sent, 13213 received
    Segs in: 110 OK, 35 pushed
    Max use: 1 in retransmit Q
Respond to TCPMAINT :
   OK
Monitor: SimpleResponse--SendMessage RetCode is
   OK
Monitor called:
   External interrupt handler->Monitor: Accept monitor request
 from TCPMAINT Monitor command

Monitor cmd: VMCF receive completed.
```

Figure 38 (Part 3 of 3). A Sample of a MONITOR Trace Using MORETRACE

NOPROCESS or NO-PROCESS or NONE

NOPROCESS or NO-PROCESS or NONE all suppress tracing. They are similar to the NOTRACE and LESSTRACE commands.

NOTIFY

NOTIFY traces the NOTIFY VMCF transactions between users and TCPIP. It provides information about MSGIG, CALLCODEs, ACB numbers, text of notices, return codes of VMCF transactions, and transaction parameters, such as LENA, LENB, VADA, and VADB. Figure 39 shows a sample of a NOTIFY trace.

```
Notify called for ACB 13719104:
   Send notice -> Notify (from UDP-request)
   Last touched: 70090
      Client: TCPMAINT
      Notice: UDP data delivered
   UDP data delivered is NOT valid.
Notify called for ACB 13719104:
   Send notice -> Notify (from IP-up)
   Last touched: 70090
      Client: NAMESRV
      Notice: UDP data delivered
   UDP data delivered is valid.
ProduceMessage: Message id = 111 CallCode = 16 ReturnCode = 0
NOTIFY: UDP INFO
LenA = 49
LenB = 234881024 VadB = 1039
AnInteger = 49
Connection = 4096
```

Figure 39 (Part 1 of 2). A Sample of a NOTIFY Trace

```
Notify called for ACB 13719104:
   Terminate notice -> Notify (from External interrupt handler)
   Last touched: 70090
      Client name: NAMESRV
      Message identifier:111
Notify called for ACB 13719104:
   Send notice -> Notify (from IP-up)
   Last touched: 70090
      Client: TCPMAINT
      Notice: UDP data delivered
   UDP data delivered is valid.
ProduceMessage: Message id = 113 CallCode = 16 ReturnCode = 0
NOTIFY: UDP INFO
LenA = 65
LenB = 234881024 VadB = 53
AnInteger = 65
Connection = 4096
Notify called for ACB 13719416:
   Send notice -> Notify (from UDP-request)
   Last touched: 70090
      Client: NAMESRV
      Notice: UDP data delivered
   UDP data delivered is NOT valid.
Notify called for ACB 13719416:
   Terminate notice -> Notify (from External interrupt handler)
   Last touched: 70090
      Client name: TCPMAINT
      Message identifier:113
Notify called for ACB 13719416:
   Send notice -> Notify (from PCCA3 common routine)
   Last touched: 70090
      Client: SNMPQE
      Notice: RawIp packets delivered
   RawIp packets delivered is valid.
Notify called for ACB 13718896:
   Send notice -> Notify (from PCCA3 common routine)
   Last touched: 70091
   Timeout: 73504.811 seconds
      Client: TCPMAINT
      Notice: Ping response or timeout
         PingTurnCode: OK
         Elapsed time: 0.109 seconds
   Ping response or timeout is valid.
ProduceMessage: Message id = 115 CallCode = 30 ReturnCode = 0
```

Figure 39 (Part 2 of 2). A Sample of a NOTIFY Trace

Figure 40 shows a sample of a NOTIFY trace using the MORETRACE command, which provides additional information about allocated buffers for users and the number of notices stacked.

```
Notify called for ACB 13720144:
   Send notice -> Notify (from PCCA3 common routine)
   Last touched: 70215
   Timeout: 73349.117 seconds
      Client: SNMPQE
      Notice: RawIp packets delivered
   Notify allocates buffer #0
   FindAndSendNotice(SNMPQE) finds 1 notices queued
   RawIp packets delivered is valid.
   WrapUp(SNMPQE): 13719728:
      Send notice -> Notify (from PCCA3 common routine)
      Last touched: 70215
      Timeout: 73349.117 seconds
         Client: SNMPQE
         Notice: RawIp packets delivered
   WrapUp frees buffer #0
Notify called for ACB 13719520:
   Send notice -> Notify (from PCCA3 common routine)
   Last touched: 70215
   Timeout: 73635.007 seconds
      Client: TCPMAINT
      Notice: Ping response or timeout
         PingTurnCode: OK
         Elapsed time: 0.110 seconds
   Notify allocates buffer #0
   FindAndSendNotice(TCPMAINT) finds 1 notices queued
   Ping response or timeout is valid.
ProduceMessage: Message id = 121 CallCode = 30 ReturnCode = 0
   Send ExternalBuffer 0 to TCPMAINT
Notify called for ACB 13719520:
   Terminate notice -> Notify (from External interrupt handler)
   Last touched: 70215
   Timeout: 73635.007 seconds
      Client name: TCPMAINT
      Message identifier:121
   WrapUp(TCPMAINT): 13720144:
      Send notice -> Notify (from PCCA3 common routine)
      Last touched: 70215
      Timeout: 73635.007 seconds
         Client: TCPMAINT
         Notice: Ping response or timeout
            PingTurnCode: OK
            Elapsed time: 0.110 seconds
   WrapUp frees buffer #0
```

Figure 40. A Sample of a NOTIFY Trace Using MORETRACE

PARSE-TCP

The PARSE-TCP trace provides information about the options and statements parsed during TCPIP initialization or after reading an OBEYFILE containing information about home links. PARSE-TCP produces the TCP/IP configuration if it is specified in the TRACE statement of the TCPIP PROFILE. This trace is helpful when running many test cases, because it can suggest the traces that should be executed.

Figure 41 shows a sample of the console log after executing an OBEYFILE command. The OBEYFILE contained the following commands:

```
TRACE parse-tcp
MORETRACE tcp
LESSTRACE tcp-request.
```

Note: MORETRACE activates TCP traces on both the TRACE and DETAILEDTRACE statements in Figure 41. For more information on TCP group processes, see "TCP" on page 116. TCPREQUEST is not listed in the DETAILEDTRACE statement in Figure 41, because the LESSTRACE command in the OBEYFILE excludes TCP-request.

```
   All tracing goes to screen
   Trace: TCP congestion control, Notify, Parse-Tcp, Retransmit-datagram,
   Roundtrip, TCP-down, TCP-request, TCP-up
   DetailedTrace: TCP congestion control, Notify, Retransmit-datagram,
   Roundtrip, TCP-down, TCP-up
BSD info for links:
ETH1: BrdAddr 9.67.58.63, DstAddr *, MaxMtu 0, Metric 0, SubnetMask 255.255.255.224
TR1: BrdAddr 9.67.58.255, DstAddr *, MaxMtu 0, Metric 0, SubnetMask 255.255.255.224
TR2: BrdAddr 9.67.58.223, DstAddr *, MaxMtu 0, Metric 0, SubnetMask 255.255.255.224
```

Figure 41. A Sample of a PARSE-TCP Trace Using MORETRACE and LESSTRACE

PING

The PING trace provides information about outgoing PING requests from home clients, ICMP datagrams, and associated data. It is helpful to match ICMP datagram data with CCW traces.

Figure 42 shows a sample of a PING trace.

```
Ping called:
13714800:
   Accept ping request -> Ping process (from External interrupt handler)
   Last touched: 23
   Timeout: 203.493 seconds
      Client name: TCPMAINT
      Address: 9.67.43.126
      Length: 256
      Timeout: 10

DoPing sending datagram:
   version: 0
   Internet Header Length: 5 = 20 bytes
   Type of Service:Precedence = Routine
   Total Length: 276 bytes
   Identification: 1234
   Flags: May Fragment, Last Fragment
   Fragment Offset: 0
   Time To Live: 60
   Protocol: ICMP
   Header CheckSum: 43
   Source Address: 09433AE9
   Destination Address: 09432B7E
```

Figure 42 (Part 1 of 2). A Sample of a PING Trace

```
   Data:
08 00 23 43 00 D1 46 A8 47 83 D5 AB 53 8D 8B 5B
7F D6 A3 7F 8D 5B 7B ED 22 72 5C 92 64 42 3E 79
18 27 2F ED 6B B9 68 04 B1 04 66 C5 27 80 03 9D
78 BB 4F 97 53 A2 0A 52 39 85 D4 A9 5D 53 DA B8
02 6D 9D 11 28 2B 06 E1 DE 16 C9 5F 2B CC 3A 08
C6 7E 72 00 BB C8 C0 E4 11 E3 C5 A8 76 C2 2A 6D
72 13 47 6F 4D F0 3E C9 34 29 02 F9 4E 5C B8 80
74 F3 01 33 FA 1C 8B CB D9 45 B7 9B D3 9B B3 5A
5D A1 06 68 B3 8F 20 E0 CC 82 50 C8 2B 63 AC BD
0D 21 5A EE 3B DB C9 96 DB 6F B5 7B 91 48 EC 56
39 82 E8 37 FB 0E DF E4 F3 91 D1 AF 3C 13 7D 29
B8 AF 57 73 23 E8 97 B6 4E A2 12 1D 6B 8B 7F A5
CF A9 64 2B C5 62 1D 1D 62 C2 3B 0A B5 E0 35 12
8D C9 E3 0B 09 EB 9E 8E 3C 37 A5 16 07 F0 83 29
B6 BC 09 3A C8 40 E1 A1 84 73 F5 F5 73 86 97 1E
E1 C2 BA 0B 30 05 E2 D9 33 21 36 C5 53 75 19 23

UpToPing processing datagram:
   version: 0
   Internet Header Length: 5 = 20 bytes
   Type of Service:Precedence = Routine
   Total Length: 276 bytes
   Identification: 1234
   Flags: May Fragment, Last Fragment
   Fragment Offset: 0
   Time To Live: 58
   Protocol: ICMP
   Header CheckSum: 555
   Source Address: 09432B7E
   Destination Address: 09433AE9
   Data:
00 00 2B 43 00 D1 46 A8 47 83 D5 AB 53 8D 8B 5B
7F D6 A3 7F 8D 5B 7B ED 22 72 5C 92 64 42 3E 79
18 27 2F ED 6B B9 68 04 B1 04 66 C5 27 80 03 9D
78 BB 4F 97 53 A2 0A 52 39 85 D4 A9 5D 53 DA B8
02 6D 9D 11 28 2B 06 E1 DE 16 C9 5F 2B CC 3A 08
C6 7E 72 00 BB C8 C0 E4 11 E3 C5 A8 76 C2 2A 6D
72 13 47 6F 4D F0 3E C9 34 29 02 F9 4E 5C B8 80
74 F3 01 33 FA 1C 8B CB D9 45 B7 9B D3 9B B3 5A
5D A1 06 68 B3 8F 20 E0 CC 82 50 C8 2B 63 AC BD
0D 21 5A EE 3B DB C9 96 DB 6F B5 7B 91 48 EC 56
39 82 E8 37 FB 0E DF E4 F3 91 D1 AF 3C 13 7D 29
B8 AF 57 73 23 E8 97 B6 4E A2 12 1D 6B 8B 7F A5
CF A9 64 2B C5 62 1D 1D 62 C2 3B 0A B5 E0 35 12
8D C9 E3 0B 09 EB 9E 8E 3C 37 A5 16 07 F0 83 29
B6 BC 09 3A C8 40 E1 A1 84 73 F5 F5 73 86 97 1E
E1 C2 BA 0B 30 05 E2 D9 33 21 36 C5 53 75 19 23

UpToPing: Ping was requested by TCPMAINT
UpToPing: Ping took 0.314 seconds
```

Figure 42 (Part 2 of 2). A Sample of a PING Trace

ROUNDTRIP or ROUND-TRIP

The ROUNDTRIP or ROUND-TRIP trace shows the average round-trip time.

Figure 43 shows a sample of the ROUNDTRIP trace.

```
RecordSend: Timeout interval is 300 timer units
   Ack #1 took   0.043; # acked: 1, ave RT:   0.043
   Avg time in burst:  0.043, err   0.000 => smooth RT:  0.043, smooth var:   0.022
RecordSend: Timeout interval is 75 timer units
   Ack #4 took   0.075; # acked: 2, ave RT:   0.059
   Avg time in burst:  0.075, err   0.032 => smooth RT:  0.047, smooth var:   0.024
RecordSend: Timeout interval is 75 timer units
   Ack #22 took   0.040; # acked: 3, ave RT:   0.053
   Avg time in burst:  0.040, err   0.007 => smooth RT:  0.046, smooth var:   0.020
RecordSend: Timeout interval is 75 timer units
   Ack #25 took   0.041; # acked: 4, ave RT:   0.050
   Avg time in burst:  0.041, err   0.005 => smooth RT:  0.045, smooth var:   0.016
RecordSend: Timeout interval is 75 timer units
   Ack #31 took   0.058; # acked: 5, ave RT:   0.051
   Avg time in burst:  0.058, err   0.013 => smooth RT:  0.047, smooth var:   0.015
RecordSend: Timeout interval is 75 timer units
   Ack #34 took   0.049; # acked: 6, ave RT:   0.051
   Avg time in burst:  0.049, err   0.002 => smooth RT:  0.047, smooth var:   0.012
```

Figure 43. A Sample of a ROUNDTRIP Trace

SCHEDULER

The SCHEDULER trace shows the next main process to be executed. Because scheduler trace entries contain a time stamp, it is often helpful to include TRACE SCHEDULER when diagnosing other problems so that events can be placed in time.

Figure 44 shows a sample of a SCHEDULER trace.

```
Scheduler: 2312233908 Accept TCP request -> TCP-request
Scheduler: 2312801249 Accept TCP request -> TCP-request
Scheduler: 2312801447 Accept TCP request -> TCP-request
Scheduler: 2312801649 Accept monitor request -> Monitor
Scheduler: 2312801997 Accept ping request -> Ping process
Scheduler: 2312802206 Examine incoming datagram -> IP-up
Scheduler: 2312802343 Examine incoming datagram -> IP-up
Scheduler: 2312802446 Send notice -> Notify
Scheduler: 2312802615 Terminate notice -> Notify
Scheduler: 2312802739 Accept TCP request -> TCP-request
Scheduler: 2313031379 Accept TCP request -> TCP-request
Scheduler: 2313031645 Accept monitor request -> Monitor
```

Figure 44. A Sample of a SCHEDULER Trace

Note: The number in each line of the SCHEDULER trace is a partial time stamp that shows in relative terms when each event occurred. The values are in 16-microsecond units.

Figure 45 shows a sample of a SCHEDULER trace using the MORETRACE command, which adds information about the ACB to be processed. This trace provides information, such as message identifiers, client calls, and details related to VMCF communication.

```
DASD 03EE LINKED R/O; R/W BY TCPMNTA
DMSACP723I Z (3EE) R/O
DASD 03EE DETACHED
DTCSCH004I Scheduler: 2339349463 Accept TCP request -> TCP-request
DTCPRI048I    32871464:
DTCPRI058I    Accept TCP request -> TCP-request (from External interrupt handler)
DTCPRI061I       Client name: TCPMNTA
DTCPRI062I       Message identifier:10
DTCPRI063I       Client call: End TCP/IP service
DTCSCH004I Scheduler: 2339442590 Look at Timer Queue -> Timer
DTCPRI048I    32871464:
DTCPRI058I    Look at Timer Queue -> Timer (from External interrupt handler)
DTCSCH004I Scheduler: 2339442967 Check consistency -> Consistency checker
DTCPRI048I    32871944:
DTCPRI058I    Check consistency -> Consistency checker (from Timer)
DTCSCH004I Scheduler: 2339443369 Terminate notice -> Notify
DTCPRI048I    32871944:
DTCPRI058I    Terminate notice -> Notify (from External interrupt handler)
DTCPRI098I       Client name: FTPSRVA
DTCPRI099I       Message identifier:-3
DTCPRI100I       Return code: Abnormal condition during inter-VM communication (VMCF Rc=0 User=FTPSRVA)
DTCSCH004I Scheduler: 2339449984 Look at Timer Queue -> Timer
DTCPRI048I    32871944:
DTCPRI058I    Look at Timer Queue -> Timer (from External interrupt handler)
DTCSCH004I Scheduler: 2339450329 Internal Telnet timeout -> Internal Telnet timeout handler
DTCPRI048I    32871584:
DTCPRI058I    Internal Telnet timeout -> Internal Telnet timeout handler (from Timer)
DTCPRI103I       Timer Datum: 16777216, Timer Number: 1
DTCSCH004I Scheduler: 2339450814 Internal Telnet notification -> Internal Telnet server
DTCPRI048I    32871944:
DTCPRI058I    Internal Telnet notification -> Internal Telnet server (from Internal Telnet timeout handler)
DTCPRI005I       Notification: Timer expired
DTCPRI015I          Datum: 16777216, Associated timer: 1
DTCSCH004I Scheduler: 2339521596 Accept TCP request -> TCP-request
DTCPRI048I    32871944:
DTCPRI058I    Accept TCP request -> TCP-request (from External interrupt handler)
DTCPRI061I       Client name: TCPMNTA
DTCPRI062I       Message identifier:6
DTCPRI063I       Client call: Begin TCP/IP service
DTCSCH004I Scheduler: 2339522504 Accept TCP request -> TCP-request
DTCPRI048I    32871944:
DTCPRI058I    Accept TCP request -> TCP-request (from External interrupt handler)
DTCPRI061I       Client name: TCPMNTA
DTCPRI062I       Message identifier:8
DTCPRI063I       Client call: Handle notice
DTCPRC104I       Notices: Buffer space available, Connection state changed
, Data delivered, User-defined notification, Datagram space available
, Urgent pending, UDP data delivered, UDP datagram space available
, Other external interrupt received, User delivers line
, User wants attention, Timer expired, FSend response, FReceive error
, RawIp packets delivered, RawIp packet space available, IUCV interrupt
, I/O interrupt, Resources available for TcpOpen
, Resources available for UdpOpen, Ping response or timeout, SMSG received
DTCSCH004I Scheduler: 2339523820 Accept monitor request -> Monitor
DTCPRI048I    32871944:
DTCPRI058I    Accept monitor request -> Monitor (from External interrupt handler)
DTCPRI061I       Client name: TCPMNTA
DTCPRI062I       Message identifier:10
DTCPRI063I       Client call: Monitor query
DTCSCH004I Scheduler: 2339524493 Accept ping request -> Ping process
DTCPRI048I    32871944:
DTCPRI058I    Accept ping request -> Ping process (from External interrupt handler)
DTCPRI070I       Client name: TCPMNTA
DTCPRI071I       Address: 9.130.3.2
DTCPRI072I       Length: 256
DTCPRI073I       Timeout: 10
DTCSCH004I Scheduler: 2339525028 Examine incoming datagram -> IP-up
DTCPRI048I    32871824:
DTCPRI058I    Examine incoming datagram -> IP-up (from Ping process)
DTCPRI280I    Timeout: 64.829 seconds
DTCSCH004I Scheduler: 2339525285 Examine incoming datagram -> IP-up
DTCPRI048I    32871944:
DTCPRI058I    Examine incoming datagram -> IP-up (from IP-up)
DTCSCH004I Scheduler: 2339525450 Send notice -> Notify
DTCPRI048I    32871704:
```

Figure 45. A Sample of a SCHEDULER Trace Using MORETRACE

```
DTCPRI058I    Send notice -> Notify (from IP-up)
DTCPRI280I    Timeout: 492.394 seconds
DTCPRI081I       Client: TCPMNTA
DTCPRI084I       Notice: Ping response or timeout
DTCPRI092I          PingTurnCode: OK
DTCPRI093I          Elapsed time: 0.004 seconds
DTCSCH004I Scheduler: 2339528415 Terminate notice -> Notify
DTCPRI048I    32871704:
DTCPRI058I Terminate notice -> Notify (from External interrupt handler)
DTCPRI280I    Timeout: 492.394 seconds
DTCPRI098I       Client name: TCPMNTA
DTCPRI099I       Message identifier:5
DTCSCH004I Scheduler: 2339529294 Accept TCP request -> TCP-request
DTCPRI048I    32871824:
DTCPRI058I    Accept TCP request -> TCP-request (from External interrupt handler)
DTCPRI280I    Timeout: 64.829 seconds
DTCPRI061I       Client name: TCPMNTA
DTCPRI062I       Message identifier:14
DTCPRI063I       Client call: End TCP/IP service
DTCSCH004I Scheduler: 2339670616 Accept TCP request -> TCP-request
DTCPRI048I    32871824:
DTCPRI058I    Accept TCP request -> TCP-request (from External interrupt handler)
DTCPRI280I    Timeout: 64.829 seconds
DTCPRI061I       Client name: TCPMNTA
DTCPRI062I       Message identifier:6
DTCPRI063I       Client call: Begin TCP/IP service
DTCSCH004I Scheduler: 2339671667 Accept monitor request -> Monitor
DTCPRI048I    32871824:
DTCPRI058I    Accept monitor request -> Monitor (from External interrupt handler)
DTCPRI280I    Timeout: 64.829 seconds
DTCPRI061I       Client name: TCPMNTA
DTCPRI062I       Message identifier:8
DTCPRI063I       Client call: Monitor command
DASD 03EE LINKED R/O; R/W BY TCPMNTA
DMSACP723I Z (3EE) R/O
DASD 03EE DETACHED
```

Figure 46. continuation of the SCHEDULER Trace

SHUTDOWN or SHUT-DOWN

The SHUTDOWN or SHUT-DOWN trace provides information about clients and servers, TCPIP shut down, and the status of pending communication between clients and TCPIP.

Figure 47 shows a sample of a SHUTDOWN trace.

```
11:01:57 09/07/90 Shutdown KILL TCB #1001 (FTPSERVE) TCP/IP service is being shut down
    Bytes: 0 sent, 0 received
    Max use: 0 in retransmit Q
11:01:57 09/07/90 Shutdown KILL TCB #1003 (SMTP    ) TCP/IP service is being shut down
    Bytes: 0 sent, 0 received
    Max use: 0 in retransmit Q
11:01:57 09/07/90 Shutdown KILL TCB #1007 (NAMESRV ) TCP/IP service is being shut down
    Bytes: 0 sent, 0 received
    Max use: 0 in retransmit Q
```

Figure 47 (Part 1 of 2). A Sample of a SHUTDOWN Trace

```
11:01:57 09/07/90 Shutdown KILL TCB #1000 (INTCLIEN) TCP/IP service is being shut down
    Bytes: 0 sent, 0 received
    Max use: 0 in retransmit Q
11:01:57 09/07/90 Shutdown KILL TCB #1008 (SNMP    ) You aborted the connection
    Bytes: 0 sent, 0 received
    Max use: 0 in retransmit Q
11:01:57 09/07/90 Shutdown KILL TCB #1002 (PORTMAP ) You aborted the connection
    Bytes: 0 sent, 0 received
    Max use: 0 in retransmit Q
11:01:57 09/07/90 Shutdown KILL TCB #1006 (SNMPQE  ) You aborted the connection
    Bytes: 0 sent, 0 received
    Max use: 0 in retransmit Q

7 active clients, with 4 connections in use.
I will delay shutting down for 30 seconds, so that
RSTs and shutdown notifications may be delivered.
If you wish to shutdown immediately, without warning,
type #CP EXT again.

Server Telnet closed down. Bye.
PCCA3 shutting down:
   Device LCS1:
      Type: LCS, Status: Ready
      Envelope queue size: 0
      Address: 0560
UnlockAll issuing "CP UNLOCK TCPIP 0 DFF"
COMMAND COMPLETE
ShutDown at 75442.687 seconds
```

Figure 47 (Part 2 of 2). A Sample of a SHUTDOWN Trace

SNMPDPI

The SNMPDPI trace provides SNMP “sub-agent” tracing. It lists the MIB queries by the SNMP agent.

Figure 48 shows a sample of an SNMPDPI trace.

```
SNMP DPI process called for ACB 13657768:
   Process SNMP agent request -> SNMP DPI sub-agent (from Sock-request)
      SnmpAgentCcb SNMPD, SnmpAgentSockNumber 7
ProcessMibRequest: Cmd 2, ObjectId 1.3.6.1.2.1.2.2.1.2.1.,
      GroupId 1.3.6.1.2.1.2.2.1.2..
ProcessMibRequest: Name ifDescr, EffectiveCmd 2,
      EffectiveObjectId 1.3.6.1.2.1.2.2.1.2.1., Instance 1
mkDPIresponse: ret_code 0
object_id 1.3.6.1.2.1.2.2.1.2.2, set_type 2, value_len 13
D80638:49424D20 4E505349 20582E32 35000000
SNMP DPI process called for ACB 13657456:
   Process SNMP agent request -> SNMP DPI sub-agent (from Sock-request)
   Timeout: 209.996 seconds
      SnmpAgentCcb SNMPD, SnmpAgentSockNumber 7
ProcessMibRequest: Cmd 1, ObjectId 1.3.6.1.2.1.2.2.1.2.7.,
      GroupId 1.3.6.1.2.1.2.2.1.2.7.
ProcessMibRequest: Name ifDescr, EffectiveCmd 1,
      EffectiveObjectId 1.3.6.1.2.1.2.2.1.2.7., Instance 7
mkDPIresponse: ret_code 2
```

Figure 48. A Sample of an SNMPDPI Trace

SOCKET

The SOCKET trace provides information about the requests made through the IUCV socket interface, as well as most responses.

Figure 49 shows a sample of a SOCKET trace.

```
 .
 .
 .
SkSimpleResponse: Client USER8    06319a70, retcode 0 errno 49
Sock-request called for ACB TCPPRI048I 106078608:
DTCPRI052I    IUCV interrupt -> Sock-request (from External interrupt handler)
DTCPRI038I    Interrupt type: Pending message
DTCPRI039I       Path id: 3
      MsgId 666, Length 16, TrgCls: 00190003, Reply len 8, Flags 07
SkSimpleResponse: Client USER8    06319a70, retcode 3 errno 0
Sock-request called for ACB TCPPRI048I 106078608:
DTCPRI052I    IUCV interrupt -> Sock-request (from External interrupt handler)
DTCPRI038I    Interrupt type: Pending message
DTCPRI039I       Path id: 3
      MsgId 667, Length 16, TrgCls: 00020003, Reply len 8, Flags 07
SkSimpleResponse: Client USER8    06319a70, retcode 0 errno 0
Sock-request called for ACB TCPPRI048I 106078608:
DTCPRI052I    IUCV interrupt -> Sock-request (from External interrupt handler)
DTCPRI038I    Interrupt type: Pending message
DTCPRI039I       Path id: 3
      MsgId 668, Length 0, TrgCls: 000D0003, Reply len 8, Flags 87
      PrmMsgHi 0, PrmMsgLo 5
SkSimpleResponse: Client USER8    06319a70, retcode 0 errno 0
Sock-request called for ACB TCPPRI048I 106078608:
DTCPRI052I    IUCV interrupt -> Sock-request (from External interrupt handler)
DTCPRI038I    Interrupt type: Pending message
DTCPRI039I       Path id: 3
      MsgId 669, Length 16, TrgCls: 00190004, Reply len 8, Flags 07
SkSimpleResponse: Client USER8    06319a70, retcode 4 errno 0
Sock-request called for ACB TCPPRI048I 106078608:
DTCPRI052I    IUCV interrupt -> Sock-request (from External interrupt handler)
DTCPRI038I    Interrupt type: Pending message
DTCPRI039I       Path id: 3
      MsgId 670, Length 16, TrgCls: 00020004, Reply len 8, Flags 07
SkSimpleResponse: Client USER8    06319a70, retcode 0 errno 0
Sock-request called for ACB TCPPRI048I 106078608:
DTCPRI052I    IUCV interrupt -> Sock-request (from External interrupt handler)
DTCPRI038I    Interrupt type: Pending message
DTCPRI039I       Path id: 3
      MsgId 671, Length 52, TrgCls: 00130008, Reply len 40, Flags 07
SkBlockRequest: Pathid 3, Msgid 671, Retryable F
 .
 .
 .
```

Figure 49. A Sample of a SOCKET Trace

TCPDOWN or TCP-DOWN

The TCPDOWN or TCP-DOWN trace provides information about the outgoing TCP datagrams, such as data byte length, source port, destination port, and the connection to which the call is related. TCPDOWN also provides some information about the other fields in outgoing datagrams, such as:

- Sequence (seq) number
- Acknowledgment (ack) number
- Segment size.

Figure 50 shows a sample of a TCPDOWN trace in which A or AP control bits are posted (Ack and PUSH).

```
TCP-down called for ACB 13716048:
   ACK timeout fails #1007 -> TCP-down (from Timer)
   Last touched: 2782
   TCP-down constructing datagram with 0 bytes of text
   ConstructGram sending header: Port 1037->23: #626673280 Ack=639844686 Wnd=65527 A
TCP-down called for ACB 13715736:
   Send TCP data #1007 -> TCP-down (from TCP-request)
   Last touched: 2783
   Timeout: 2947.615 seconds
TCP-down: desired segment size = 18 -> PUSH
   TCP-down finds ready segment size = 18
   TCP-down constructing datagram with 18 bytes of text
   ConstructGram sending header: Port 1037->23: #626673280 Ack=639844686 Wnd=65527 AP
   TCP-down has sent out 18 bytes data; SegLen 18 ; SndNxt 22, ClientSndNxt = 22
```

Figure 50. A Sample of a TCPDOWN Trace

When you activate a TCPDOWN trace using the MORETRACE command, the foreign host IP address is given and the format of the output is easier to read.

Figure 51 shows a sample of a TCPDOWN trace using the MORETRACE command.

```
   MakeHead in TCP-down: SourcePort is 1038
                     DestinationPort is TELNET (23)
                     ConnectionName is 1007
   TCP-down making header seq #650306676
   TCP-down: window size: 32768
   GuessSegSize(9.67.43.126) => 0.0.0.0 -> 9.67.58.234 Link Name: TR1,
    Link Type: IBMTR, Dev Name: LCS1, Dev Type: LCS, max: 0
   TCP-down sending max seg size = 536
   TCP-down constructing datagram with 0 bytes of text
   ConstructGram sending header:
      Source Port: 1038
      Destination Port: 23
      Sequence Number: 650306676
      Data Offset: 6
      Control Bits: SYN
      Window: 32768
```

Figure 51 (Part 1 of 2). A Sample of a TCPDOWN Trace Using MORETRACE

```
      Checksum: 15721
      Options:
         Maximum segment size: 536
   GuessSegSize(9.67.43.126) => 0.0.0.0 -> 9.67.58.234 Link Name: TR1,
    Link Type: IBMTR, Dev Name: LCS1, Dev Type: LCS, max: 0
TCP-down called for ACB 13715632:
   ACK timeout fails #1007 -> TCP-down (from Timer)
   Last touched: 2872
   Timeout: 3015.271 seconds
   MakeHead in TCP-down: SourcePort is 1038
                    DestinationPort is TELNET (23)
                    ConnectionName is 1007
   TCP-down making header seq #650306677
   TCP-down acking #666910577
   TCP-down: window size: 32768
   TCP-down constructing datagram with 0 bytes of text
   ConstructGram sending header:
      Source Port: 1038
      Destination Port: 23
      Sequence Number: 650306677
      Acknowledgement Number: 666910577
      Data Offset: 5
      Control Bits: ACK
      Window: 32768
      Checksum: 31536
TCP-down called for ACB 13715736:
   Send TCP data #1007 -> TCP-down (from TCP-request)
   Last touched: 2873
   Timeout: 3042.149 seconds
TCP-down: desired segment size = 3 -> PUSH
   TCP-down finds ready segment size = 3
   MakeHead in TCP-down: SourcePort is 1038
                    DestinationPort is TELNET (23)
                    ConnectionName is 1007
   TCP-down making header seq #650306677
   TCP-down acking #666910577
   TCP-down: window size: 32768
   TCP-down constructing datagram with 3 bytes of text
   TCP-down: CopyAllText takes 3 bytes from a buffer
   ConstructGram sending header:
      Source Port: 1038
      Destination Port: 23
      Sequence Number: 650306677
      Acknowledgement Number: 666910577
      Data Offset: 5
      Control Bits: ACK PSH
      Window: 32768
      Checksum: 57145
   TCP-down has sent out 3 bytes data; SegLen 3 ; SndNxt 4, ClientSndNxt = 4
```

Figure 51 (Part 2 of 2). A Sample of a TCPDOWN Trace Using MORETRACE

TCPUP or TCP-UP

The TCPUP or TCP-UP trace provides information about incoming TCP datagrams, such as the connection number, local destination port, sequence number, acknowledgment number, and window size.

Figure 52 shows a sample of a TCPUP trace.

```
   TCP-up's next segment: Port 1073->23: #568559375 Ack=500632569 Wnd=15652 A
   Valid TCP checksum
   #1006 Established I=1 O=1H1
W57921 RNxt=275 CliRNxt=275 SNxt=42269 SUna=42025 SWnd=15896 MaxSWnd=16384 CWnd=
33641 Thresh=5912 Con Re Pen2048
      Acceptable segment
 * #1006 Established I=1 RNxt=275 CliRNxt=275 SNxt=42269 SUna=42269 SWnd=15652 M
axSWnd=16384 CWnd=33755 Thresh=5912 Pen2048
   TCP-up's next segment: Port 1071->23: #495605235 Ack=323725624 Wnd=14676 A
   Valid TCP checksum
   #1000 Established I=1 O=1H1
W114300 RNxt=235 CliRNxt=235 SNxt=99624 SUna=99380 SWnd=14920 MaxSWnd=16384 CWnd
=1960 Thresh=7460 Con Re Pen2048
      Acceptable segment
 * #1000 Established I=1 RNxt=235 CliRNxt=235 SNxt=99624 SUna=99624 SWnd=14676 M
axSWnd=16384 CWnd=1960 Thresh=7460 Pen2048
   TCP-up's next segment: Port 1072->23: #536754847 Ack=469782320 Wnd=15652 A
   Valid TCP checksum
   #1007 Established I=1 O=1H1
W83772 RNxt=247 CliRNxt=247 SNxt=68120 SUna=67876 SWnd=15896 MaxSWnd=16384 CWnd=
38023 Thresh=7216 Con Re Pen2048
      Acceptable segment
 * #1007 Established I=1 RNxt=247 CliRNxt=247 SNxt=68120 SUna=68120 SWnd=15652 M
axSWnd=16384 CWnd=38124 Thresh=7216 Pen2048
```

Figure 52. A Sample of a TCPUP Trace

Figure 53 shows a sample of a TCPUP trace using the MORETRACE command, which provides complete information about each incoming TCP datagram, except the data.

```
   Next TCP header:
      Source Port:1073
      Destination Port: 23
      Sequence Number: 568559378
      Acknowledgement Number: 500650786
      Data Offset: 5
      Control bits: ACK
      Window: 15284
      Checksum: 2161
   Client text starts at 21
   Valid TCP checksum
```

Figure 53 (Part 1 of 4). A Sample of a TCPUP Trace Using MORETRACE

```
5240128:
  PrevTcb: 5241080
  NextTcb: 12153680
  Backoff count 0
  Client: INTCLIEN
  Last state notice: Open
  ClientRcvNxt: 568559378
  ClientSndNxt: 500650786
  CongestionWindow: 23488, SlowStartThreshold: 8070
  Local connection name: 1006
  ConnectionTimeoutTime in 150 seconds
  Foreign socket: net address = 9.67.58.225, port= 1073
  Sender frustration level: Contented
  Incoming segment queue: Queue size = 1
     5940600:
     PrevDataBuffer: 5241032
     NextDataBuffer: 5241032
     First Unused Sequence Number: 568559378
     Offset of last byte delivered: 0
     Offset of last byte received: 0
     Sequence number of first byte: 568559378

  Incoming window number: 568561149
  Initial receive sequence number: 568559100
  Initial send sequence number: 500590300
  Maximum segment size: 1960
  Local socket: net address = 9.67.58.233, port= TELNET (23)
  Outgoing segment queue: Queue size = 1
     5944840:
     PrevDataBuffer: 5241056
     NextDataBuffer: 5241056
     First Unused Sequence Number: 500650786
     Offset of last byte delivered: 0
     Offset of last byte received: 220
     Sequence number of first byte: 500650566

  Outgoing window number: 500666070
  Precedence: Routine
  RcvNxt: 568559378
  Round-trip information:
     How many in use: 1
     First free: 14
     First used: 13
     Max number unacked: 1
     Retransmission timeout: 1181.832 seconds
     Smooth trip time: 0.049
     Smooth variance: 0.032
     Total acked: 252
     Average trip time: 0.185
     Acks not counted in round-trip time: 3
     ReplaceSmooth FALSE
  SndNxt: 500650786
  SndUna: 500650566
  SndWl1: 568559378
  SndWl2: 500650566
  SndWnd: 15504
  MaxSndWnd: 16384
  State: Established
  Pending TCP-receive buffer: 2048
   WorkOn called:
```

Figure 53 (Part 2 of 4). A Sample of a TCPUP Trace Using MORETRACE

```
ClientTextStart = 21
ForeignAddress = 9.67.58.225
ForeignPort = 1073
LocalAddress = 9.67.58.233
LocalPort = TELNET (23)
SegPrc = Routine
SegLen = 0
TextLength = 0
TCB = 5240128:
   PrevTcb: 5241080
   NextTcb: 12153680
   Backoff count 0
   Client: INTCLIEN
   Last state notice: Open
   ClientRcvNxt: 568559378
   ClientSndNxt: 500650786
   CongestionWindow: 23488, SlowStartThreshold: 8070
   Local connection name: 1006
   ConnectionTimeoutTime in 145 seconds
   Foreign socket: net address = 9.67.58.225, port= 1073
   Sender frustration level: Contented
   Incoming segment queue: Queue size = 1
      5940600:
      PrevDataBuffer: 5241032
      NextDataBuffer: 5241032
      First Unused Sequence Number: 568559378
      Offset of last byte delivered: 0
      Offset of last byte received: 0
      Sequence number of first byte: 568559378

   Incoming window number: 568561149
   Initial receive sequence number: 568559100
   Initial send sequence number: 500590300
   Maximum segment size: 1960
   Local socket: net address = 9.67.58.233, port= TELNET (23)
   Outgoing segment queue: Queue size = 1
      5944840:
      PrevDataBuffer: 5241056
      NextDataBuffer: 5241056
      First Unused Sequence Number: 500650786
      Offset of last byte delivered: 0
      Offset of last byte received: 220
      Sequence number of first byte: 500650566

   Outgoing window number: 500666070
   Precedence: Routine
   RcvNxt: 568559378
   Round-trip information:
      How many in use: 1
      First free: 14
      First used: 13
      Max number unacked: 1
      Retransmission timeout: 1181.832 seconds
      Smooth trip time: 0.049
      Smooth variance: 0.032
      Total acked: 252
      Average trip time: 0.185
      Acks not counted in round-trip time: 3
      ReplaceSmooth FALSE
   SndNxt: 500650786
```

Figure 53 (Part 3 of 4). A Sample of a TCPUP Trace Using MORETRACE

```
         SndUna: 500650566
         SndWl1: 568559378
         SndWl2: 500650566
         SndWnd: 15504
         MaxSndWnd: 16384
         State: Established
         Pending TCP-receive buffer: 2048
      Acceptable segment
      SND.UNA = 60486
      Old: SndWnd = 15504, Wl1 = 278, Wl2 = 60266
      New: SndWnd = 15284, Wl1 = 278, Wl2 = 60486
      Finished with DataBuffer ending at 60486
 * #1006 Established I=1 RNxt=278 CliRNxt=278 SNxt=60486 SUna=60486 SWnd=15284 M
axSWnd=16384 CWnd=23651 Thresh=8070 Pen2048
   Next TCP header:
      Source Port: 1073
      Destination Port: 23
      Sequence Number: 568559378
      Acknowledgement Number: 500651006
      Data Offset: 5
      Control Bits: ACK
      Window: 15064
      Checksum: 2161
   Client text starts at 21
   Valid TCP checksum
```

Figure 53 (Part 4 of 4). A Sample of a TCPUP Trace Using MORETRACE

TCPREQUEST or TCP-REQUEST

The TCPREQUEST or TCP-REQUEST trace provides information about all TCP service requests from local clients and servers. TCP services are requested by the standard procedure. For more information about the standard request procedure, see the *TCP/IP Programmer's Reference*. TCPREQUEST traces can be matched with client traces, such as FTP traces.

The information contained in a TCPREQUEST trace includes:

- Client name: User ID of the requester
- Message identifier
- Client call (VMCF function only)
- Connection number
- Length
- Handle notices requests, if applicable.

The connection number is the TCP/IP connection number shown by NETSTAT in client traces. This number is computed to match TCP/IP clients with VMCF connections.

Figure 54 shows a sample of a TCPREQUEST trace. In this sample trace, the length equals 65535. A port value of 65535 is an X'FFFF' UNSPECIFIEDport. If a port is specified on a foreign socket, the UNSPECIFIEDaddress (X'00000000') and UNSPECIFIEDport means that the client or server is on a passive open port. However, local ports and addresses are specified.

```
TCP-request called for ACB 13715112:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1259
      Client name: TCPUSRX
      Message identifier:10
      Client call: End TCP/IP service

TCP-request KILLING CLIENT: TCPUSRX Client has ended TCP/IP service
TCP-request called for ACB 13715112:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1275
      Client name: TCPUSRX
      Message identifier:6
      Client call: Begin TCP/IP service

TCP-request KILLING CLIENT: TCPUSRX Client reinitialized TCP/IP service
TCP-request called for ACB 13715840:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1288
      Client name: TCPUSRX
      Message identifier:12
      Client call: Handle notice
      Notices: Buffer space available, Connection state changed, Data delivered,
 UDP data delivered, Timer expired, FSend response, FReceive error, IUCV interrupt
TCP-request called for ACB 13714800:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1288
   Timeout: 1190.212 seconds
      Client name: TCPUSRX
      Message identifier:24
      Client call: Open TCP
TcpRequest FindTcb: OurClientOwnsPort: FALSE, OtherClientOwnsPort: FALSE
TCP-request called for ACB 13715840:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1288
   Timeout: 1411.224 seconds
      Client name: TCPUSRX
      Message identifier:26
      Client call: FReceive TCP
      Connection number: 1009
      Length: 65535
TCP-request called for ACB 13715840:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1295
   Timeout: 1411.224 seconds
      Client name: TCPUSRX
      Message identifier:28
      Client call: FSend TCP
      Connection number: 1009
      Length: 14
TCP-request called for ACB 13715840:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1295
   Timeout: 1411.224 seconds
      Client name: TCPUSRX
      Message identifier:30
      Client call: FReceive TCP
      Connection number: 1009
      Length: 65535
```

Figure 54. A Sample of a TCPREQUEST Trace

The TCPREQUEST trace using the MORETRACE command adds the following information:

- Foreign and local IP addresses on active open ports
- Status of the open client port on passive open ports
- Parameters of established connections.

Figure 55 shows a sample of the TCPREQUEST trace using MORETRACE.

```
TCP-request called for ACB 13715632:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1377
   Timeout: 1347.787 seconds
      Client name: TCPUSRX
      Message identifier:22
      Client call: Handle notice
      Notices: Buffer space available, Connection state changed, Data delivered,
 FSend response, FReceive error, IUCV interrupt
TCP-request called for ACB 13715632:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1377
   Timeout: 1347.787 seconds
      Client name: TCPUSRX
      Message identifier:24
      Client call: Open TCP
Client Open: Ccb found.
Client Open: VMCF receive completed.
Active Open: Foreign Addr: 9.67.43.126
            Local Addr: 9.67.58.233
Client Open: sockets OK.
TcpRequest FindTcb: OurClientOwnsPort: FALSE, OtherClientOwnsPort: FALSE
Open: Tcb #1004 owned by TCPUSRX found in state Closed
New Open: Incoming buffer OK.
Open timeout set for 1504.859 seconds
New Open: Ready to send SYN.
DoOpen: ready to exit.
Open: Ready to OK open.
Client Open: ready to exit.
TCP-request called for ACB 13715840:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1378
   Timeout: 1504.859 seconds
      Client name: TCPUSRX
      Message identifier:26
      Client call: FReceive TCP
      Connection number: 1004
      Length: 65535
#1004 Established I=1 RNxt=1 CliRNxt=1 SNxt=1 SUna=1
SWnd=8192 MaxSWnd=8192 CWnd=536 Thresh=4096 Pen65535
TCP-request called for ACB 13715840:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1384
   Timeout: 1504.859 seconds
      Client name: TCPUSRX
      Message identifier:28
      Client call: FSend TCP
      Connection number: 1004
      Length: 14
TCP-request called for ACB 13715840:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1384
   Timeout: 1504.859 seconds
      Client name: TCPUSRX
      Message identifier:30
      Client call: FReceive TCP
      Connection number: 1004
      Length: 65535
#1004 Established I=2 O=1H1W8193 RNxt=131 CliRNxt=131 SNxt=15 SUna=1SWn d=8192
MaxSWnd=8192 CWnd=536 Thresh=4096  ConRe Pen65535
```

Figure 55 (Part 1 of 2). A Sample of a TCPREQUEST Trace Using MORETRACE

```
 .
 .
 .
TCP-request called for ACB 13715840:
   Accept TCP request -> TCP-request (from External interrupt handler)
   Last touched: 1398
      Client name: TCPUSRX
      Message identifier:36
      Client call: Open TCP
Client Open: Ccb found.
Client Open: VMCF receive completed.
Client Open: sockets OK.
TcpRequest FindTcb: OurClientOwnsPort: FALSE, OtherClientOwnsPort: FALSE
Open: Tcb #1000 owned by TCPUSRX found in state Closed
New Open: Incoming buffer OK.
Open timeout set for 1526.740 seconds
15:09:38 TCPUSRX Passive open #1000 Local = SA23, port 1036;
                           Foreign = RALVMM port Unspecified
TCPUSRX  has 3 sockets:
   Perm=F, AutoCli=F, Local=SA23 1033, TCB Q = 1
      1009 Closed, Foreign=RALVMM 21
   Perm=F, AutoCli=F, Local=SA23 1035, TCB Q = 1
      1004 Established, Foreign=RALVMM 21
   Perm=F, AutoCli=F, Local=SA23 1036, TCB Q = 1
      1000 Listen, Foreign=RALVMM 65535
DoOpen: ready to exit.
Open: Ready to OK open.
Client Open: ready to exit.
```

Figure 55 (Part 2 of 2). A Sample of a TCPREQUEST Trace Using MORETRACE

TELNET

Although the TELNET server is different from other protocols, TELNET must be traced like an internal TCPIP process. The TELNET trace includes events that are not specifically related to TELNET. It provides information about inbound and outbound negotiations, negotiated options, and the status of connections.

Table 8 describes the TELNET commands from RFC 854, when the codes and code sequences are preceded by an IAC. For more information about TELNET commands, see RFC 854. These commands can be retrieved in TELNET traces for SendNegotiation events and data. Subnegotiations that are started with an SB command, code 250 (X'FA') and code 240 (X'F0'), are also provided.

Table 8. Telnet Commands from RFC 854

Command	Code	Description
SE	240	End of subnegotiation parameters.
NOP	241	No operation.
Data Mark	242	The data stream portion of a Synch. This should always be accompanied by a TCP Urgent notification.
Break	243	NVT character BRK.
Interrupt Process	244	The function IP.
Abort output	245	The function AO.
Are You There	246	The function AYT.
Erase character	247	The function EC.
Erase Line	248	The function EL.
Go ahead	249	The GA signal.
SB	250	Indicates that what follows is subnegotiation of the indicated option.
WILL (option code)	251	Indicates the desire to begin performing, or confirmation that you are now performing, the indicated option.
WON'T (option code)	252	Indicates the refusal to perform, or continue performing, the indicated option.
DO (option code)	253	Indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the indicated option.
DON'T (option code)	254	Indicates the demand that the other party stop performing, or confirmation that you are no longer expecting the other party to perform, the indicated option.
IAC	255	Data Byte 255.

Table 9 lists the options available for TELNET commands from RFC1060, and RFC1647. For more information about TELNET protocols, see RFC's 1060, 1011 and 1647.

Table 9. Telnet Command Options from RFC 1060

Option	Name
0	Binary Transmission
1	Echo
2	Reconnection
3	Suppress Go Ahead
4	Approx Message Size Negotiation
5	Status
6	Timing Mark
7	Remote Controlled Trans and Echo
8	Output Line Width
9	Output Page Size
10	Output Carriage-Return Disposition
11	Output Horizontal Tab Stops
12	Output Horizontal Tab Disposition
13	Output Formfeed Disposition
14	Output Vertical Tabstops
15	Output Vertical Tab Disposition
16	Output Linefeed Disposition
17	Extended ASCII
18	Logout
19	Byte Macro
20	Data Entry Terminal
21	SUPDUP
22	SUPDUP Output
23	Send Location
24	Terminal Type
25	End of Record
26	TACACS User Identification
27	Output Marking
28	Terminal Location Number
29	Telnet 3270 Regime
30	X.3 PAD
31	Negotiate About Window Size
32	Terminal Speed
33	Remote Flow Control
34	Linemode
35	X Display Location
40	TN3270E
255	Extended-Options-List

Figure 56 shows a sample of a TELNET trace. A terminal type subnegotiation, option 24 X'18', is included in this sample. The urgent field in TCP datagrams is sometimes used for TELNET connections. For more information about the urgent field, see the DATA MARK command in Table 8 on page 93.

```
Internal client sees Acb:
13715528:
   Internal Telnet notification -> Internal Telnet server (from Notify)
   Last touched: 594
      Connection: 1007
      Notification: Connection state changed
         New state: Trying to open
         Reason: OK
TcpNoteGotten: Tag = Connection state changed
; NewState = Trying to open
Internal client sees Acb:
13715528:
   Internal Telnet notification -> Internal Telnet server (from Notify)
   Last touched: 594
      Connection: 1007
      Notification: Connection state changed
         New state: Open
         Reason: OK
TcpNoteGotten: Tag = Connection state changed
; NewState = Open
Conn 1: StToCpStateChanged: New state (ord) is 1
Conn 1: StToTcpStateChanged: New state (ord) is 1
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
Conn 1: in SendNegotiation: sending claim (ord)      253 for option (ord) 24
Conn 1: LenToSend: 3 ToTcpPos: 3 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 3
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
CONNECTION OPENED 09/26/90 at 13:17:04
STMASTER StateArray index: 1;  Tcp Conn#: 1007
Telnet server: Conn 1:Connection opened 09/26/90 at 13:17:04
Conn 1:    Foreign internet address and port:  net address = 9.67.58.226, port= 1059
   Foreign internet address and port:  net address = 9.67.58.226, port= 1059
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: StToCpGo returns TOcpDONE.
Internal client sees Acb:
13716568:
   Internal Telnet notification -> Internal Telnet server (from Notify)
   Last touched: 595
      Connection: 1007
      Notification: Data delivered
         Bytes delivered: 3
         Push flag: TRUE
TcpNoteGotten: Tag = Data delivered
```

Figure 56 (Part 1 of 3). A Sample of a TELNET Trace

```
Conn 1: StToCpStateChanged: New state (ord) is 5
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: StToCpGo returns TOcpTELNETdata.
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
Conn 1: Negot. received for TERMINALtype
Conn 1: in SendSEND
Conn 1: LenToSend: 6 ToTcpPos: 6 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 6
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: StToCpGo returns TOcpDONE.
Internal client sees Acb:
13716568:
   Internal Telnet notification -> Internal Telnet server (from Notify)
   Last touched: 595
      Connection: 1007
      Notification: Data delivered
         Bytes delivered: 18
         Push flag: TRUE
TcpNoteGotten: Tag = Data delivered
Conn 1: StToCpStateChanged: New state (ord) is 5
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: StToCpGo returns TOcpTELNETdata.
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
Conn 1: SB received for TERMINALtype
Conn 1: Terminal type is settled; it is: IBM-3278-2-E
Conn 1: TermTypeSubNeg. complete; Result is (ord) 3
Conn 1: in SendNegotiation: sending claim (ord)      253 for option (ord) 25
Conn 1: LenToSend: 3 ToTcpPos: 3 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 3
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: in SendNegotiation: sending claim (ord)      251 for option (ord) 25
Conn 1: LenToSend: 3 ToTcpPos: 3 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 3
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: in SendNegotiation: sending claim (ord)      253 for option (ord) 0
Conn 1: LenToSend: 3 ToTcpPos: 3 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 3
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: in SendNegotiation: sending claim (ord)      251 for option (ord) 0
Conn 1: LenToSend: 3 ToTcpPos: 3 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 3
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
```

Figure 56 (Part 2 of 3). A Sample of a TELNET Trace

```
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: StToCpGo returns TOcpDONE.
Internal client sees Acb:
13716360:
   Internal Telnet notification -> Internal Telnet server (from Notify)
   Last touched: 595
      Connection: 1007
      Notification: Data delivered
         Bytes delivered: 3
         Push flag: TRUE
TcpNoteGotten: Tag = Data delivered
Conn 1: StToCpStateChanged: New state (ord) is 5
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: StToCpGo returns TOcpTELNETdata.
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
Conn 1: Negot. received for USEeor
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: StToCpGo returns TOcpDONE.
```

Figure 56 (Part 3 of 3). A Sample of a TELNET Trace

Figure 57 shows a sample of a TELNET trace using the MORETRACE command. MORETRACE provides all of the data that is sent and received between two hosts connected by TELNET. The data is displayed in hexadecimal and EBCDIC characters and, therefore, you can trace the complete negotiations and data exchanges.

```
Internal client sees Acb:
13715216:
   Internal Telnet notification -> Internal Telnet server (from Notify)
   Last touched: 831
   Timeout: 778.669 seconds
      Connection: 1007
      Notification: Connection state changed
         New state: Trying to open
         Reason: OK
```

Figure 57 (Part 1 of 4). A Sample of a TELNET Trace Using MORETRACE

```
TcpNoteGotten: Tag = Connection state changed
; NewState = Trying to open
Internal client sees Acb:
13715216:
   Internal Telnet notification -> Internal Telnet server (from Notify)
   Last touched: 832
   Timeout: 778.669 seconds
      Connection: 1007
      Notification: Connection state changed
         New state: Open
         Reason: OK
TcpNoteGotten: Tag = Connection state changed
; NewState = Open
Conn 1: StToCpStateChanged: New state (ord) is 1
Conn 1: StToTcpStateChanged: New state (ord) is 1
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
Conn 1: in SendNegotiation: sending claim (ord) 253 for option ( ord) 24
Conn 1: LenToSend: 3 ToTcpPos: 3 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 3
FF FD 18
 }
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
CONNECTION OPENED 09/26/90 at 13:21:12
STMASTER StateArray index: 1;  Tcp Conn#: 1007
Telnet server: Conn 1:Connection opened 09/26/90 at 13:21:12
Conn 1:    Foreign internet address and port:  net address = 9.67.58.226, port= 1061
   Foreign internet address and port:  net address = 9.67.58.226, port= 1061
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: StToCpGo returns TOcpDONE.
Internal client sees Acb:
13716048:
   Internal Telnet notification -> Internal Telnet server (from Notify)
   Last touched: 832
      Connection: 1007
      Notification: Data delivered
         Bytes delivered: 3
         Push flag: TRUE
TcpNoteGotten: Tag = Data delivered
Conn 1: StToCpStateChanged: New state (ord) is 5
Conn 1: Telnet data received from TCP:
FF
FB
18

 Û
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: TnToCp Gobblechar: Found IAC at offset 0, FromTcpPos is 0
Conn 1: In GetIac: FirstChar is FB {
Conn 1: In GetIac: FirstChar is 18
Conn 1: StToCpGo returns TOcpTELNETdata.
```

Figure 57 (Part 2 of 4). A Sample of a TELNET Trace Using MORETRACE

```
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
Conn 1: Negot. received for TERMINALtype
Conn 1: in SendSEND
Conn 1: LenToSend: 6 ToTcpPos: 6 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 6
FF FA 18 01 FF F0
 z   p
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: StToCpGo returns TOcpDONE.
Internal client sees Acb:
13716464:
   Internal Telnet notification -> Internal Telnet server (from Internal Telnet
timeout handler)
   Last touched: 832
      Notification: Timer expired
         Datum: 2000, Associated timer: 1
TcpNoteGotten: Tag = Timer expired
Entering ScanConnections
Internal client sees Acb:
13716152:
   Internal Telnet notification -> Internal Telnet server (from Notify)
   Last touched: 832
      Connection: 1007
      Notification: Data delivered
         Bytes delivered: 18
         Push flag: TRUE
TcpNoteGotten: Tag = Data delivered
Conn 1: StToCpStateChanged: New state (ord) is 5
Conn 1: Telnet data received from TCP:
FF
FA
18
00
49
42
4D
2D
33
32
37
38
2D
32
2D
45
FF
F0

Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: TnToCp Gobblechar: Found IAC at offset 0, FromTcpPos is 0
Conn 1: In GetIac: FirstChar is FA z
Conn 1: In GetIac: FirstChar is 18
```

Figure 57 (Part 3 of 4). A Sample of a TELNET Trace Using MORETRACE

```
Conn 1: In GetIac: FirstChar is 00
Conn 1: In GetIac: FirstChar is 49 I
Conn 1: In GetIac: FirstChar is 42 B
Conn 1: In GetIac: FirstChar is 4D M
Conn 1: In GetIac: FirstChar is 2D -
Conn 1: In GetIac: FirstChar is 33 3
Conn 1: In GetIac: FirstChar is 32 2
Conn 1: In GetIac: FirstChar is 37 7
Conn 1: In GetIac: FirstChar is 38 8
Conn 1: In GetIac: FirstChar is 2D -
Conn 1: In GetIac: FirstChar is 32 2
Conn 1: In GetIac: FirstChar is 2D -
Conn 1: In GetIac: FirstChar is 45 E
Conn 1: In GetIac: FirstChar is FF
Conn 1: In GetIac: FirstChar is F0 p
Conn 1: StToCpGo returns TOcpTELNETdata.
Schedule called. FirstOneToDo = -1; LastOneToDo = -1; NextToDo = -1
Conn 1: SB received for TERMINALtype
Conn 1: Terminal type is settled; it is: IBM-3278-2-E
Conn 1: TermTypeSubNeg. complete; Result is (ord) 3
Conn 1: in SendNegotiation: sending claim (ord) 253 for option ( ord) 25
Conn 1: LenToSend: 3 ToTcpPos: 3 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 3
FF FD 19
 }
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: in SendNegotiation: sending claim (ord) 251 for option ( ord) 25
Conn 1: LenToSend: 3 ToTcpPos: 3 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 3
FF FB 19
 {
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: in SendNegotiation: sending claim (ord) 253 for option ( ord) 0
Conn 1: LenToSend: 3 ToTcpPos: 3 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 3
FF FD 00
 }
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: in SendNegotiation: sending claim (ord) 251 for option ( ord) 0
Conn 1: LenToSend: 3 ToTcpPos: 3 UrgentHighWaterMark: -1
Conn 1: TcpSend: TurnCode = OK; LenToSend = 3
FF FB 00
 {
Conn 1:  TcpSend successful --
 ToTcpPos: 0 UrgentHighWaterMark: -1
Conn 1: LenToSend: 0 ToTcpPos: 0 UrgentHighWaterMark: -1
MainLoop calling 1; LastOneToDo = 1; NextToDo = -1
Conn 1: CallToCp Which Conn = 1
Conn 1: Urginfo: Mode is ;  Number bytes is 0
Conn 1: StToCpGo returns TOcpDONE.
```

Figure 57 (Part 4 of 4). A Sample of a TELNET Trace Using MORETRACE

TIMER

The TIMER trace shows the processes with time-out marks. Figure 58 shows a sample of a TIMER trace.

```
In SetTheComparator, time is: 1809.320 seconds
Setting clock comparator to 1819.320 seconds
In SetTheComparator, time is: 1809.464 seconds
Setting clock comparator to 1821.709 seconds
Timer called at 1821.711 seconds
Timeout due: Internal Telnet timeout handler = Internal Telnet timeout
  -> 2 pending timeouts left; 1 active signals
In SetTheComparator, time is: 1821.724 seconds
Setting clock comparator to 1831.011 seconds
Timer called at 1831.014 seconds
Timeout due: Consistency checker = Check consistency
  -> 2 pending timeouts left; 1 active signals
In SetTheComparator, time is: 1831.026 seconds
Setting clock comparator to 1941.737 seconds
In SetTheComparator, time is: 1831.066 seconds
Setting clock comparator to 1891.066 seconds
In SetTheComparator, time is: 1845.219 seconds
Setting clock comparator to 1855.219 seconds
In SetTheComparator, time is: 1845.295 seconds
Setting clock comparator to 1891.066 seconds
In SetTheComparator, time is: 1854.782 seconds
Setting clock comparator to 1864.781 seconds
Timer called at 1864.784 seconds
Timeout due: Ping process = Ping timeout fails
  -> 4 pending timeouts left; 1 active signals
In SetTheComparator, time is: 1864.797 seconds
Setting clock comparator to 1891.066 seconds
```

Figure 58. A Sample of a TIMER Trace

When you execute a TIMER trace with the MORETRACE command, it provides details about each timer event and request from a process. Figure 59 shows a sample of a TIMER trace using MORETRACE.

```
PutAcbInOrder adding Acb:
13715216:
   Ping timeout fails -> No process! (from Timer)
   Last touched: 1812
   Timeout: 1910.945 seconds
In PutAcbInOrder, timer queue is
The time is 1900.966 seconds

Timer Queue:Queue size = 5
   13715216:
   PrevACB: Timer queue
   NextACB: 13714904
              QueueHead:Timer queue
   Ping timeout fails -> No process! (from Timer)
   Last touched: 1812
   Timeout: 1910.945 seconds
```

Figure 59 (Part 1 of 2). A Sample of a TIMER Trace Using MORETRACE

```
   13714904:
   PrevACB: 13715216
   NextACB: 13715632
              QueueHead:Timer queue
   Internal Telnet timeout -> Internal Telnet timeout handler (from Timer)
   Last touched: 1737
   Timeout: 1941.737 seconds
      Timer Datum: 2000, Timer Number: 1

   13715632:
   PrevACB: 13714904
   NextACB: 13716048
              QueueHead:Timer queue
   Check consistency -> Consistency checker (from Timer)
   Last touched: 1803
   Timeout: 1951.205 seconds

   13716048:
   PrevACB: 13715632
   NextACB: 13715320
              QueueHead:Timer queue
   ARP timeout expires -> ARP (from Timer)
   Last touched: 1769
   Timeout: 2034.862 seconds

   13715320:
   PrevACB: 13716048
   NextACB: Timer queue
              QueueHead:Timer queue
   Open timeout fails #1006 -> TCP-request (from Timer)
   Last touched: 71
   Timeout: 604874.674 seconds

In SetTheComparator, time is: 1901.123 seconds
Setting clock comparator to 1910.945 seconds
CancelTimeout removing ACB:
13715216:
   PrevACB: Timer queue
   NextACB: 13714904
              QueueHead:Timer queue
   Ping timeout fails -> Ping process (from Timer)
   Last touched: 1812
   Timeout: 1910.945 seconds
In SetTheComparator, time is: 1901.251 seconds
Setting clock comparator to 1941.737 seconds
PutAcbInOrder adding Acb:
13715736:
   Ping timeout fails -> No process! (from Timer)
   Last touched: 1827
   Timeout: 1926.436 seconds
In PutAcbInOrder, timer queue is
The time is 1916.457 seconds
```

Figure 59 (Part 2 of 2). A Sample of a TIMER Trace Using MORETRACE

UDPREQUEST

The UDPREQUEST trace provides information about all UDP service requests from local clients and servers. Figure 60 shows a sample of a UDPREQUEST trace.

```
UDP-request called for ACB 13706816:
   Accept UDP request -> UDP-request (from External interrupt handler)
      Client name: VMNFS
      Message identifier:10
      Client call: Open UDP
      Connection number: 0
UDP-request called for ACB 13706816:
   Accept UDP request -> UDP-request (from External interrupt handler)
      Client name: VMNFS
      Message identifier:14
      Client call: Send UDP
      Connection number: 0
      VadA: 0075A028, LenA: 56, VadB: 111, LenB: 14.0.0.0
UDP-request: Local Socket:
       net address = *, port= 2049
UDP-request: Foreign Socket:
       net address = 14.0.0.0, port= PORTMAP (111)
UDP-request called for ACB 13706608:
   Accept UDP request -> UDP-request (from External interrupt handler)
      Client name: VMNFS
      Message identifier:16
      Client call: Receive UDP
      Connection number: 0
UDP-request called for ACB 13707128:
   Accept UDP request -> UDP-request (from External interrupt handler)
      Client name: VMNFS
      Message identifier:18
      Client call: Send UDP
      Connection number: 0
      VadA: 0075A028, LenA: 56, VadB: 111, LenB: 14.0.0.0
UDP-request: Local Socket:
       net address = *, port= 2049
UDP-request: Foreign Socket:
       net address = 14.0.0.0, port= PORTMAP (111)
```

Figure 60. A Sample of a UDPREQUEST Trace

When you execute a UDPREQUEST trace using the MORETRACE command, it adds information about datagram checksums and UCBs. Figure 61 shows a sample of the UDPREQUEST trace using MORETRACE.

```
UDP-checksum: datagram = 8DD1 pseudo-header = 88AE final = E97F
UDP-checksum: datagram = C48C pseudo-header = 88C8 final = B2AA
UDP-checksum: datagram = 8DD1 pseudo-header = 88AD final = E980
```

Figure 61 (Part 1 of 2). A Sample of a UDPREQUEST Trace Using MORETRACE

```
UDP-request called for ACB 13706504:
   Accept UDP request -> UDP-request (from External interrupt handler)
   Timeout: 1190.772 seconds
      Client name: VMNFS
      Message identifier:10
      Client call: Open UDP
      Connection number: 0
UDP-request: Ccb found.
UDP-request: Client UdpOpen called.
ClientUDPOpen: Response.Connection = 34817
UDP-request called for ACB 13706504:
   Accept UDP request -> UDP-request (from External interrupt handler)
   Timeout: 1190.772 seconds
      Client name: VMNFS
      Message identifier:14
      Client call: Send UDP
      Connection number: 0
      VadA: 0075A028, LenA: 56, VadB: 111, LenB: 14.0.0.0
UDP-request: Ccb found.
UDP-request: Client UdpSend called.
CheckClient: Ucb found
5028920:
   PrevUcb: 12952304
   NextUcb: 12952304
   BytesIn: 0, BytesOut: 0
   Socket:
VMNFS has 0 TCBs for socket *.2049 *Perm *Autolog   ConnIndex: 0, Frustration: Contented
   IncomingDatagram queue size: 0
   ShouldChecksum: TRUE, UdpReceivePending: FALSE,WhetherDatagramDelivered: FALSE
UDP-request: Local Socket:
       net address = *, port= 2049
UDP-request: Foreign Socket:
       net address = 14.0.0.0, port= PORTMAP (111)
UDP-request: Udp-Send: sending 64 byte UDP datagram.
UDP-checksum: datagram = 15FF pseudo-header = 1C51 final = CDAF
UDP-checksum: datagram = 15FF pseudo-header = 1C51 final = CDAF
UDP-checksum: datagram = 0896 pseudo-header = 1C35 final = DB34
UDP-checksum: datagram = 0896 pseudo-header = 1C35 final = DB34
```

Figure 61 (Part 2 of 2). A Sample of a UDPREQUEST Trace Using MORETRACE

UDPUP

The UDPUP trace provides information about incoming UDP datagrams. Figure 62 shows a sample of a UDPUP trace using the MORETRACE command with a remote VM/NFS server and a local Portmapper client. Note that the control blocks for UDP connections are UCBs and not TCBs.

```
DASD 3EE DETACHED
UptoUDP called:
UptoUDP: Destination port # 34078936
UptoUDP: Ucb not found - dropping datagram
UptoUDP called:
UptoUDP: Destination port # 34078936
UptoUDP: Ucb not found - dropping datagram
UptoUDP called:
UptoUDP: Destination port # 34078929
UptoUDP: Ucb not found - dropping datagram
UptoUDP called:
UptoUDP: Destination port # 7274560
UptoUDP: Ucb found:
5028816:
   PrevUcb: 12686112
   NextUcb: 12686112
   BytesIn: 0, BytesOut: 0
   Socket:
PORTMAP has 0 TCBs for socket *.PORTMAP (111)   ConnIndex: -23, Frustration: Contented
   IncomingDatagram queue size: 0
   ShouldChecksum: TRUE, UdpReceivePending: FALSE,WhetherDatagramDelivered: FALSE
UptoUDP called:
UptoUDP: Destination port # 134283479
UptoUDP: Ucb found:
5028920:
   PrevUcb: 12952304
   NextUcb: 12952304
   BytesIn: 0, BytesOut: 64
   Socket:
VMNFS has 0 TCBs for socket *.2049 *Perm *Autolog   ConnIndex: 0, Frustration: Contented
   IncomingDatagram queue size: 0
   ShouldChecksum: TRUE, UdpReceivePending: FALSE,WhetherDatagramDelivered: FALSE
UptoUDP called:
UptoUDP: Destination port # 7274560
UptoUDP: Ucb found:
5028816:
   PrevUcb: 12686112
   NextUcb: 12686112
   BytesIn: 56, BytesOut: 36
   Socket:
PORTMAP has 0 TCBs for socket *.PORTMAP (111)   ConnIndex: -23, Frustration: Contented
   IncomingDatagram queue size: 0
   ShouldChecksum: TRUE, UdpReceivePending: FALSE,WhetherDatagramDelivered: FALSE
```

Figure 62. A Sample of a UDPUP Trace Using MORETRACE

Group Process Names

Group process names combine more than one single process into the same process name. In all trace commands, TRACE, NOTRACE, MORETRACE, and LESSTRACE, you can enter more than one group process name.

ALL

The ALL trace provides information about all available events. You must be very careful when using the ALL trace, because it can overwhelm the console and adversely affect system response time.

CETI

The CETI group process combines TOCETI, ELANS, ILANS, and TOX25ICA traces. This group traces all activities related to 9370 integrated adapters. CETI provides information about CCW addresses, CCW status, and packet information such as protocols, LLC types, node addresses, and ARP address translations performed by CETI adapters. Figure 63. shows a sample of a CETI trace.

```
13714696:
   Have completed I/O -> To-CETI (from Scheduler)
   Last touched: 33
      IoDevice 0240
      Csw:
         Keys: 00, CcwAddress: 00000000
         Unit Status: 80, Channel Status: 00
         Byte Count: 0
Device status is Ready
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 1
ILANS ILANS1: CheckCetiIo finds inbound msg uses 1 buffers.  Msg type 33554538
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS ILANS1: Entering GetCetiMessage:  Max len 2074 PeekOnly 54042
ILANS ILANS1: Entering UpdateControlBlocks. UsedBuffs 1
ILANS device ILANS1: CIOA MAC primitive type is CC31
ILANS device ILANS1: DLM_RTV_ATTRIB.confirm: Completion status 00000000
Node address: 10005A4209A0
ILANS device ILANS1: Sending DL_ACTIVATE_SAP.request
ILANS ILANS1: Entering SendCetiMessage: DataLen 88 MsgType 4 MsgFlags '00'X
ILANS ILANS1: CheckCetiSpace returns SpaceAvail 41680, Cspace 37512
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 0
ILANS ILANS1: Entering EnableCetiRupt
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 0
ILANS ILANS1: ToCeti: Acb Received:
 .
 .
 .
13714696:
   Have completed I/O -> To-CETI (from Scheduler)
   Last touched: 34
      IoDevice 0240
      Csw:
         Keys: 00, CcwAddress: 00000000
         Unit Status: 80, Channel Status: 00
         Byte Count: 0
Device status is Ready
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 1
ILANS ILANS1: CheckCetiIo finds inbound msg uses 1 buffers.  Msg type 67108929
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS ILANS1: Entering GetCetiMessage:  Max len 2074 PeekOnly 16831258
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS ILANS1: Entering GetCetiMessage:  Max len 2085 PeekOnly 13482920
ILANS ILANS1: Entering UpdateControlBlocks. UsedBuffs 1
ILANS device ILANS1: Entering DispatchTr: EtherType '0806'X
Arp adds translation9.67.58.226 = IBMTR: 10005A0019F5
ILANS ILANS1: Entering CheckCetiOutput. Queue sizes: 0 1
ILANS ILANS1: Entering SendCetiMessage: DataLen 63 MsgType 4 MsgFlags '00'X
ILANS ILANS1: CheckCetiSpace returns SpaceAvail 41680, Cspace 35428
ILANS ILANS1: SendCetiMessage returns 0
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 0
ILANS ILANS1: Entering EnableCetiRupt
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 0
ILANS ILANS1: ToCeti: Acb Received:
```

Figure 63. A Sample of a CETI Trace

When you execute a CETI trace using the MORETRACE command, trace data is in hexadecimal form with a MESSAGE TEXT entry, and details about IP packets are provided. Figure 64 shows a sample of a CETI trace using MORETRACE.

```
13714696:
   Send datagram -> Device driver(ILANS1) (from UDP-request)
   Last touched: 32
Device status is Ready
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 0
ILANS ILANS1: Entering CheckCetiOutput. Queue sizes: 1 0
ILANS ILANS1: Entering SendCetiMessage: DataLen 65 MsgType 4 MsgFlags '00'X
ILANS ILANS1: CheckCetiSpace returns SpaceAvail 41680, Cspace 35428
Message Text
000000:001E0D11 00230000 000711DA FFFFFFFF  FFFFAA00 00FF0002 00210000 80008220
000020:00000008 06000608 00060400 0110005A  4209A009 433AE94E 38001616 9009433A
000040:EA000101
ILANS ILANS1: SendCetiMessage returns 0
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 2
ILANS ILANS1: CheckCetiIo finds inbound msg uses 1 buffers.  Msg type 67108929
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS ILANS1: Entering GetCetiMessage:  Max len 2074 PeekOnly 16831258
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS ILANS1: Entering GetCetiMessage:  Max len 2085 PeekOnly 13482920
ILANS ILANS1: Entering UpdateControlBlocks. UsedBuffs 1
Message Text
000000:001E4D11 00230000 00000000 00000000  10005A42 09A0AA01 00020021 00010220
000020:00000008 06000608 00060400 0110005A  4209A009 433AE94E 38001616 9009433A
000040:EA000101
ILANS device ILANS1: Entering DispatchTr: EtherType '0806'X
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 1
ILANS ILANS1: CheckCetiIo finds inbound msg uses 1 buffers.  Msg type 67108929
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS ILANS1: Entering GetCetiMessage:  Max len 2074 PeekOnly 16831258
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS ILANS1: Entering GetCetiMessage:  Max len 2085 PeekOnly 13482920
ILANS ILANS1: Entering UpdateControlBlocks. UsedBuffs 1
Message Text
000000:001E4D11 00230000 00000000 00000000  10005A25 0858AA00 00020021 000202A0
000020:00000008 06000608 00060400 0210005A  25085809 433AEA10 005A4209 A009433A
000040:E9000101
ILANS device ILANS1: Entering DispatchTr: EtherType '0806'X
Arpin: Processing Arp packet:
   ArpHardwareType: 6
   ArpProtocolType: 2048
   ArpHardwareLen: 6
   ArpProtocolLen: 4
   ArpOp: 0
   ArpSenderHardwareAddr: 10005A250858
   ArpSenderInternetAddr: 9.67.58.234
   ArpTargetHardwareAddr: 10005A4209A0
   ArpTargetInternetAddr: 9.67.58.233
Arp adds translation9.67.58.234 = IBMTR: 10005A250858
ILANS ILANS1: Entering CheckCetiOutput. Queue sizes: 1 0
ILANS ILANS1: Entering SendCetiMessage: DataLen 112 MsgType 4 MsgFlags '00'X
ILANS ILANS1: CheckCetiSpace returns SpaceAvail 41680, Cspace 33344
Message Text
000000:001E0D11 00520000 000711DA 10005A25  0858AA00 00FF0000 00520000 80000000
000020:00080045 00004D00 2B00003C 1105A309  433AE909 432B6404 00003500 39ED0000
000040:01010000 01000000 00000006 52414C56  4D4D0854 43504950 44455607 52414C45
000060:49474803 49424D03 434F4D00 00010001
```

Figure 64 (Part 1 of 2). A Sample of a CETI Trace Using MORETRACE

```
ILANS ILANS1: SendCetiMessage returns 0
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 1
ILANS ILANS1: CheckCetiIo finds inbound msg uses 1 buffers.  Msg type 67109025
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS ILANS1: Entering GetCetiMessage:  Max len 2074 PeekOnly 16831258
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS ILANS1: Entering GetCetiMessage:  Max len 2085 PeekOnly 13480744
ILANS ILANS1: Entering UpdateControlBlocks. UsedBuffs 1
Message Text
000000:001E4D11 00830000 00000000 00000000  10005A25 0858AA00 00020081 000302A0
000020:00000008 00450000 7C75E700 001C11AF  B709432B 6409433A E9003504 000068B3
000040:B7000185 80000100 01000000 00065241  4C564D4D 08544350 49504445 56075241
000060:4C454947 48034942 4D03434F 4D000001  00010652 414C564D 4D085443 50495044
000080:45560752 414C4549 47480349 424D0343  4F4D0000 01000100 000E1000 0409432B
0000A0:7E000000
ILANS device ILANS1: Entering DispatchTr: EtherType '0800'X
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 0
ILANS ILANS1: Entering EnableCetiRupt
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 0
ILANS ILANS1: ToCeti: Acb Received:
13715008:
   Send datagram -> Device driver(ILANS1) (from Ping process)
   Last touched: 33
Device status is Ready
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 0
ILANS ILANS1: Entering CheckCetiOutput. Queue sizes: 1 0
ILANS ILANS1: Entering SendCetiMessage: DataLen 311 MsgType 4 MsgFlags '00'X
ILANS ILANS1: CheckCetiSpace returns SpaceAvail 41680, Cspace 31260
Message Text
000000:001E0D11 01190000 000711DA 10005A25  0858AA00 00FF0000 01190000 80000000
000020:00080045 00011404 D200003C 01002B09  433AE909 432B7E08 0024E300 D1450847
000040:83D5AB53 8D8B5B7F D6A37F8D 5B7BED22  725C9264 423E7918 272FED6B B96804B1
000060:0466C527 80039D78 BB4F9753 A20A5239  85D4A95D 53DAB802 6D9D1128 2B06E1DE
000080:16C95F2B CC3A08C6 7E7200BB C8C0E411  E3C5A876 C22A6D72 13476F4D F03EC934
0000A0:2902F94E 5CB88074 F30133FA 1C8BCBD9  45B79BD3 9BB35A5D A10668B3 8F20E0CC
0000C0:8250C82B 63ACBD0D 215AEE3B DBC996DB  6FB57B91 48EC5639 82E837FB 0EDFE4F3
0000E0:91D1AF3C 137D29B8 AF577323 E897B64E  A2121D6B 8B7FA5CF A9642BC5 621D1D62
000100:C23B0AB5 E035128D C9E30B09 EB9E8E3C  37A51607 F08329B6 BC093AC8 40E1A184
000120:73F5F573 86971EE1 C2BA0B30 05E2D933  2136C553 75192300
ILANS ILANS1: SendCetiMessage returns 0
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 1
ILANS ILANS1: CheckCetiIo finds inbound msg uses 1 buffers.  Msg type 67109177
ILANS device ILANS1: Entering IlansPortInput: MsgType 0
ILANS ILANS1: Entering GetCetiMessage:  Max len 2074 PeekOnly 16831258
ILANS device ILANS1: CIOA LLC primitive type is 4D11
ILANS ILANS1: Entering GetCetiMessage:  Max len 2085 PeekOnly 13480744
ILANS ILANS1: Entering UpdateControlBlocks. UsedBuffs 1
Message Text
000000:001E4D11 011B0000 00000000 00000000  10005A25 0858AA00 00020119 000402A0
000020:00000008 00450001 1404D200 003A0102  2B09432B 7E09433A E900002C E300D145
000040:084783D5 AB538D8B 5B7FD6A3 7F8D5B7B  ED22725C 9264423E 7918272F ED6BB968
000060:04B10466 C5278003 9D78BB4F 9753A20A  523985D4 A95D53DA B8026D9D 11282B06
000080:E1DE16C9 5F2BCC3A 08C67E72 00BBC8C0  E411E3C5 A876C22A 6D721347 6F4DF03E
0000A0:C9342902 F94E5CB8 8074F301 33FA1C8B  CBD945B7 9BD39BB3 5A5DA106 68B38F20
0000C0:E0CC8250 C82B63AC BD0D215A EE3BDBC9  96DB6FB5 7B9148EC 563982E8 37FB0EDF
0000E0:E4F391D1 AF3C137D 29B8AF57 7323E897  B64EA212 1D6B8B7F A5CFA964 2BC5621D
000100:1D62C23B 0AB5E035 128DC9E3 0B09EB9E  8E3C37A5 1607F083 29B6BC09 3AC840E1
000120:A18473F5 F5738697 1EE1C2BA 0B3005E2  D9332136 C5537519 23000000
ILANS device ILANS1: Entering DispatchTr: EtherType '0800'X
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 0
ILANS ILANS1: Entering EnableCetiRupt
ILANS ILANS1: CheckCetiIo finds inbound port has FilledBuffs 0
ILANS ILANS1: ToCeti: Acb Received:
```

Figure 64 (Part 2 of 2). A Sample of a CETI Trace Using MORETRACE

HANDLERS

The HANDLERS group process combines A220 handler, external interrupt handler, I/O interrupt handler, DDN1822 I/O interrupt handler, IUCV handler, and PCCA handler traces.

HCH

The HCH group process combines A220 handler and A220 common routine traces.

IUCV

The IUCV group process combines IUCV handler and TOIUCV traces. It provides information about IUCV activities. Figure 65 shows a sample of an IUCV trace in which the local TCPIP client is TCPIP1, the other local TCPIP server is user TCPIP2, and the device name is LOCIUVC.

Figure 65 also shows an ICMP trace. An ICMP datagram with an ICMP request code of 8 and a PING trace executed from TCPIP2 is also shown.

```
TCPIP1   AT VMHOST01  VIA RSCS     09/26/97 14:34:12 EST      WEDNESDAY
VM TCP/IP V2R4
   Initializing...
UnlockAll issuing "CP UNLOCK TCPIP1 0 DFF"
COMMAND COMPLETE
LCS devices will use diagnose 98 real channel program support
Trying to open VMHOST01 TCPIP *
Using profile file VMHOST01 TCPIP *
IUCV initializing:
 Device LOCIUCV:
    Type: PVM IUCV, Status: Not started
    Envelope queue size: 0
    VM id: TCPIP2
    UserDoubleWord 1: XYZZY, UserDoubleWord 2: XYZZY
    Our PVM node: A
PVM IUCV LOCIUCV         : ToIucv IssueConnect: Vm Id: TCPIP2, DWord1: XYZZY, DWord2: XYZZY
PVM IUCV LOCIUCV         : ToIucv: Connect returns pathid 1
Telnet server: Using port 23
Telnet server: No inactivity timeout
Telnet server: Every 1800 seconds a timing mark option packet will be sent.
******************************************
Log of IBM TCP/IP Telnet Server Users started on 09/26/90 at 14:35:04

TCP-IP initialization complete.
ToIucv: Acb Received:
13592024:
   IUCV interrupt -> To-IUCV (from External interrupt handler)
   Last touched: 48
   Interrupt type: Pending connection
      Path id: 0
      VMid: TCPIP2, User1: XYZZY, User2: XYZZY
```

Figure 65 (Part 1 of 3). A Sample of an IUCV Trace

```
ToIucv: Received PENDCONN. pendcuser1: XYZZY, pendcuser2: XYZZY, pendcvmid: TCPIP2,
   IucvPathid: 0
 Device LOCIUCV:
    Type: PVM IUCV, Status: Issued connect
    Envelope queue size: 0
    VM id: TCPIP2
    UserDoubleWord 1: XYZZY, UserDoubleWord 2: XYZZY
    Our PVM node: A
ToIucv: Severing path 1
PVM IUCV LOCIUCV         : ToIucv: Accepting path 0
PVM IUCV LOCIUCV         : ToIucv PackWrites: Queuesize, SavedEnv: 0 0
Telnet server: Global connection to *CCS CP System Service established
Telnet server: First line of *CCS logo is: VIRTUAL MACHINE/SYSTEM PRODUCT

ToIucv: Acb Received:
13591920:
   Try IUCV connect -> To-IUCV (from Timer)
   Last touched: 103
 Device LOCIUCV:
    Type: PVM IUCV, Status: Connected
    Envelope queue size: 0
    VM id: TCPIP2
    UserDoubleWord 1: XYZZY, UserDoubleWord 2: XYZZY
    Our PVM node: A
ToIucv: Acb Received:
13591920:
   IUCV interrupt -> To-IUCV (from External interrupt handler)
   Last touched: 187
   Interrupt type: Pending message
      Path id: 0
      MsgId 1586, Length 280, TrgCls: 00000000, Reply len 0, Flags 17
 Device LOCIUCV:
    Type: PVM IUCV, Status: Connected
    Envelope queue size: 0
    VM id: TCPIP2
    UserDoubleWord 1: XYZZY, UserDoubleWord 2: XYZZY
    Our PVM node: A
PVM IUCV LOCIUCV         : ToIucv UnpackReads:  bytestomove = 276
   IP-up sees ICMP datagram, code 8, sub code: 0, source: HOST02, dest: HOST01, len: 256
PVM IUCV LOCIUCV    : IUCV UnpackReads: BlockHeader copied from InputPosition: 12672 278
PVM IUCV LOCIUCV    : ToIUCV UnpackReads:  PacketsInInBlock = 1
ToIucv: Acb Received:
13592440:
   Send datagram -> Device driver(LOCIUCV) (from To-IUCV)
   Last touched: 188
 Device LOCIUCV:
    Type: PVM IUCV, Status: Connected
    Envelope queue size: 1
    VM id: TCPIP2
    UserDoubleWord 1: XYZZY, UserDoubleWord 2: XYZZY
    Our PVM node: A
PVM IUCV LOCIUCV         : ToIucv PackWrites: Queuesize, SavedEnv: 1 0
PVM IUCV LOCIUCV         : PackWrites packing packet with length 276
```

Figure 65 (Part 2 of 3). A Sample of an IUCV Trace

```
ToIucv: Acb Received:
13592440:
   IUCV interrupt -> To-IUCV (from External interrupt handler)
   Last touched: 188
   Interrupt type: Pending message completion
      Path id: 0
      audit: 0000
 Device LOCIUCV:
    Type: PVM IUCV, Status: Sending message
    Envelope queue size: 0
    VM id: TCPIP2
    UserDoubleWord 1: XYZZY, UserDoubleWord 2: XYZZY
    Our PVM node: A
PVM IUCV LOCIUCV         : ToIUCV write complete.  PacketsInOutBlock = 1
PVM IUCV LOCIUCV         : ToIucv PackWrites: Queuesize, SavedEnv: 0 0

#CP EXT
14:37:39 09/26/90 Shutdown KILL TCB #1000 (INTCLIEN) TCP/IP service is being shut down
    Bytes: 0 sent, 0 received
    Max use: 0 in retransmit Q

1 active client, with 1 connection in use.
I will delay shutting down for 30 seconds, so that
RSTs and shutdown notifications may be delivered.
If you wish to shutdown immediately, without warning,
type #CP EXT again.

Server Telnet closed down. Bye.
ToIucv: Acb Received:
13591816:
   Device-specific activity -> To-IUCV (from Timer)
   Last touched: 217
 Device LOCIUCV:
    Type: PVM IUCV, Status: Connected
    Envelope queue size: 0
    VM id: TCPIP2
    UserDoubleWord 1: XYZZY, UserDoubleWord 2: XYZZY
    Our PVM node: A
IUCV shutting down:
 Device LOCIUCV:
    Type: PVM IUCV, Status: Connected
    Envelope queue size: 0
    VM id: TCPIP2
    UserDoubleWord 1: XYZZY, UserDoubleWord 2: XYZZY
    Our PVM node: A
ToIucv: Severing path 0
UnlockAll issuing "CP UNLOCK TCPIP 0 DFF"
COMMAND COMPLETE
ShutDown at 234.795 seconds
```

Figure 65 (Part 3 of 3). A Sample of an IUCV Trace

PCCA

The PCCA group process combines PCCA handler and PCCA common routine traces. It provides information about I/O operations to be performed on the channel-attached LAN adapters. The trace output lists the device, type, CCW address, CCW operation, number of bytes, and unit status of I/O requested operations.

Figure 66 shows a sample of a PCCA trace in which an ACB (13715112) acquires the home hardware address for link TR2 with ctrlcommand 04 on networktype 2, adapter 1. Figure 66 also shows an ACB with an ARP address translation for IP address 9.67.58.234. For more information about the commands used in this trace, see “CCW” on page 206.

```
ToPcca3: Acb Received:
13715112:
   Have completed I/O -> PCCA3 common routine (from PCCA3 handler)
   Last touched: 20
      IoDevice 0560
      Csw:
         Keys: E0, CcwAddress: 007B7118
         Unit Status: 0C, Channel Status: 00
         Byte Count: 20402
      Device LCS1:
         Type: LCS, Status: Ready
         Envelope queue size: 0
         Address: 0560
PCCA3 device LCS1: Received PCCA control packet:
PccaCtrlCommand: 4, PccaCtrlNetType2: 2, PccaCtrlAdapter2: 1
PccaCtrlRetcode: 0, PccaCtrlSequence: 0, PccaCtrlFlags: 00
PccaCtrlHardwareAddress: 10005A6BAFDF
PCCA3 device LCS1: PCCA reports home hardware address 10005A6BAFDF for link TR2
PCCA3 device LCS1: ToPcca3: BlockHeader copied from InputPosition: 0 76
PCCA3 device LCS1: ToPcca UnpackReads:  PacketsInInBlock = 1
PCCA3 device LCS1: CallSio: Starting I/O on device 0560. First command 02, UseDiag98 True
PCCA3 device LCS1: ToPcca PackWrites: Queuesizes, SavedEnv: 0 0 0
ToPcca3: Acb Received:
13715008:
   Send datagram -> PCCA3 common routine (from PCCA3 common routine)
   Last touched: 20
      Device LCS1:
         Type: LCS, Status: Ready
         Envelope queue size: 0
         Address: 0560
PCCA3 device LCS1: ToPcca PackWrites: Queuesizes, SavedEnv: 0 0 0
ToPcca3: Acb Received:
13715008:
   Send datagram -> Device driver(LCS1) (from UDP-request)
   Last touched: 23
      Device LCS1:
         Type: LCS, Status: Ready
         Envelope queue size: 1
         Address: 0560
PCCA3 device LCS1: ToPcca PackWrites: Queuesizes, SavedEnv: 0 1 0
PCCA3 device LCS1: ToPcca PackWrites: LengthOfData, BlockHeader: 54 56
PCCA3 device LCS1: CallSio: Starting I/O on device 0561. First command 01, UseDiag98 True
```

Figure 66 (Part 1 of 2). A Sample of a PCCA Trace

```
ToPcca3: Acb Received:
13715008:
   Have completed I/O -> PCCA3 common routine (from PCCA3 handler)
   Last touched: 23
      IoDevice 0561
      Csw:
         Keys: E0, CcwAddress: 007B70C0
         Unit Status: 0C, Channel Status: 00
         Byte Count: 0
      Device LCS1:
         Type: LCS, Status: Ready
         Envelope queue size: 0
         Address: 0560
PCCA3 device LCS1: ToPcca write complete. PacketsInOutBlock = 1
PCCA3 device LCS1: ToPcca PackWrites: Queuesizes, SavedEnv: 0 0 0
ToPcca3: Acb Received:
13715008:
   Have completed I/O -> PCCA3 common routine (from PCCA3 handler)
   Last touched: 23
      IoDevice 0560
      Csw:
         Keys: E0, CcwAddress: 007B7118
         Unit Status: 0C, Channel Status: 00
         Byte Count: 20422
      Device LCS1:
         Type: LCS, Status: Ready
         Envelope queue size: 0
         Address: 0560
PCCA3 device LCS1: UnpackReads: NetType 2 AdapterNumber 0 BytesToMove 54
PCCA3 device LCS1: ToPcca3: BlockHeader copied from InputPosition: 0 56
PCCA3 device LCS1: ToPcca UnpackReads:  PacketsInInBlock = 1
PCCA3 device LCS1: CallSio: Starting I/O on device 0560. First command 02, UseDiag98 True
PCCA3 device LCS1: ToPcca PackWrites: Queuesizes, SavedEnv: 0 0 0
ToPcca3: Acb Received:
13715008:
   Have completed I/O -> PCCA3 common routine (from PCCA3 handler)
   Last touched: 23
      IoDevice 0560
      Csw:
         Keys: E0, CcwAddress: 007B7118
         Unit Status: 0C, Channel Status: 00
         Byte Count: 20422
      Device LCS1:
         Type: LCS, Status: Ready
         Envelope queue size: 0
         Address: 0560
PCCA3 device LCS1: UnpackReads: NetType 2 AdapterNumber 0 BytesToMove 54
Arp adds translation 9.67.58.234 = IBMTR: 10005A250858
PCCA3 device LCS1: ToPcca3: BlockHeader copied from InputPosition: 0 56
PCCA3 device LCS1: ToPcca UnpackReads:  PacketsInInBlock = 1
PCCA3 device LCS1: CallSio: Starting I/O on device 0560. First command 02, UseDiag98 True
PCCA3 device LCS1: ToPcca PackWrites: Queuesizes, SavedEnv: 0 1 0
PCCA3 device LCS1: ToPcca PackWrites: LengthOfData, BlockHeader: 101 104
PCCA3 device LCS1: CallSio: Starting I/O on device 0561. First command 01, UseDiag98 True
```

Figure 66 (Part 2 of 2). A Sample of a PCCA Trace

The PCCA trace using the MORETRACE command provides the following additional information for Pccactrl fields:

- Command
- Return code
- Net numbers
- Adapter numbers
- Flags.

Hardware addresses, IP headers, ICMP headers, and ARP headers are also provided.

Figure 67 shows a sample of a PCCA trace using the MORETRACE command. The following information is shown.

- ACB 13715216 receives a PCCA control packet for the first adapter on a token-ring.
- The first command was 02 (read).
- ACB 13714696 is an ARP request from the local host to IP address 9.67.58.234.
- The CCW is 01 (write). For more information about CCW codes, see Table 22 on page 206.
- The last ACB is the ARP response from 9.67.58.234. It provides ARP packet information: hardware type (6), hardware addresses of both hosts, and IP addresses.

Information about LLC, such as the source SAP (AA), the destination SAP (AA), and protocol type (0806) is also shown.

```
PCCA3 device LCS1: ToPcca PackWrites: Queuesizes, SavedEnv: 0 0 0
ToPcca3: Acb Received:
13715216:
   Have completed I/O -> PCCA3 common routine (from PCCA3 handler)
   Last touched: 20
      IoDevice 0560
      Csw:
         Keys: E0, CcwAddress: 00559118
         Unit Status: 0C, Channel Status: 00
         Byte Count: 20402
      Device LCS1:
         Type: LCS, Status: Ready
         Envelope queue size: 0
         Address: 0560
PCCA3 device LCS1: Received PCCA control packet:
PccaCtrlCommand: 4, PccaCtrlNetType2: 2, PccaCtrlAdapter2: 0
PccaCtrlRetcode: 0, PccaCtrlSequence: 0, PccaCtrlFlags: 00
PccaCtrlHardwareAddress: 10005A6BB806
PCCA3 device LCS1: PCCA reports home hardware address 10005A6BB806 for link TR1
PCCA3 device LCS1: ToPcca3: BlockHeader copied from InputPosition: 0 76
PCCA3 device LCS1: ToPcca UnpackReads:  PacketsInInBlock = 1
PCCA3 device LCS1: CallSio: Starting I/O on device 0560. First command 02, UseDiag98 True
PCCA3 device LCS1: ToPcca3: Sio returned 0 on device 0560

PCCA3 device LCS1: ToPcca PackWrites: Queuesizes, SavedEnv: 0 0 0
 .
 .
 .
ToPcca3: Acb Received:
13714696:
```

Figure 67 (Part 1 of 3). A Sample of a PCCA Trace Using MORETRACE

```
   Send datagram -> Device driver(LCS1) (from UDP-request)
   Last touched: 23
      Device LCS1:
         Type: LCS, Status: Ready
         Envelope queue size: 1
         Address: 0560
PCCA3 device LCS1: ToPcca PackWrites: Queuesizes, SavedEnv: 0 1 0
PCCA3 device LCS1: Sending envelope to PCCA:
   Access control field: 60
   Frame control field: 40
   Token ring dest address: FFFFFFFFFFFF
   Token ring src address: 90005A6BB806
   Routing info: 8220
   Destination SAP: AA
   Source SAP: AA
   Control: 03
   Protocol id:000000
   Ethernet type: 0806
   ARP packet:
      ArpHardwareType: 6
      ArpProtocolType: 2048
      ArpHardwareLen: 6
      ArpProtocolLen: 4
      ArpOp: 0
      ArpSenderHardwareAddr: 10005A6BB806
      ArpSenderInternetAddr: 9.67.58.233
      ArpTargetHardwareAddr: C53400D7C530
      ArpTargetInternetAddr: 9.67.58.234
PCCA3 device LCS1: ToPcca PackWrites: LengthOfData, BlockHeader: 54 56
PCCA3 device LCS1: StartPccaOutputIo: OutputPosition is 56
PCCA3 device LCS1: CallSio: Starting I/O on device 0561. First command 01, UseDiag98 True
PCCA3 device LCS1: ToPcca3: Sio returned 0 on device 0561
 .
 .
 .
ToPcca3: Acb Received:
13714696:
   Have completed I/O -> PCCA3 common routine (from PCCA3 handler)
   Last touched: 23
      IoDevice 0560
      Csw:
         Keys: E0, CcwAddress: 00559118
         Unit Status: 0C, Channel Status: 00
         Byte Count: 20422
      Device LCS1:
         Type: LCS, Status: Ready
         Envelope queue size: 0
         Address: 0560
PCCA3 device LCS1: UnpackReads: NetType 2 AdapterNumber 0 BytesToMove 54
PCCA3 device LCS1: Received envelope from PCCA:
   Access control field: 18
   Frame control field: 40
   Token ring dest address: 10005A6BB806
   Token ring src address: 90005A250858
   Routing info: 02A0
   Destination SAP: AA
   Source SAP: AA
   Control: 03
   Protocol id:000000
   Ethernet type: 0806
```

Figure 67 (Part 2 of 3). A Sample of a PCCA Trace Using MORETRACE

```
   ARP packet:
      ArpHardwareType: 6
      ArpProtocolType: 2048
      ArpHardwareLen: 6
      ArpProtocolLen: 4
      ArpOp: 0
      ArpSenderHardwareAddr: 10005A250858
      ArpSenderInternetAddr: 9.67.58.234
      ArpTargetHardwareAddr: 10005A6BB806
      ArpTargetInternetAddr: 9.67.58.233
Arpin: Processing Arp packet:
   ArpHardwareType: 6
   ArpProtocolType: 2048
   ArpHardwareLen: 6
   ArpProtocolLen: 4
   ArpOp: 0
   ArpSenderHardwareAddr: 10005A250858
   ArpSenderInternetAddr: 9.67.58.234
   ArpTargetHardwareAddr: 10005A6BB806
   ArpTargetInternetAddr: 9.67.58.233
Arp adds translation 9.67.58.234 = IBMTR: 10005A250858
PCCA3 device LCS1: ToPcca3: BlockHeader copied from InputPosition: 0 56
PCCA3 device LCS1: ToPcca UnpackReads:  PacketsInInBlock = 1
PCCA3 device LCS1: CallSio: Starting I/O on device 0560. First command 02, UseDiag98 True
PCCA3 device LCS1: ToPcca3: Sio returned 0 on device 0560
```

Figure 67 (Part 3 of 3). A Sample of a PCCA Trace Using MORETRACE

RAWIP

The RAWIP group process combines RAWIPREQUEST and RAWIPUP traces.

TCP

The TCP group process combines TCP congestion control, notify, retransmit, round-trip, TCPDOWN, TCPREQUEST, and TCPUP traces.

TCPIP or TCP-IP

The TCPIP or TCP-IP group process combines TCP congestion control, IPDOWN, IPREQUEST, IPUP, notify, retransmit, round-trip, TCPDOWN, TCPREQUEST, and TCPUP traces.

UDP

The UDP group process combines UDPREQUEST and UDPUP traces.

Commonly Used Trace Options

The preceding sections have attempted to provide information and examples of the various types of traces that can be obtained for the TCP/IP virtual machine. The slightly more difficult task is to determine which trace options are complementary and which are the most beneficial or most expensive in terms of obtaining viable problem determination data. The table below provides a high-level overview of the most commonly used trace options, along with brief explanations of the type of events they generate and the “relative” cost of activating the trace option.

Table 10 (Page 1 of 6). Commonly-used Trace Options

Option name	TRACE output	Addl MORETRACE output
ARP	Maintenance of queue of packets waiting for ARP response. Errors in ARP processing. *No* output caused by received ARP broadcasts.	All received ARP packets. Can generate a lot of output if much broadcast ARP traffic on network.
CETI	Internal workings of CETI drivers (ELANS, ILANS, X.25 ICA). I/O interrupts. Length of packets received and sent. Device initialization. Can generate a lot of output if there's a lot of broadcast traffic on the network, even if little activity is occurring locally on the host.	Hex dump of received and transmitted packets, and ARP processing info. Main disadvantage of MORETRACE is that large packets increase the size of the trace. This is mainly a concern with FTP and other bulk data transfers. But the packet trace can be invaluable.
CLAW	Information about CLAW read and write channel program processing. Start I/O and write complete notifications. CSW information on I/O completions. Data from Sense ID channel command execution. Statistical information about packets. ACB information.	MORETRACE CLAW output adds envelope and CLAW control packet information, IP datagram information, and read / write channel program information when I/O is started.
CONGESTION	Traces some aspects of TCP-layer “congestion-control.” Usable as part of TCP or TCPIP tracing; not useful by itself.	No additional tracing
CONSISTENCYCHECKER	Every 5 minutes, print various queue sizes. Useful to determine free pool status in Version 1.	More detail. MORETRACE doesn't cost much more than TRACE, since output is only every 5 minutes.
HCH	Hyperchannel device driver message headers, some return codes	Queue sizes, packet sizes, I/O interrupts. If Hyperchannel tracing is needed, then MORETRACE is worthwhile. That is, TRACE alone isn't too useful.
ICMP	Received ICMP packets	Additional information on Redirect packets
IPDOWN	Errors in ICMP packet generation. Redirect processing. Fragmentation of outbound packets. Routing of outbound packets.	IP headers of outbound packets and fragments.

Table 10 (Page 2 of 6). Commonly-used Trace Options

Option name	TRACE output	Addl MORETRACE output
IPUP	Internal IPUP activity information, Reassembly of fragments, Bad received checksums, Information on received datagrams, IP option errors, and Packet forwarding.	Additional details on reassembly and redirect. IP headers of packets other than TCP protocol.
Note: If IP tracing is required, it is almost always worthwhile to trace IPUP and IPDOWN together. In two sample traces of the same traffic, MORETRACE IPUP IPDOWN generated 2.5 times as many lines of output as TRACE IPUP IPDOWN, mainly because of the multiple-line tracing of outbound IP headers generated by MORETRACE IPDOWN. TRACE IPUP output includes datagram id's of incoming packets, useful for correlating with network monitor tracing. MORETRACE IPDOWN must be used to get datagram id's of outgoing packets.		
IUCV	IUCV driver (PVMIUCV, SNAIUCV, IUCV, X25NPSI devices) details, including path establishment	No additional tracing
IUCVSIGNON	IUCV driver, path establishment only	No additional tracing
NOTIFY	Tracing related to sending of notifications to the internal client (Telnet server) and VMCF clients (Pascal interface and direct VMCF interface). In addition, events involving IUCV clients (socket interface and direct IUCV interface) are processed through TCNOTIF PASCAL, so they will show up here too, even though no VMCF message is actually sent.	Additional details. In two sample traces of the same traffic, MORETRACE NOTIFY generated twice as many lines of output as TRACE NOTIFY. If notifications are suspected to be a problem, the extra output is worthwhile.
PCCA	LCS driver packet sizes, block headers, I/O interrupts. Can generate a lot of output if there is a lot of broadcast traffic on the network, even if little activity is occurring locally on the host.	Packet headers, SIO return codes
PING	Traces ping requests and responses generated through the PING command or directly by the PingRequest Pascal call or PINGreq VMCF call.	No additional tracing

Table 10 (Page 3 of 6). Commonly-used Trace Options

Option name	TRACE output	Addl MORETRACE output
RAWIPREQUEST	Traces requests using Raw IP through the Pascal interface or VMCF interface. Raw IP routines include • RawIpOpen (OPENrawip) • RawIpClose (CLOSErawip) • RawIpSend (SENDrawip) • RawIpReceive (RECEIVErawip)	IP packet headers as supplied by application, before they are completed by the FillIpHeader routine. In two sample traces of the same traffic, MORETRACE RAWIPREQUEST generated 1.6 times as many lines of output as TRACE RAWIPREQUEST. The extra output is worthwhile.
RAWIPUP	Messages pertaining to queuing received IP packets for applications using Raw IP interface or raw sockets.	No additional tracing
Note: NOTIFY is also useful for looking at raw IP activity, since it traces RAWIPpacketsDELIVERED notifications.		
RETRANSMIT, REXMIT	Retransmissions by local TCP. Duplicate packets received, indicating possibly unnecessary retransmission by foreign TCP.	No additional tracing
ROUNDTRIP	“Round-trip” times, i.e. time between sending TCP packet and receiving acknowledgment. Not very useful by itself.	No additional tracing
SCHEDULER	Lists the internal TCPIP processes as they are called. Listing is one per line.	Much more detail on why each process is called. MORETRACE SCHEDULER is gives a good overall view of what is happening in TCPIP; quite useful as a debugging tool.
SNMPDPI	SNMP“sub-agent” tracing. Lists MIB queries by the SNMP agent.	No additional tracing
SOCKET	Trace requests made through IUCV socket interface, and most responses.	A little extra tracing in bind() processing
TCP	Includes TCPREQUEST, TCPDOWN, TCPUP, ROUNDTRIP, NOTIFY, REXMIT, and CONGESTION.	See individual entries. MORETRACE TCP sets detailed tracing for all the above names.

Table 10 (Page 4 of 6). Commonly-used Trace Options

Option name	TRACE output	Addl MORETRACE output
TCPDOWN	Trace information related to outbound TCP packets, both data packets and acknowledgments.	More verbose listing, can be twice as long as TRACE TCPDOWN. Much of the extra output is redundant and verbose, and is not worthwhile, especially if a large data transfer is to be traced.
TCPIP, TCP-IP	Includes TCPREQUEST, TCPDOWN, TCPUP, ROUNDTRIP, NOTIFY, REXMIT, CONGESTION, IPDOWN, and IPUP	See individual entries. MORETRACE TCPIP sets detailed tracing for all the above names.
TCPREQUEST	Information pertaining to execution of the following Pascal-interface and VMCF-interface requests: • TcpAbort (ABORTtcp) • TcpClose (CLOSEtcp) • TcpOpen and TcpWaitOpen (OPENtcp) • TcpSend (SENDtcp) • TcpReceive (RECEIVEtcp) • TcpStatus (STATUStcp) • TcpFReceive and TcpWaitReceive (FRECEIVEtcp) • TcpFSend and TcpWaitSend (FSENDtcp) • BeginTcpIp (BEGINtcpIPservice) • EndTcpIp (ENDtcpIPservice) • Handle (HANDLEnotice) • IsLocalAddress (IShostLOCAL) Also traces requests produced by the Version 1 socket interface module, CMSOCKET C, for stream sockets and initialization.	In two sample traces of the same traffic, MORETRACE TCPREQUEST generated 1.5 times as many lines of output as TRACE TCPREQUEST. But the extra detail, including information on open calls, and compact display of TCB's, is worthwhile.

Table 10 (Page 5 of 6). Commonly-used Trace Options

Option name	TRACE output	Addl MORETRACE output
TCPUP	Information related to processing of incoming TCP packets.	In two sample traces of the same traffic, MORETRACE TCPUP generated *14* times as many lines of output as TRACE TCPUP. This extra volume makes a huge difference when tracing a large data transfer. So MORETRACE TCPUP is probably unnecessary in the first stage of gathering trace information.
TELNET	The Telnet server is a TCP/IP application program that, unlike other applications, runs as a process in the TCPIP virtual machine (the “internal client”) instead of in its own virtual machine. So tracing of the Telnet server application is enabled via the TRACE and MORETRACE commands used in the rest of TCPIP.	MORETRACE output adds tracing of data, including: Data accepted from logical devices, data presented to logical devices, data sent to TCP, data received from TCP, data sent to *CCS, data received from *CCS. Some data is printed one byte per line, which greatly increases the number of lines of trace output, though not necessarily the space occupied on disk or tape. For most Telnet server problems, MORETRACE TELNET is probably a good choice.
TIMER	Information related to internal timeout processing within TCPIP. Probably useful only for debugging internal problems.	If a timer problem is suspected, then MORETRACE TIMER output would be useful to a person familiar with TCPIP internals. Output may be *12* times as large as TRACE.
UDPREQUEST	Information pertaining to execution of the following Pascal-interface and VMCF-interface requests: • UdpClose (CLOSEudp) • UdpOpen (OPENudp) • UdpSend (SENDudp) • UdpNReceive (NRECEIVEtcp) • UdpReceive (RECEIVEudp) Also traces requests produced by the Version 1 socket interface module, CMSOCKET C, for datagram sockets.	In two sample traces of the same traffic, MORETRACE UDPREQUEST generated 2.5 times as many lines of output as TRACE UDPREQUEST. But the extra detail, including display of UCB's, is worthwhile.

Table 10 (Page 6 of 6). Commonly-used Trace Options

Option name	TRACE output	Addl MORETRACE output
UDPUP	Information about processing of inbound UDP packets. Useless without MORETRACE.	Port number in following message is wrong: UptoUDP: Destination port # 65536108 The port number is only the *high-order halfword*. 65536108 = X'03E8006C', so port number is X'3E8' = 1000.

Note: NOTIFY is also useful for looking at UDP activity, since it traces UDPdatagramDELIVERED notifications.

Connection State

A connection state is a description of the status of a logical communication path between two "sockets."[1] The terms used to describe this status vary according to the perspective from which the connection state is viewed. The following sections discuss the connection state as seen from the perspectives of the TCP layer, Pascal or VMCF applications, and socket applications.

Connection State As Known by TCP

The TCP layer in the host at each end of a TCP connection keeps its own variable containing the state of the connection, using the connection states defined in RFC 793. This is the state shown in NETSTAT output.

Ignoring state transitions, which do not tend to conform to these simplistic definitions, the following table lists the connection states and what each typically implies about the state of the connection. See section 3.2 of RFC 793 for more information on connection states.

Table 11 (Page 1 of 3). TCP Connection States

State name	Typical Situation
LISTEN	Waiting for a connection request from the address and port listed in the Foreign Socket column of NETSTAT. • "HOSTA..*" means waiting for a connection request from any port on host HOSTA. • "*..100" means waiting for a connection request from port 100 on any host. • "*..*" means waiting for a connection request from any port on any host. If the application uses the Pascal interface or VMCF interface, it has done a TcpOpen (or TcpWaitOpen) with an initial pseudo-state of LISTENING. If the application uses the socket interface, from C or via IUCV, it has done a listen(), and the listen backlog has not been reached.

1 In this context, a socket is merely an address which specifically includes a port identifier; that is, the concatenation of an internet address with a TCP port.

Table 11 (Page 2 of 3). TCP Connection States

State name	**Typical Situation**
SYN-SENT	The application has done an "active open" and is waiting for a response from the foreign server. If the application uses the Pascal interface or VMCF interface, it has done a TcpOpen (or TcpWaitOpen) with an initial pseudo-state of TRYINGtoOPEN. If the application uses the socket interface, from C or via IUCV, it has done a connect().
SYN-RECEIVED	Represents a condition where TCP is waiting for a confirming connection request acknowledgement after having received and sent a connection request. This sometimes means that a SYN was received on a connection in LISTEN state, but connection establishment hasn't been able to proceed further because a routing problem prevents the response from reaching the foreign host.
ESTABLISHED	Connection is completely established. Both sides can send and receive data. This is the normal state for the data transfer phase of a connection.
FIN-WAIT-1	Application has issued a TcpClose or close(). A FIN packet was sent but not acknowledged, and a FIN hasn't been received from the foreign host.
FIN-WAIT-2	Application has issued a TcpClose or close(). FIN packet was sent and has been acknowledged. TCP is now waiting for the foreign host to send a FIN. This is the state a connection enters when the application closes but the application on the other end doesn't close. There is no timeout in this state, since the FIN has been acknowledged. If the foreign host sends an ACK packet in response to the the local host's FIN and then goes away without sending an RST, or if the RST is lost, then the connection will stay in this state for an indefinite period of time (until the application aborts the connection or terminates). In this state, data can be received but not sent. Some applications may intentionally put the connection into this state because they plan to send data in one direction. However, in most cases, this is not a long-term state. Usually, persistence of this state indicates an error condition.
CLOSE-WAIT	The local host has received a FIN from the foreign host and has acknowledged it, but the application hasn't issued a TcpClose or close(). In this state, data can be sent but not received. Some applications may intentionally put the connection in this state because they plan to send data in one direction. However, in most cases, this is not a long-term state. Usually, persistence of this state indicates an error condition.
CLOSING	Represents waiting for a connection termination request acknowledgement from the remote TCP. This state (and the LAST-ACK state) indicates that both sides have closed the connection. Data cannot be sent in either direction.

Table 11 (Page 3 of 3). TCP Connection States

State name	Typical Situation
LAST-ACK	Represents waiting for an acknowledgement of the connection termination request previously sent to the remote TCP (which included an acknowledgement of the remote TCP's connection termination request). This state (and the CLOSING state) indicates that both sides have closed the connection. Data cannot be sent in either direction.
TIME-WAIT	Both sides have closed the connection, and all packets have been acknowledged. The connection stays in this state for 2 * MSL (MSL = 60 seconds) as required by the protocol specification, to ensure that foreign host has received the acknowledgment of its FIN. In VM TCP/IP, connections in TIME-WAIT state do not usually appear in the output from the NETSTAT command. The ALLCONN or TELNET parameters must be supplied on the NETSTAT command to see connections in this state.
CLOSED	The connection is completely closed. In TCP/IP for VM, connections in CLOSED state do not usually appear in the output from the NETSTAT command. The ALLCONN parameter must be supplied on the NETSTAT command to see connections in this state.

Connection State As Known by Pascal or VMCF Applications

Pascal and direct VMCF applications do not see the actual TCP states described in Table 11. Rather, the connection state in the StatusInfoType record and in CONNECTIONstateCHANGED notifications is expressed as a “pseudo-state.” The pseudo-state contains the connection state information needed by an application program, while hiding protocol details that are not important to an application.

Table 12 (Page 1 of 2). Connection Pseudo-states

State name	Meaning, from CMCOMM COPY	Corresponding TCP states
LISTENING	Waiting for a foreign site to open a connection	LISTEN
TRYINGtoOPEN	Trying to contact a foreign site to establish a connection.	SYN-SENT, SYN-RECEIVED
OPEN	Data can go either way on the connection	Either: • ESTABLISHED • CLOSE-WAIT, but input data still queued for application
SENDINGonly	Data can be sent out but not received on this connection. This means that the foreign site has done a one-way close.	CLOSE-WAIT, and no input data queued for application

Table 12 (Page 2 of 2). Connection Pseudo-states

State name	Meaning, from CMCOMM COPY	Corresponding TCP states
RECEIVINGonly	Data can be received but not sent on this connection. This means that the client has done a one-way close.	Either: • FIN-WAIT-1 • FIN-WAIT-2 • LAST-ACK, but input data still queued for application • CLOSING, but input data still queued for application • TIME-WAIT, but input data still queued for application
CONNECTIONclosing	Data may no longer be transmitted on this connection since the TCP/IP service is in the process of closing down the connection.	Either: • LAST-ACK, and no input data queued for application • CLOSING, and no input data queued for application • TIME-WAIT, and no input data queued for application
NONEXISTENT	The connection no longer exists.	CLOSED

Connection State As Known by Socket Applications

The socket interface does not allow for programs to see explicit connection states. The connection state is inferred from the response to various socket calls.

- A successful return from connect() means that the connection is in an OPEN pseudo-state. The socket returned from a successful accept() call is also assumed to be in an OPEN pseudo-state.
- A return code of 0 from read(), recv(), etc., indicates that foreign host has done one-way close. This is like SENDINGonly pseudo-state.
- A return code of -1 from read(), recv(), etc., with an *errno* value of ECONNABORTED, ECONNRESET, or ETIMEDOUT, indicates that the connection has been abruptly closed (reset) for the given reason.

Note that internal TCP/IP traces show CONNECTIONstateCHANGED notifications being sent to socket programs. In fact, the notification is converted to the proper socket state information so that the program may find out about the state change on its next socket call.

Traceroute Function (TRACERTE)

The Traceroute function sends UDP requests with varying Time-to-Lives (TTL) and listens for TTL-exceeded messages from the routers between the local host and the foreign host. Traceroute uses RAW sockets, so you must have OBEYFILE authority to use this command. The range of port numbers that Traceroute uses are normally invalid, but you can change it if the target host is using a nonstandard UDP port.

To debug network problems, use the TRACERTE command. See the TCP/IP User's Guide for a complete format of the TRACERTE command.

```
►►──TRACERTE──┬─?─────────┬────────────────────────────────────────►◄
              └─host_name─┘
```

The following are examples of using the TRACERTE command:

```
tracerte cyst.watson.ibm.com
Trace route to CYST.WATSON.IBM.COM (9.2.91.34)
1 (9.67.22.2) 67 ms 53 ms 60 ms
2 * * *
3 (9.67.1.5) 119 ms 83 ms 65 ms
4 (9.3.8.14) 77 ms 80 ms 87 ms
5 (9.158.1.1) 94 ms 89 ms 85 ms
6 (9.31.3.1) 189 ms 197 ms *
7 * * (9.31.16.2) 954 ms
8 (129.34.31.33) 164 ms 181 ms 216 ms
9 (9.2.95.1) 198 ms 182 ms 178 ms
10 (9.2.91.34) 178 ms 187 ms *
> Note that the second hop does not send Time-to-live exceeded
> messages. Also, we occasionally lose a packet (hops 6,7, and 10).

Ready;
tracerte 129.35.130.09
Trace route to 129.35.130.09 (129.35.130.9)
1 (9.67.22.2) 61 ms 62 ms 56 ms
2 * * *
3 (9.67.1.5) 74 ms 73 ms 80 ms
4 (9.3.8.1) 182 ms 200 ms 184 ms
5 (129.35.208.2) 170 ms 167 ms 163 ms
6 * (129.35.208.2) 192 ms !H 157 ms !H
> The network was found, but no host was found

tracerte 129.45.45.45
Trace route to 129.45.45.45 (129.45.45.45)
1 (9.67.22.2) 320 ms 56 ms 71 ms
2 * * *
3 (9.67.1.5) 67 ms 64 ms 65 ms
4 (9.67.1.5) 171 ms !N 68 ms !N 61 ms !N
> Could not route to that network.
```

Traceroute uses the site tables for inverse name resolution rather than the domain name server. If a host name is found in the site table, it is printed along with its IP address.

```
tracerte EVANS
Trace route to EVANS (129.45.45.45)
1 BART (9.67.60.85) 20 ms 56 ms 71 ms
2 BUZZ (9.67.60.84) 55 ms 56 ms 54 ms
3 EVANS (9.67.30.25) 67 ms 64 ms 65 ms
```

Chapter 8. FTP Traces

This chapter describes File Transfer Protocol (FTP) traces, including the relationship between FTP user and server functions. This chapter also describes how to activate and interpret FTP client and server traces.

FTP Connection

A control connection is initiated by the user-Protocol Interpreter (PI) following the Telnet protocol (x) and the server-Protocol Interpreter (PI) response to the standard FTP commands. Figure 68 shows the relationship between user and server functions.

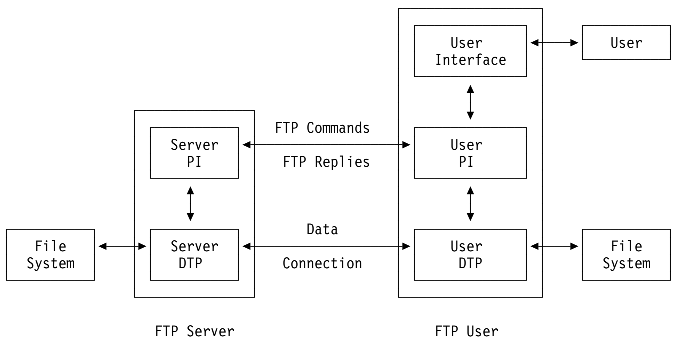


Figure 68. The FTP Model

Note: PI is the Protocol Interpreter and DTP is the Data Transfer Process. The data connection can be used in either direction and it does not have to be active.

Once the parameters from the data connections have been transmitted, the user-DTP must be in listen status on the specified data port. The server initiates the data connection using the default data port requested by the user. For VM FTP implementations, the client issues a PORT command. The port is then assigned by TCPIP after an open request. The format of the PORT command is:

```
►►──PORT──9.67.58.127──p1──p2──────────────────────────────────►◄
```

The only parameter for the PORT command is:

Parameter	Description
9.67.58.127,*p1*,*p2*	Is the address space for the default data port.

The server initiates, maintains, and closes the data connection. However, when a user transmits data, an end of file (EOF) closes the data connection.

FTP Client Traces

The following sections describe how to activate FTP client traces and interpret the output.

Activating Traces

FTP client traces are activated by specifying the **TRACE** parameter in addition to the usual processing parameters on invocation of the FTP command. Tracing can also be activated interactively once an FTP session has been established by using the DEBUG subcommand of FTP. The following is the format for the FTP TRACE command in VM:

```
►►──FTP──foreignhost──┬────────────┬──TRACE──────────────────────────►◄
                      └─portnumber─┘
```

The parameters for the FTP command are:

Parameter	Description
foreignhost	Specifies the name of the foreign host to which you are connecting. The host may be specified by its host name or internet address.
portnumber	Specifies the number of the port to request connection to. This parameter is usually used for system testing only.
EXIT	Terminates FTP when an error occurs, returning an error code.
TRACE	Starts the generation of tracing output. TRACE is used to assist in debugging.
TRANSLATE *filename*	Specifies the name of a translation table file other than a standard table.

To enable or disable the trace mode interactively, use the DEBUG subcommand of FTP. The format of the DEBUG subcommand is:

```
►►──DEBUG────────────────────────────────────────────────────────────►◄
```

The DEBUG subcommand has no parameters.

Trace output is directed to the virtual machine console.

For more information about the FTP command and DEBUG subcommand, see the *TCP/IP User's Guide*.

Trace Output

The output from FTP traces shows the sequence of commands requested by the TCP/IP user. Transferred data is not traced.

You can relate FTP client and server traces if the connection has been interrupted or closed at the client's request or initiated by the server. TCP requests that are traced by the client program include:

- TcpOpen
- BeginTcpIp
- TcpWaitReceive
- TcpWaitSend.

The messages issued by FTP are referenced in RFC 959. The first five significant digit values for FTP return codes are:

1yz	Positive preliminary reply
2yz	Positive completion reply
3yz	Positive intermediate reply
4yz	Transient negative completion reply
5yz	Permanent negative completion reply.

Figure 69 shows a sample of an FTP client trace. In the trace, input from the keyboard or a file is preceded by:

===

Information that the FTP client is sending over the control connection is preceded by:

>>>

Action taken by the FTP client program is preceded by:

==>

The other statements in the trace flow are self-explanatory and can be found in the source code of the FTP modules.

```
FTP 9.67.43.126 TRACE
VM TCP/IP FTP V2R4
about to call BeginTcpIp
Connecting to 9.67.43.126, port 21
SysAct 0 21 155396990 CC -1
==> Active open to host 9.67.43.126 port 21
from host 0 port  65535
In SysRead, calling TcpWaitReceive with args: 0 00035BD4
65535
In SysRead, TcpWaitReceive returned: 131
220-FTPSERVE at HOSTVM.ENDICOTT.IBM.COM, 08:56:14 EDT TUESDAY 10/02/97
220 Connection will close if idle for more than 5 minutes.
GetReply returns 220
USER (identify yourself to the host):
===tcpusrx
>>>USER tcpusrx
In SysSendFlush, calling TcpWaitSend with args: 0 00034260
14
In SysSendFlush, TcpWaitSend returned: OK
In SysRead, calling TcpWaitReceive with args: 0 00035BD4 65535
In SysRead, TcpWaitReceive returned: 27
331 Send password, please
GetReply returns 331
Password:
===________ (non-display entry)
```

Figure 69 (Part 1 of 2). A Sample of an FTP Client Trace

```
>>>PASS ********
In SysSendFlush, calling TcpWaitSend with args: 0 00034260 13
In SysSendFlush, TcpWaitSend returned: OK
In SysRead, calling TcpWaitReceive with args: 0 00035BD4 65535
In SysRead, TcpWaitReceive returned: 56
230 SYLVAIN logged in; working directory = SYLVAIN 191
GetReplCodeText returns 230 230 SYLVAIN logged in; working directory = SYLVAIN
191
leaving dologin
Command:
===get example.fileone
Filename: "$FTCOPY$.FTPUT1.A"
==> Passive open
Passive open successful: Fd = 3, TcpId = 1
>>>PORT 9,67,58,226,4,72
In SysSendFlush, calling TcpWaitSend with args: 0 00034260 23
In SysSendFlush, TcpWaitSend returned: OK
In SysRead, calling TcpWaitReceive with args: 0 00035BD4 65535
In SysRead, TcpWaitReceive returned: 21
200 Port request OK
GetReply returns 200
>>>RETR example.fileone
In SysSendFlush, calling TcpWaitSend with args: 0 00034260 22
In SysSendFlush, TcpWaitSend returned: OK
In SysRead, calling TcpWaitReceive with args: 0 00035BD4 65535
In SysRead, TcpWaitReceive returned: 36
150 Sending file 'example.fileone'
GetReplCodeText returns 150 150 Sending file 'example.fileone'
In UntilOpen: Note received:
=>TcpId 1 Connection state changed Trying to open
In UntilOpen: Note received:
 => TcpId 1 Connection state changed Open
Transferring in AsciiToRecord
In GetFromTcp, calling TcpWaitReceive with args: 1002BF000 32768
In GetFromTcp, TcpWaitReceive returned: 1072
GetFromTcp: 1072 bytes in buffer
In GetFromTcp, calling TcpWaitReceive with args: 1002BF000 32768
In GetFromTcp, TcpWaitReceive returned: -35
Sysclose called with fd = 3
In SysClose: Note received:  => TcpId 1 Connection state changed Sending only
In SysClose: Note received:  => TcpId 1 Connection state changed Connection
closing
Exiting from SysClose: fd = 3, TcpId = 1
In SysRead, calling TcpWaitReceive with args: 0 00035BD4 65535
In SysRead, TcpWaitReceive returned: 37
250 Transfer completed successfully
GetReply returns 250
1072 bytes transferred. Transfer rate 2.17 Kbytes/sec.
Command:
===quit
>>>QUIT
In SysSendFlush, calling TcpWaitSend with args: 0 00034260 6
In SysSendFlush, TcpWaitSend returned: OK
In SysRead, calling TcpWaitReceive with args: 0 00035BD4 65535
In SysRead, TcpWaitReceive returned: 37
221 Quit command received.  Goodbye
GetReply returns 221
Entering WaitAndClose
In WaitAndClose: Note received:  => TcpId 1 Connection state changed
   Nonexistent
In WaitAndClose: Note received:  => TcpId 0 Connection state changed
Sending only
```

Figure 69 (Part 2 of 2). A Sample of an FTP Client Trace

The following describes the sequence of major events in the FTP client trace sample output:

1. The connection to the remote host is opened through the FTP server's listen port.

Trace Item	Description
9.67.43.126	Address space of the remote host server.
21	Port 21, which is used for FTP connections.
0	Local host number.
65535	UNSPECIFIEDport, which is used to request an available port from TCPIP with TcpOpen functions.

2. Data is received from the remote server.

Trace Item	Description
In SysRead	Name of the FTP client procedure.
TcpWaitReceive	Name of a TCPIP client procedure.
0	ID of the connection between the TCPIP and FTP client.
00035BD4	Buffer address that contains the data or text to be sent by the remote host.
65535	Buffer size authorized by the client.
131	Length of the data received, plus carriage returns and line feeds (CR/LF) for: • Outbound connections (preceding the text line of output) • Inbound connections (following the text line of output); if this number is negative, it is a return code.

3. SysSendFlush, an FTP client procedure, flushes buffered output and adds CR/LFs.

Trace Item	Description
TcpWaitSend	Name of the TCP/IP function called.
0	Name of the control connection ID.
00034260	Buffer address of the data or command sent to the remote host.
13	Length of the command sent plus CR/LF.

4. FTP performs a passive open to the remote host FTP server.

Trace Item	Description
Fd=3	Internal connection slot number in the FTP client program; the Fd for the first control connection is 1.
TcpId	Connection ID between the TCPIP and FTP client.
9,67,58,226,4,72	Host address space, 9.67.58.226, and port number, 1096. The port number is obtained by converting 4 and 72 to hexadecimal (`X'04'` and `X'48'`), and then converting `X'0448'` to a decimal.

5. The trace shows the evolving status of the data connection (TcpId 1) while in routine UntilOpen. The purpose of the UntilOpen routine is to wait for the server to complete the data connection after the open has been requested from TCPIP. The status of the connection changes from:

`TryingToOpen`

to

`Open.`

6. A return code (-35) signifies that the remote host is closing the connection.
7. The status of the data connection being closed by TCPIP is displayed.

FTP Server Traces

The following sections describe how to activate FTP server traces and interpret the output.

Activating Traces

Activation of the tracing facilities within the FTP server is accomplished by specifying a trace parameter at server initialization, or by using the FTP Server SMSG interface to issue an SMSG TRACE ON command. For information about the SMSG TRACE command, refer to the 'Configuring the FTP Virtual Machine' chapter in the *TCP/IP Planning and Customization* The method of specifying the parameters varies according to operating system environment.

In the VM environment, the FTP server is activated during processing performed in the server virtual machine when its PROFILE EXEC executes the SRVRFTP command. Tracing is activated by specifying the **TRACE** parameter in addition to the usual processing parameters on command invocation. Figure 70 demonstrates the use of the **TRACE** parameter for the SRVRFTP command:

```
              ┌─PORT 21──────────┐  ┌─INACTIVE 300────────────┐
►►──SRVRFTP───┼──────────────────┼──┼─────────────────────────┼──┬───────┬──┬──────┬──────►
              └─PORT port_number─┘  └─INACTIVE number_seconds─┘  └─TRACE─┘  └─RACF─┘

►──┬──────────┬───────────────────────────────────────────────────────────────────────────►◄
   └─ANONYMOU─┘
```

Figure 70. SRVRFTP Command Invocation Parameters

The processing parameters that can be supplied are similar to those shown in Figure 70.

Note that Figure 70 is only intended to highlight the specification of the parameter necessary to activate tracing. Refer to the *TCP/IP Planning and Customization* for information on the usage of the other parameters.

The TRACE option of FTP server provides three different types of information:

Console Output
: Console output is standard for normal operations. The trace option adds information about the general FTP server operations. For example, LINK operations with return codes are provided. Console output is obtained in the following format:

  VM — Standard console output.

Log
: The log gives information about abnormal run-time situations, such as broken connections with TCPIP or with the remote client and remote port that is unavailable. The following describes how to obtain the log:

	VM `FTPSERVE LOG A` file.
Debug File	The debug file provides complete information about internal FTP server activities. The following describes how to obtain the debug file:
	VM `FILE DEBUGTRA A` file.

Trace Output

Tracing the internal operations of the FTP server provides information about the processes, ports, and connections. The complete text of messages sent to clients, operations, and the status of the data and control connections are also documented.

Figure 71 shows a sample of an FTP Server Trace.

```
OpenConnection(00000000,21,00000000,65535,2147483647,FALSE
AdvertizeService gets connection #0
Got note Connection state changed for #0, Trying to open
OpenConnection(00000000,21,00000000,65535,2147483647,FALSE
AdvertizeService gets connection #1
Got note Connection state changed for #0, Open
Allocating buffer of 32768 bytes
Allocating RdFromDiskBuf of 16384 bytes
Send reply '220-FTPSERVE at ENDVM23.TCPIPDEV.ENDICOTT.IBM, 18:22:54 EST THURSDAY
10/04/97'
Send reply '220 Connection will close if idle for more than 5 minutes.'
ReinitContConn(0)
GetData(0)
In GetData, TcpFReceive: Where = 1
Got note Data delivered for #0, 14 bytes
14 bytes arrived on conn #0
Send reply '230 TCPUSR1 logged in with no special access privileges'
GetData(0)
In GetData, TcpFReceive: Where = 1
Got note Data delivered for #0, 13 bytes
13 bytes arrived on conn #0
Send reply '332 Supply minidisk password using 'account''
GetData(0)
In GetData, TcpFReceive: Where = 1
Got note Data delivered for #0, 11 bytes
11 bytes arrived on conn #0
Send reply '230 Working directory is TCPUSR1 191 (ReadOnly)'
GetData(0)
In GetData, TcpFReceive: Where = 1
Got note Data delivered for #0, 22 bytes
22 bytes arrived on conn #0
Send reply '200 Port request OK.'
GetData(0)
In GetData, TcpFReceive: Where = 1
Got note Data delivered for #0, 19 bytes
19 bytes arrived on conn #0
```

Figure 71 (Part 1 of 2). A Sample of an FTP Server Trace

```
OpenConnection(09433AE9,20,09433AE9,1029,30,TRUE
Send reply '125 List started OK'
SOpenfscb: name is: CONN-2.FTPLIST.A
SOpenFscb: ESTATE returns: 0
TidyFile: FINIS returns -450887680
GetData(0)
In GetData, TcpFReceive: Where = 1
Got note Connection state changed for #2, Open
Allocating buffer of 32768 bytes
Allocating RdFromDiskBuf of 16384 bytes
Data connection 2 open for sending
ReinitDataConn(2)
FtpFormat: A FtpMode: S FtpOptFormat: 0
RecordFormat: V RecordLength: 65535
StartTransfer for 2:
Xfread: totalread = 79 Result = 0 FByte = 1 LByte = 0
79 bytes sent on connection 2
Got note FSend response for #2, SendTurnCode = 0
Xfread: totalread = 0 Result = -12 FByte = 80 LByte = 79
Calling CMS(ERASE CONN-2 FTPLIST A)
Closing connection #2
Completed CloseConnection
Got note Connection state changed for #2, Receiving only
Got note Connection state changed for #2, Connection closing
CloseCompleted on #2: OK
Send reply '250 List completed successfully.'
DataReply: Setting CmdInProgress to CUNKNOWN on conn #2, was SYST
Got note Other external interrupt received for #-48, RuptCode = 64
Aborting connection #0
CloseCompleted on #0: Software error in TCP/IP!
Error completion on #0: Software error in TCP/IP!
```

Figure 71 (Part 2 of 2). A Sample of an FTP Server Trace

Chapter 9. Simple Mail Transfer Protocol Traces

This chapter describes how to activate and interpret Simple Mail Transfer Protocol (SMTP) traces.

SMTP Client Traces

The client interface to SMTP is in the form of some type of electronic mailing handling program. There is no formal command interface. The mailing programs (procedures) communicate with the IBM TCP/IP implementation of SMTP. The client programming interfaces that are available for use with the TCP/IP Feature for VM/ESA are the CMS SENDFILE and NOTE commands.

Activating Traces

Trace activation in the client environment is dependent on the type of mail handling facilities made available at an installation. The client interfaces provided with the TCP/IP product are in the form of a REXX EXEC procedures for VM.

The NOTE and SENDFILE EXEC procedures are written in the REXX procedures language, so various levels of traces are available for use. Refer to the applicable level of the *System Product Interpreter Reference* publication for more information. The results of any chosen trace level will be directed to the user's console.

Obtaining Queue Information

Clients can obtain information about mail that SMTP is delivering or waiting to deliver. While this facility is not considered to be a formal diagnostic aid, it can be used in situations where it is felt that an inordinate delay in mail delivery is occurring to determine if further investigation is warranted.

The SMTPQUEU command is used to obtain the queue information. It causes the SMTP virtual machine to deliver a piece of mail that lists the mail queued for delivery at each site. The mail is spooled to the user that issued the SMPTQUEU command. Figure 72 on page 135 shows the format of the output returned by the SMTP server.

```
220-ENDVMM.ENDICOTT.IBM.COM running IBM VM SMTP Level nnn on Fri, 26 Jul 97 09:55:05 E
220 DT
050 VERB ON
250 Verbose Mode On
050 QUEU
250-Queues on ENDVMM.ENDICOTT.IBM.COM at 09:55:05 EDT on 07/26/97
250-Spool Queue: Empty
250-Undeliverable Queue: Empty
250-Resolution Queues:
250-Resolver Process Queue: Empty
250-Resolver Send Queue: Empty
250-Resolver Wait Queue: Empty
250-Resolver Retry Queue: Empty
250-Resolver Completed Queue: Empty
250-Resolver Error Pending Queue: Empty
250 OK
```

Figure 72. Sample Outout form a Mail Queue Query

SMTP Server Traces

The following sections describe how to activate and interpret SMTP server traces. In order to help with interpreting trace output, a list of the SMTP commands that can appear in the trace data along with descriptions of these commands is supplied below. The SMTP server provides the interface between the internet and IBM host systems. For more information about the SMTP protocol, see RFC 821.

Activating Traces

SMTP server traces can be activated by including a TRACE statement in the SMTP CONFIG file, or by using the SMSG interface to the SMTP machine to issue an SMSG TRACE command. For information on the syntax of the TRACE statement or the SMSG TRACE command as well as information on what types of traces are available, refer to the SMTP chapter in the VM TCP/IP Planning and Customization manual. Sample trace data for several of the available trace commands is provided at the end of this chapter.

SMTP Commands

SMTP commands define the mail transfer or the mail system function requested by the user. The commands are character strings terminated by the carriage return and line feed characters (CR/LF). The SMTP command codes are alphabetic characters. These characters are separated by a space if parameters follow the command or a CR/LF if there are no parameters.

Table 13 on page 136 describes the SMTP commands that are helpful when interpreting SMTP trace output.

Table 13 (Page 1 of 2). SMTP Commands

Name	Command	Description
DATA	DATA	The receiver treats the lines following the DATA command as mail data from the sender. This command causes the mail data that is transferred to be appended to the mail data buffer. The mail data can contain any of the 128 ASCII character codes. The mail data is terminated by a line containing only a period, that is the character sequence CR/LF CR/LF.
EXTENDED HELLO	EHLO	This command identifies the SMTP client to the SMTP server and asks the server to send a reply stating which SMTP Service Extensions the server supports. The argument field contains the host name of the client.
EXPAND	EXPN	This command asks the receiver to confirm that the argument identifies a mailing list and, if so, to return the membership of that list. The full name of the users, if known, and the fully specified mailboxes are returned in a multiline reply.

Table 13 (Page 2 of 2). SMTP Commands

Name	Command	Description
HELLO	HELO	This command identifies the sender-SMTP to the receiver-SMTP. The argument field contains the host name of the sender-SMTP.
HELP	HELP	This command causes the receiver to send information to the sender of the HELP command. The command returns specific information about any command listed as a HELP argument.
MAIL	MAIL	This command initiates a mail transaction for mail data that is delivered to one or more mailboxes. The required argument field contains a reverse path. If the EHLO command was specified, the optional SIZE field may be used to indicate the size of the mail in bytes, and the optional BODY field may be used to specify whether a 7-bit message or an 8-bit MIME message is being sent.
NOOP	NOOP	This command requests an OK reply from the receiver. It does not affect any parameters or previously entered commands.
QUIT	QUIT	This command requests an OK reply from the receiver, and then it closes the transmission channel.
RECIPIENT	RCPT	This command identifies an individual recipient of the mail data; multiple recipients are specified by multiple RCPT commands.
RESET	RSET	This command aborts the current mail transaction. Any stored sender, recipient, or mail data is discarded, and all buffers and state tables are cleared. The receiver sends an OK reply.
VERIFY	VRFY	This command asks the receiver to confirm that the argument identifies a user. If it is a user name, the full name of the user, if known, and the fully specified mailbox are returned.

Figure 73 on page 138 shows the SMTP reply codes. The information shown in Figure 73 on page 138 is from RFC 821, and RFC 1869.

```
RFC's 821 and 1869
                                           Simple Mail Transfer Protocol


      4.2.1.  REPLY CODES BY FUNCTION GROUPS

         500 Syntax error, command unrecognized
            {This may include errors such as command line too long}
         501 Syntax error in parameters or arguments
         502 Command not implemented
         503 Bad sequence of commands
         504 Command parameter not implemented

         211 System status, or system help reply
         214 Help message
            {Information on how to use the receiver or the meaning of a
            particular non-standard command; this reply is useful only
            to the human user}

         220 <domain> Service ready
         221 <domain> Service closing transmission channel
         421 <domain> Service not available,
             closing transmission channel
            {This may be a reply to any command if the service knows it
            must shut down}

         250 Requested mail action okay, completed
         251 User not local; will forward to <forward-path>
         450 Requested mail action not taken: mailbox unavailable
            {E.g., mailbox busy}
         550 Requested action not taken: mailbox unavailable
            {E.g., mailbox not found, no access}
         451 Requested action aborted: error in processing
         551 User not local; please try <forward-path>
         452 Requested action not taken: insufficient system storage
         552 Requested mail action aborted: exceeded storage allocation
         553 Requested action not taken: mailbox name not allowed
            {E.g., mailbox syntax incorrect}
         354 Start mail input; end with <CRLF>.<CRLF>
         554 Transaction failed
         555  Requested action not taken:
              parameters associated with a MAIL FROM
              or RCPT TO command are not recgnized


Postel                                                        {Page 35}
```

Figure 73. SMTP Reply Codes. From RFC 821, and RFC 1869

Sample Debug Trace

The following describes how the output from an SMTP server trace using TRACE DEBUG is organized:

Conn_number
This is the TCP connection number. A value of 257 identifies a server working in batch mode. This often occurs when a server is reading a file that it has received from a local user before sending the file to the remote host.

In/Out_char
This character indicates the way the message or command is traveling. A > symbol indicates an outgoing message or command and a < symbol indicates an incoming message or command.

Cmd_line
This is the information exchanged between hosts.

Figure 74 on page 139 is a sample of an SMTP server trace using the TRACE DEBUG statement. Although all transactions between the local and remote hosts are shown, the data transferred by the DATA command is not shown.

In Figure 74, HOSTA is the local host, and HOSTB is the remote host. All lines starting with 257 show the SMTP server handling note 00000001 from local user TCPUSRA. Lines starting with a connection number of 1 show note 00000001 being sent to TCPUSRB@HOSTB. Lines starting with a connection number of 0 show HOSTB sending a note from TCPUSRB to the local host. The local host designates this note as note 00000002.

```
IBM VM SMTP Level nnn on Tue, 23 Oct 97 17:19:23 EST
257> 220 HOSTA.IBM.COM running IBM VM SMTP Level nnn on Tue, 23 Oct 97 17:19:25 EST
257< HELO HOSTA.IBM.COM
257> 250 HOSTA.IBM.COM is my domain name.  Yours too, I see!
257< MAIL FROM:<TCPUSRA@HOSTA.IBM.COM>
257> 250 OK
257< RCPT TO:<tcpusrb@hostb>
257> 250 OK
257< DATA
257> 354 Enter mail body.  End by new line with just a '.'
257> 250 Mail Delivered
257< QUIT
257> 221 HOSTA.IBM.COM running IBM VM SMTP Level nnnMX closing connection
  1< 220 HOSTB.IBM.COM running IBM VM SMTP Level nnn on Tue, 23 Oct 90 17:22:53 EST
  1> EHLO HOSTA.IBM.COM
  1< 250-HOSTB.IBM.COM is my domain name.
  1< 250-EXPN
  1< 250-HELP
  1< 250 SIZE 20000768
  1> MAIL FROM:<TCPUSRA@HOSTA.IBM.COM> SIZE=210
  1< 250 OK
  1> RCPT TO:<tcpusrb@hostb.IBM.COM>
  1< 250 OK
  1> DATA
  1< 354 Enter mail body.  End by new line with just a '.'
  1< 250 Mail Delivered
  1> QUIT
  1< 221 HOSTB.IBM.COM running IBM VM SMTP Level nnnMX closing connection
  0> 220 HOSTA.IBM.COM running IBM VM SMTP Level nnn on Tue, 23 Oct 90 17:23:18 EST
  0< HELO HOSTB.IBM.COM
  0> 250 HOSTA.IBM.COM is my domain name.
  0< MAIL FROM:<TCPUSRB@HOSTB.IBM.COM>
  0> 250 OK
  0< RCPT TO:<tcpusra@hosta.IBM.COM>
  0> 250 OK
  0< DATA
  0> 354 Enter mail body.  End by new line with just a '.'
  0> 250 Mail Delivered
  0< QUIT
  0> 221 HOSTA.IBM.COM running IBM VM SMTP Level nnnMX closing connection
```

Figure 74. A Sample of an SMTP Server Trace Using the DEBUG Statement

Sample LOG Information

In addition to the data that can be obtained using the TRACE command, the SMTP server provides LOG information. This LOG information can be directed to the console (the default), or to the SMTP LOG file on minidisk.

Figure 75 on page 140 shows sample LOG information matching the sample trace shown in Figure 74 on page 139. For example, the line starting with 10/23/97 17:23:18 shows when HOSTB is connected to the local host's port on connection 0 before sending note 00000002.

```
IBM VM SMTP Level nnn on Tue, 23 Oct 97 17:19:23 EST
10/23/97 17:19:24 Received Spool File 2289 From TCPUSRA at HOSTA
10/23/97 17:19:25 BSMTP Helo Domain: HOSTA.IBM.COM Yours too, I see!
10/23/97 17:19:25 Received Note 00000001 via BSMTP From <TCPUSRA@HOSTA.IBM.COM>
10/23/97 17:20:31 Delivered Note 00000001 to <tcpusrb@hostb.IBM.COM>
10/23/97 17:23:18 TCP (0) Helo Domain: HOSTB.IBM.COM
10/23/97 17:24:21 Received Note 00000002 via TCP (0) From <TCPUSRB@HOSTB.IBM.COM>
10/23/97 17:24:23 Delivered Note 00000002 to TCPUSRA at HOSTA
```

Figure 75. Sample LOG Output

Sample Resolver Trace

You can also enable the Resolver Trace for the SMTP server virtual machine. The Resolver Trace displays all requests and responses for name resolution to the console. To activate this type of tracing, add a TRACE RESOLVER statement to the SMTP CONFIG file.

Figure 76 on page 140 shows a sample of a resolver trace.

```
10/25/97 07:32:12      Resolving Recipient Address: <tcpuser@9.67.58.233 >
10/25/97 07:32:12      Resolving Recipient Address: <tcpfoo@hostvm>
* * * * * Beginning of Message * * * * *
Query Id:                 1
Flags:                    0000 0001 0000 0000
Number of Question RRs:   1
Question   1: 9.67.58.233 MX IN
Number of Answer RRs:     0
Number of Authority RRs:  0
Number of Additional RRs: 0
* * * * * End of Message * * * * *
10/25/97 07:32:12 #  1 UDP Query Sent, Try: 1 to NS(.1.) := 14.0.0.0
10/25/97 07:32:12 #  1 Adding Request to Wait Queue
10/25/97 07:32:12 #  1 Setting Wait Timer: 30 seconds
* * * * * Beginning of Message * * * * *
Query Id:                 2
Flags:                    0000 0001 0000 0000
Number of Question RRs:   1
Question   1: hostvm.ENDICOTT.IBM.COM MX IN
Number of Answer RRs:     0
Number of Authority RRs:  0
Number of Additional RRs: 0
* * * * * End of Message * * * * *
```

Figure 76 (Part 1 of 2). A Sample of an SMTP Resolver Trace

```
10/25/97 07:32:12 #  2 UDP Query Sent, Try: 1 to NS(.1.) := 14.0.0.0
10/25/97 07:32:12 #  2 Adding Request to Wait Queue
10/25/97 07:32:13      UDP packet arrived, 50 bytes, FullLength 50 bytes.
* * * * * Beginning of Message * * * * *
Query Id:                2
Flags:                   1000 0101 1000 0011
Number of Question RRs:  1
Question   1: hostvm.ENDICOTT.IBM.COM MX IN
Number of Answer RRs:    0
Number of Authority RRs: 0
Number of Additional RRs: 0
* * * * * End of Message * * * * *
* * * * * Beginning of Message * * * * *
Query Id:                3
Flags:                   0000 0001 0000 0000
Number of Question RRs:  1
Question   1: hostvm.ENDICOTT.IBM.COM A IN
Number of Answer RRs:    0
Number of Authority RRs: 0
Number of Additional RRs: 0
* * * * * End of Message * * * * *
10/25/97 07:32:27 #  3 UDP Query Sent, Try: 1 to NS(.1.) := 14.0.0.0
10/25/97 07:32:27 #  3 Adding Request to Wait Queue
10/25/97 07:32:28      UDP packet arrived, 50 bytes, FullLength 50 bytes.
* * * * * Beginning of Message * * * * *
Query Id:                3
Flags:                   1000 0101 1000 0011
Number of Question RRs:  1
Question   1: hostvm.ENDICOTT.IBM.COM A IN
Number of Answer RRs:    0
Number of Authority RRs: 0
Number of Additional RRs: 0
* * * * * End of Message * * * * *
```

Figure 76 (Part 2 of 2). A Sample of an SMTP Resolver Trace

Sample Notification Trace

TCP/IP Notification Tracing is enabled via a TRACE NOTICE statement in the SMTP CONFIG file. All TCP/IP notification events are traced to the console. Figure 77 on page 141 shows a sample of a notification trace.

```
12/10/97 22:59:14  TCP/IP Event Notification: I/O Interrupt
12/10/97 22:59:14  TCP/IP Event Notification: IUCV Interrupt
12/10/97 22:59:14  TCP/IP Event Notification: IUCV Interrupt
12/10/97 22:59:14  TCP/IP Event Notification: UDP Datagram Delivered
12/10/97 22:59:14  TCP/IP Event Notification: UDP Datagram Delivered
12/10/97 22:59:14  TCP/IP Event Notification: UDP Datagram Delivered
12/10/97 22:59:14  TCP/IP Event Notification: Connection State Changed
12/10/97 22:59:14  TCP/IP Event Notification: Data Delivered on Conn 1, bytes delivered=92
12/10/97 22:59:14  TCP/IP Event Notification: Data Delivered on Conn 1, bytes delivered=50
12/10/97 22:59:14  TCP/IP Event Notification: Data Delivered on Conn 1, bytes delivered=8
12/10/97 22:59:14  TCP/IP Event Notification: Data Delivered on Conn 1, bytes delivered=8
12/10/97 22:59:14  TCP/IP Event Notification: Data Delivered on Conn 1, bytes delivered=55
12/10/97 22:59:15  TCP/IP Event Notification: Data Delivered on Conn 1, bytes delivered=20
12/10/97 22:59:15  TCP/IP Event Notification: Connection State Changed
12/10/97 22:59:16  TCP/IP Event Notification: Connection State Changed
12/10/97 22:59:16  TCP/IP Event Notification: Connection State Changed
```

Figure 77. A Sample of a Notification Trace

Sample Connection Activity Trace

TCP/IP Connection Activity Tracing is enabled via a TRACE CONN statement in the SMTP CONFIG file. All connection state changes are logged to the console. Figure 78 on page 142 shows a sample of a connection activity trace.

```
12/10/97 22:44:30  Connection State Change, Conn = 1, State = Open
12/10/97 22:44:31  Connection State Change, Conn = 1, State = Connection closing
12/10/97 22:44:31  Connection State Change, Conn = 1, State = Nonexistent
```

Figure 78. A Sample of a Connection Activity Trace

Chapter 10. RPC Programs

This chapter describes Remote Procedure Call (RPC) programs, including call messages and reply messages. For more information about RPC, see RFCs 1014 and 1057. This chapter also describes Portmapper.

General Information about RPC

The current version of RPC is Version 2. The layout for RPC messages is either a CALL-MSG or REPLY-MSG. Both layouts need a transaction identifier (XID) to identify and reliably map port numbers, and a field to identify whether the message is a CALL-MSG or REPLY-MSG.

The following sections describe the structure of call and reply messages.

RPC Call Messages

The first word in a call message is the XID, the message identifier. The second word indicates the type of message, which is 0 for a call message. Figure 79 shows the structure of a call message. The offsets and their corresponding field descriptions are:

Offset	**Field Description**
X'00'	XID, message identifier
X'04'	Type of message (0)
X'08'	RPC version
X'0C'	RPC program number
X'10'	Program version
X'14'	Procedure number
X'18'	Authentication credentials field
X'1C'	Byte length of `Cred Data` field
X'1C'+Cred-L	Authentication verifier (see Table 14 on page 144)
X'20'+Cred-L	Authentication verifier data length
Data field	Data specific to the procedure called.

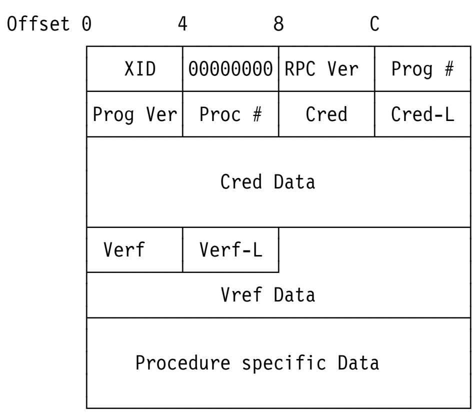


Figure 79. RPC Call Message Structure

Table 14 describes the RPC credentials found in the `Cred` data field, shown in Figure 79.

Table 14. RPC Credentials

Name	Number	Description
AUTH_NULL	0	The client does not know its identity or the server does not need to know the identity of the client.
AUTH_UNIX	1	Client identifies itself as a UNIX system.
AUTH_SHORT	2	Used as an abbreviated authentication structure.
AUTH_DES	3	Used for a DES authentication.

RPC Reply Messages

The first word in a reply message is the XID. The second word indicates the type of message, which is 1 for a reply message. There are two types of reply messages: accepted and rejected. If the value of the reply_stat field is 0, the message has been accepted. If the value of the reply_stat field is 1, the message has been rejected.

Accepted Reply Messages

Figure 80 shows the structure of an accepted reply message. The offsets and their corresponding field descriptions are:

Offset	Field Description
X'00'	XID, message identifier
X'04'	Type of message, 1
X'08'	Reply stat
X'0C'	Authentication verifier (see Table 14)
X'10'	Authentication verifier data byte length
X'14'	Accept_stat
X'18'	Acc_stat dependent data.

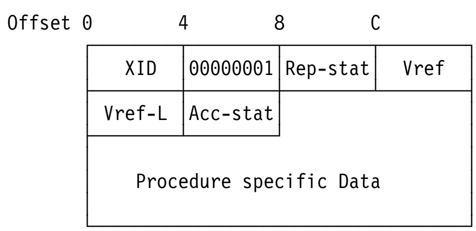


Figure 80. Structure of an RPC Accepted Reply Message

Acc_stat is a one word return code for NFS procedures that has a value described in Table 15. If acc_stat=SUCCESS, the data is specific to the procedure. If acc_stat=PROG_MISMATCH, two words with the latest and earliest supported versions of the program are returned. For the other acc_stat values described in Table 15, data is not returned. For more information about acc_stat values, see RFC 1057.

Table 15 (Page 1 of 2). RPC Accept_stat Values

Name	Number	Description
SUCCESS	0	RPC executed successfully.
PROG_UNAVAIL	1	Remote has not exported program.
PROG_MISMATCH	2	Program cannot support version number.
PROC_UNAVAIL	3	Program cannot support procedure.

Table 15 (Page 2 of 2). RPC Accept_stat Values

Name	Number	Description
GARBAGE_ARGS	4	Procedure cannot decode parameters.

Rejected Reply Messages

Figure 81 shows the structure of a rejected reply message. The offsets and their corresponding field descriptions are:

Offset **Field Description**
X'00' XID, message identifier
X'04' Type of message, 1
X'08' Reply_stat, 1
X'0C' Reject_stat switch
X'10' Reject_stat specific data.

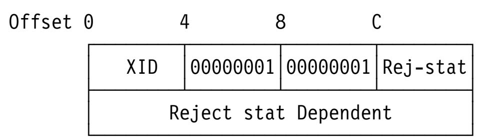


Figure 81. Structure of an RPC Rejected Reply Message

The reject_stat switch indicates the reason for a rejected reply message. If the value of the reject_stat switch is 1, an RPC_MISMATCH, indicating that the version of RPC is not supported, has occurred. The reject_stat dependent field, shown in Figure 81, contains the latest and earliest RPC supported versions. If the value of the reject_stat switch is 0, an AUTH_ERROR, indicating an authentication error, has occurred. The reject stat dependent field, shown in Figure 81, contains a one word auth_stat value. Table 16 describes the auth_stat values. For more information about auth_stat values, see RFC 1057.

Table 16. RPC Auth_stat Values

Name	Number	Description
AUTH_BACKRED	1	Bad credential, seal broken
AUTH_CTEDCRED	2	Client must begin new session
AUTH_ERF	3	Bad verifier, seal broken
AUTH_REJECTEDVERF	4	Verifier expired or replayed
AUTH_TOOWEAK	5	Rejected for security reasons

RPC Support

RPC supports the following functions:

Authentication — The mount service uses AUTH_UNIX and AUTH_NONE style authentication only.

Transport Protocols — The mount service is supported on both UDP and TCP.

Port Number — Consult the server's portmapper, described in RFC 1057, to find the port number on which the mount service is registered. The port number is usually 111.

Portmapper

Portmapper is a program that maps client programs to the port numbers of server programs. The current version for RPC program 100000 (Portmapper) is Version 2. For more information about Portmapper, see Appendix A of RFC 1057.

Portmapper Procedures

Table 17 describes Portmapper procedures.

Table 17. Portmapper Procedures

Name	Number	Description
PMAPROC_NULL	0	Procedure 0 is a dummy procedure that senses the server.
PMAPROC_SET	1	Registers a program on Portmapper.
PMAPROC_UNSET	2	Removes a registered program from Portmapper.
PMAPROC_GETPORT	3	Gives client's program and version number. The server responds to the local port of the program.
PMAPROC_DUMP	4	Lists all entries in Portmapper. This is similar to the RPCINFO command.
PMAPROC_CALLIT	5	Used by a client to call another remote procedure on the same host without the procedure number.

Chapter 11. RouteD Diagnosis

RouteD is a server that implements the Routing Information Protocol (RIP) described in RFC 1058 (RIP Version 1) and RFC 1723 (RIP Version 2). It provides an alternative to static TCP/IP gateway definitions. When properly configured, the VM/ESA host running with RouteD becomes an active RIP router in a TCP/IP network. The RouteD server dynamically creates and maintains network routing tables using RIP. This protocol allows gateways and routers to periodically broadcast their routing tables to adjacent networks, and enables the RouteD server to update its host routing table. For example, the RouteD server can determine if a new route has been created, if a route is temporarily unavailable, or if a more efficient route exists for a given destination.

Before RouteD was implemented for TCP/IP, static route tables were used for routing IP datagrams over connected networks. However, the use of static routes prevents a host from being readily able to respond to changes in the network. By implementing the Routing Information Protocol (RIP) between a host and TCP/IP, the RouteD server dynamically updates the internal routing tables when changes to the network occur.

The RouteD server reacts to network topology changes on behalf of TCP/IP by maintaining the host routing tables, processing and generating RIP datagrams, and performing error recovery procedures.

Figure 82 shows the RouteD environment.

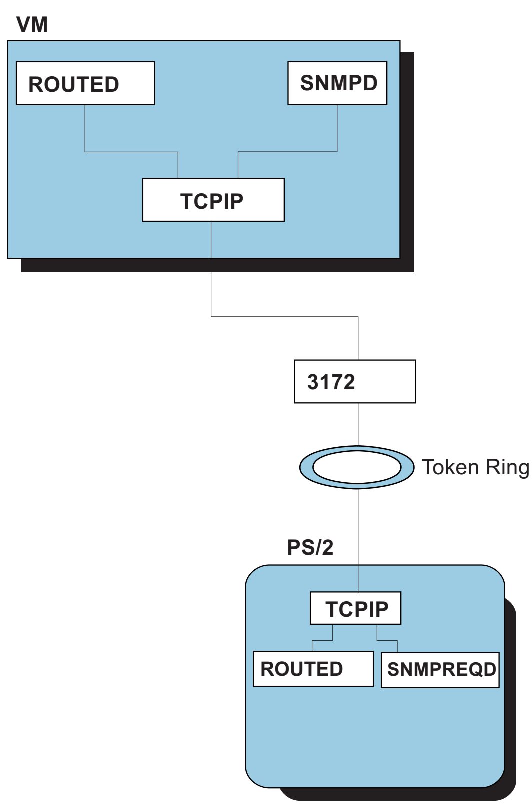


Figure 82. RouteD Environment

The RouteD protocol is based on the exchange of RIP messages. There are two types of messages:

- **Request message** - Sent from a client (another RIP router) as a request to transmit all or part of the receiving host's routing table.
- **Response message** - Sent from RouteD to a client (another RIP router) containing all or part of the sending host's routing table.

Incoming Datagram RouteD Processing

Only RIP datagrams are processed by the RouteD server, as opposed to the router itself, which actually routes datagrams as previously described. Incoming RIP datagrams contain one of the following commands:

- Request
- Response
- Trace On
- Trace Off

Incoming Request Datagrams

Request datagrams are requests from other routers for one or more of the RouteD server's routes. The internet addresses of the desired routes are listed in the datagram. A special form of the Request datagram requests a single route in an illegal address family (*AF_UNSPEC*) and lists a metric (hop count) of 16, which is considered infinity in RIP. This request is treated as a request for the server's complete routing table.

Note: This form of request is issued by RouteD only during initialization.

Incoming Response Datagrams

Response datagrams contain routing table entries, and are sent by routers periodically and on demand. The RouteD server transmits a complete set of routes on each attached network every thirty seconds, by using Response datagrams.

Incoming Trace On and Trace Off Datagrams

Tracing is not officially supported in RIP, but these datagram types are reserved and most RouteD servers choose to process them. **Trace On** turns on tracing (or expands the amount of tracing currently in effect), and **Trace Off** turns off tracing.

Outgoing Datagram RouteD Generation

The RouteD server transmits only Request and Response datagrams.

Outgoing Request Datagrams

Request datagrams are generated during RouteD startup, requesting complete route tables from adjacent routers. This is the only time Request datagrams are generated by RouteD. The other form of the Request datagram is used by other applications to query the server route tables.

Outgoing Response Datagrams

The RouteD server transmits a complete set of routes to adjacent routers every thirty seconds using Response datagrams. RouteD servers that are started as passive routers collect data only; they provide routing information only when requested via a port other than port 520.

In addition, the RouteD server replies to incoming Request datagrams by sending Response datagrams containing the requested routing information.

RouteD Route Table and Interface List

RouteD maintains its own route table, which is similar to IP's. While these two tables must be synchronized, they do not need to be identical. There are cases where routes are known to RouteD but are not known to IP, and other cases where routes are known to IP but are not known to RouteD. Therefore, two tables must be maintained. RouteD's route table is implemented as a hash table, with doubly linked lists used as hash chains to hold collisions.

RouteD also maintains an interface list, which contains all the active interfaces that RouteD can use. When an interface's three minute timer expires, that interface is removed from the active interface list. When a datagram arrives on that interface, it is again added to the active interface list. The RouteD interface list is implemented as a linked list.

Diagnosing Problems

Problems with RouteD are generally reported under one of the following categories:

- "Connection Problems"
- "PING Failures" on page 150
- "Incorrect Output" on page 153
- "Session Outages" on page 153

Use the information provided in the following sections for problem determination and diagnosis of errors reported against RouteD.

Connection Problems

RouteD connection problems are reported when RouteD is unable to connect to TCP/IP. Generally, this type of problem is caused by an error in the TCP/IP configuration or supporting definitions.

In configurations with multiple stacks, a RouteD server must be started for each stack that requires routing services. To associate with a particular stack, use the PORT statement of the TCP/IP configuration file (PROFILE TCPIP) to define the name of the RouteD server virtual machine that will service that stack. The user ID of the RouteD server for a given stack must also be included in its OBEY list in PROFILE TCPIP.

Documentation

The following documentation should be available for initial diagnosis of RouteD connection problems:

- PROFILE TCPIP information
- TCPIP DATA information
- DTCPARMS information
- RouteD ETC GATEWAYS file information
- ROUTED CONFIG file information
- Trace output

Analysis

Refer to the *TCP/IP Planning and Customization* for problems related to TCP/IP configuration.

Diagnostic steps for connection problems:

1. Verify the accuracy of the RouteD startup parameters that have been specified in the DTCPARMS file.
2. Make sure that RouteD is configured correctly in the PROFILE TCPIP information.
3. UDP port 520 must be reserved for RouteD. Verify that the assigned port number and the RouteD server user ID are correct.
4. Ensure that TCPIP DATA designates the correct TCP/IP stack machine.

PING Failures

If the `PING` command fails on a system where RouteD is being used, a client is unable to get a response to a `PING` command. Before doing anything else, run `NETSTAT GATE`. This should tell you which gateways are configured. If no gateways are configured, `PING` will not work. In addition to this, run `NETSTAT DEVLINK`, and ensure that the device for the link of the address you are trying to `PING` is in "Ready" status. If the device status is "Inactive", `PING` will not work.

Documentation

The following documentation should be available for initial diagnosis of ping failures:

- PROFILE TCPIP information
- `NETSTAT GATE` command results

More documentation that might be needed is described in the “Analysis” section.

Analysis

Table 18 on page 151 shows symptoms of ping failures and describes the steps needed for initial diagnosis of the error.

Table 18. RouteD ping Failures

ping Failure	**Action Steps**
Incorrect response (ping timed out or Network Unreachable)	1. Make sure that the ping command contains a valid destination IP address for the remote host. If the destination IP address is a virtual IP address (VIPA), make sure that VIPA is defined correctly. See the *TCP/IP Planning and Customization* for information about rules and recommendations for defining a virtual IP address. 2. Make sure that the router providing the RIP support involved in the ping transaction is active and is running with a correct level of some application that provides RIP support. If the destination router is not running RIP, make sure that static routes are defined from the destination router to the local host. 3. If the ping command was issued from a client on a VM/ESA server, issue a NETSTAT GATE command to display the routing tables. Verify that the routes and networks are correct as defined in PROFILE TCPIP and the ETC GATEWAYS file. In addition, issue a NETSTAT DEVLINK command to insure that the device associated with the link for the desired IP address is in "Ready" status. 4. If the ping command was issued from a workstation operating system, verify that the routes and networks are defined correctly in the TCP/IP configuration and the ETC GATEWAYS file of TCP/IP. 5. If there are no problems with the routes and networks, check for broken or poorly-connected cables between the client and the remote host. This includes checking the internet interfaces (such as Token-Ring and Ethernet) on the server. 6. Consider whether changes may have taken place elsewhere in the network. For example, if a second host has been added using the same IP address as a host involved in routing your PING's packets, the packets may get misrouted and the PING will time out. Likewise, failure to subnet when required can lead to packets being incorrectly routed. Some routing hardware uses more robust routing algorithms than others, so if hardware has changed anywhere along the route of your PING, an unsupported network configuration that previously functioned might now fail. 7. If you have multiple subnets with the same IP Address, see the section "Multiple Subnets with the Same IP Address" on page 152 following this table.
Unknown Host	If the ping command was issued with a *name*, try again with the actual IP address. If the ping command is successful with an IP address, then the problem is with nameserving and not RouteD.

Multiple Subnets with the Same IP Address

The new RouteD server that is available with TCP/IP Function Level 320 contains many enhancements and improvements over the RouteD server in past releases of VM TCP/IP. One major change is the handling of subnets that have the same IP address (see the figure below). Network topologies that consist of two subnets that have the same IP subnet address (140.48.96) for different internal networks cause routing ambiguity. The FL320 RouteD server now recognizes this topology and will dynamically build the routing table entries that are required. Until RouteD has fully initialized its routing tables (4 to 5 minutes), "network unreachable" conditions may occur when trying to ping through one of these subnets (Host D or Host C) that have the same IP address (140.48.96). After this initial delay, the unreachable subnet will become reachable. To avoid such problems, define one of the duplicate subnet addresses to a different subnet address (140.48.96.64) using variable subnetting. This will remove the ambiguity during initialization and ensure that both subnet networks will be reachable. For information and examples of variable subnetting, see the Varsubnetting parameter of the "ASSORTEDPARMS Statement" in the *TCP/IP Planning and Customization*.

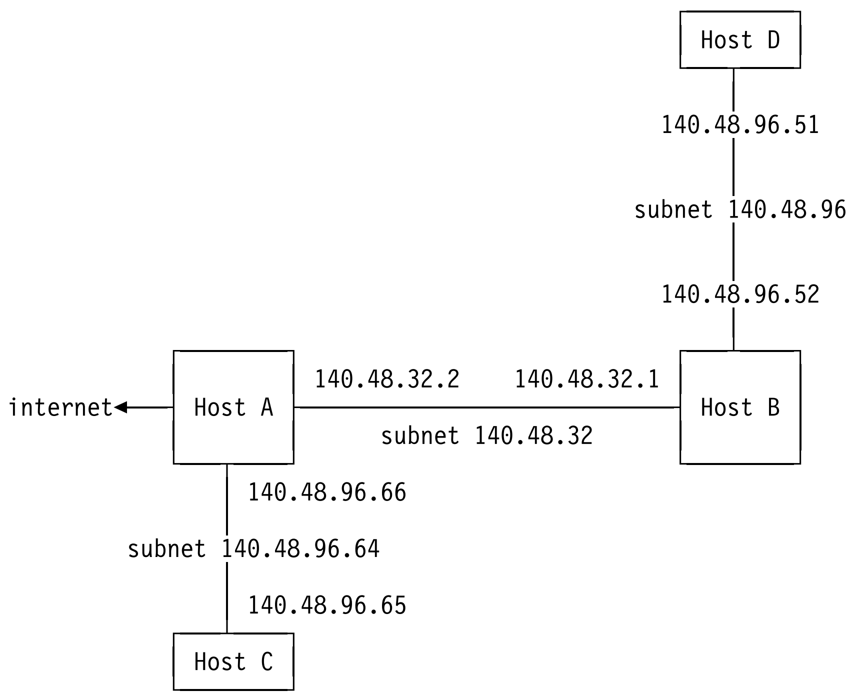


Table 19. Variable Subnetted Local Network

HOST	IP ADDRESS	SUBNET MASK	SUBNET ID	HOST ID
HOST A	140.48.32.2	255.255.255.0	140.48.32	2
	140.48.96.66	255.255.255.192	140.48.96.64	2
HOST B	140.48.32.1	255.255.255.0	140.48.32	1
	140.48.96.52	255.255.255.0	140.48.96	52
HOST C	140.48.96.65	255.255.255.192	140.48.96.64	1
HOST D	140.48.96.51	255.255.255.0	140.48.96	51

Incorrect Output

Problems with incorrect output are reported when the data sent to the client is not seen in its expected form. This could be incorrect TCP/IP output, RIP commands that are not valid, incorrect RIP broadcasting information, incorrect updates of routing tables, or truncation of packets.

Documentation

The following documentation should be available for initial diagnosis of incorrect output:

- TCP/IP and/or RouteD Messages
- Trace data
- PROFILE TCPIP information

Analysis

Table 20 shows symptoms of incorrect output and describes the actions needed for initial diagnosis of the error.

Table 20. RouteD Incorrect Output

Incorrect Output	Action Steps
TCP/IP Incorrect Output	1. If the TCP/IP console shows a message, refer to *TCP/IP Messages and Codes* and follow the directions for system programmer response for the message. 2. In the event of TCP/IP loops or hangs, refer to the *VM/ESA: Diagnosis Guide*.
RouteD Incorrect Output	If the RouteD console shows a message, refer to *TCP/IP Messages and Codes* and follow the directions for system programmer response for the message.

Session Outages

Session outages are reported as an unexpected abend or termination of a TCP/IP connection.

Documentation

The following documentation should be available for initial diagnosis of session outages:

- TCP/IP and/or RouteD Messages
- Trace data
- TCPIP PROFILE information
- `NETSTAT GATE` command results

Analysis

Table 21 on page 154 shows symptoms of session outages and describes the steps needed for initial diagnosis of the error.

Table 21. RouteD Session Outages	
Session Outage	**Action Steps**
TCP/IP session outage	1. If the TCP/IP console shows a TCP/IP error message, refer to *TCP/IP Messages and Codes* and follow the directions for system programmer response for the message. 2. In the event of a TCP/IP abend, refer to the *VM/ESA: Diagnosis Guide*.
session outage	If an error message is displayed, refer to *TCP/IP Messages and Codes* and follow the directions for system programmer response for the message.

Activating RouteD Trace and Debug

RouteD trace facilities exist that can be useful in identifying the cause of routing problems. This section discusses these trace and debug requests and how they can be started and stopped.

The activation of trace facilities in RouteD is accomplished by specifying the desired trace level parameter in addition to the usual processing parameters on command invocation.

You can initialize RouteD with the tracing option. The tracing option is set by editing the DTCPARMS file, and specifying the necessary parameters for the :Parms. tag of the DTCPARMS file.

RouteD Trace and Debug Commands

Use the ROUTED command to enable the following trace and debug parameters.

```
►►──ROUTED──┬──────┬──┬───────────┬──────────────────────────────►◄
            └─-dp──┘  │ ┌─────┐   │
                      └─▼─-t──┴(1)┘
```

Note:
[1] The -t option can be repeated up to 4 times to achieve the desired level of tracing.

Trace information is written to the spooled console of the server virtual machine.

The trace and debug parameters that can be specified for the RouteD server are:

Operands

-dp
Activates tracing of packets to and from adjacent routers in addition to RIP network routing tables that are received and broadcasted. Packets are displayed in data format. Output is written to the console.

-t Activates tracing of actions by the RouteD server.

-t -t
Activates tracing of actions and packets sent or received.

-t -t -t
Activates tracing of actions, packets sent or received, and packet history. Circular trace buffers are used for each interface to record the history of all packets traced. This history is included in the trace output whenever an interface becomes inactive.

-t -t -t -t
Activates tracing of actions, packets sent or received, packet history, and packet contents. The RIP network routing information is included in the trace output.

Note: Spaces are required between each **-t** parameter when more than one is specified.

Usage Notes

1. For information on the remaining available RouteD parameters see the *TCP/IP Planning and Customization*.
2. Parameters are separated by one or more blanks.
3. Parameters can be specified in mixed case.

RouteD Trace and Debug SMSG Commands

Use the SMSG command to enable or disable RouteD trace and debug parameters that may or may not have been specified at server initialization or on a previous SMSG command.

```
►►──SMSG──server_id──┬─HELP─────────────────┬──────────────────────►◄
                     │        ┌──────────┐  │
                     ├─PARMS──▼─parms────┴──┤
                     └─TABLES───────────────┘
```

Operands

server_id
Specifies the user ID of the virtual machine running the VM RouteD server.

HELP
Provides a list of valid SMSG commands accepted by RouteD.

PARMS
One or more of the following parameters separated by a space.

parms

-dp
Activates tracing of packets to and from adjacent routers in addition to RIP network routing tables that are received and broadcasted. Packets are displayed in data format. Output is written to the console.

-dq
Disables "-dp" tracing.

-t Activates tracing of actions by the RouteD server.

-t -t
Activates tracing of actions and packets sent or received.

-t -t -t
Activates tracing of actions, packets sent or received, and packet history. Circular trace buffers are used for each interface to record the history of all packets traced. This history is included in the trace output whenever an interface becomes inactive.

-t -t -t -t
Activates tracing of actions, packets sent or received, packet history, and packet contents. The RIP network routing information is included in the trace output.

Note: Spaces are required between each **-t** parameter when more than one is specified.

-tq Disable all "-t" traces.

TABLES
Activates the display of the following RouteD internal tables;

routing
RouteD's routing tables

interface
Interface connections defined by "Device and Link" statements.

gateways options
Options as defined in the "ETC GATEWAYS" file

Note: This option is provided primarily for debugging purposes only.

Usage Notes

1. For information about other supported RouteD SMSG parameters see the *TCP/IP Planning and Customization*.

Examples

The following SMSG command passes parameters to a RouteD server running in the ROUTED2 virtual machine.

```
smsg routed2 parms -dp -t
Ready;
10:04:20  * MSG FROM ROUTED2 : PARMS -DP -T
```

Trace Output

Figure 83 shows an example of the output received from a RouteD server with tracing enabled (that is, the **-dp -t -t -t -t** parameter had been specified).

```
DTCRUN1011E Server started at 08:39:00 on 27 Jan 1999 (Wednesday)
DTCRUN1011E Running "ROUTED -DP -T -T -T -T"
DTCRTD4820I VM TCP/IP RouteD Server Level 320
DTCRTD4929I Port 520 assigned to router
DTCRTD4828I Input parameter(s): -DP -T -T -T -T
DTCRTD4868I Tracing actions enabled Wed Jan 27 08:39:02 1999
DTCRTD4869I Tracing packets enabled Wed Jan 27 08:39:02 1999
DTCRTD4870I Tracing history enabled Wed Jan 27 08:39:02 1999
DTCRTD4871I Tracing packet contents enabled Wed Jan 27 08:39:02 1999
DTCRTD4823I Tracing debug packets enabled Wed Jan 27 08:39:02 1999
```

Figure 83 (Part 1 of 3). A Sample RouteD Server Trace

```
DTCRTD4932I *******************************************************
DTCRTD8488I Opening RouteD config file (ROUTED CONFIG)
DTCRTD4932I *******************************************************
DTCRTD8497I RIP_SUPPLY_CONTROL: RIP1
DTCRTD8497I RIP_RECEIVE_CONTROL: ANY
DTCRTD8498I RIP2 authentication disabled at router-wide level (all interfaces)
DTCRTD4932I *******************************************************
DTCRTD4850I Processing interface TR1
DTCRTD4932I *******************************************************
DTCRTD4948I This interface is not point-to-point
DTCRTD4943I Adding network route for interface
DTCRTD4882I Wed Jan 27 08:39:03 1999:
DTCRTD4883I ADD destination 9.0.0.0, router 9.127.32.100, metric 1, flags UP, state INTERFACE|CHANGED|INTERNAL, timer 0
DTCRTD4943I Adding subnetwork route for interface
DTCRTD4883I ADD destination 9.127.32.0, router 9.127.32.100, metric 1, flags UP, state INTERFACE|CHANGED|SUBNET, timer 0
DTCRTD4932I *******************************************************
DTCRTD4850I Processing interface PORTER
DTCRTD4932I *******************************************************
DTCRTD4940I Point-to-point interface, using dstaddr
DTCRTD4943I Adding subnetwork route for interface
DTCRTD4883I ADD destination 9.127.68.20, router 9.127.68.21, metric 1, flags UP, state INTERFACE|CHANGED|SUBNET, timer 0
DTCRTD4943I Adding host route for interface
DTCRTD4883I ADD destination 9.127.68.22, router 9.127.68.21, metric 1, flags UP|HOST, state INTERFACE|CHANGED, timer 0
DTCRTD4932I *******************************************************
DTCRTD4850I Processing interface STOUT
DTCRTD4932I *******************************************************
DTCRTD4940I Point-to-point interface, using dstaddr
DTCRTD4943I Adding subnetwork route for interface
DTCRTD4883I ADD destination 9.127.68.24, router 9.127.68.25, metric 1, flags UP, state INTERFACE|CHANGED|SUBNET, timer 0
DTCRTD4943I Adding host route for interface
DTCRTD4883I ADD destination 9.127.68.26, router 9.127.68.25, metric 1, flags UP|HOST, state INTERFACE|CHANGED, timer 0
DTCRTD4932I *******************************************************
DTCRTD4934I Opening ETC GATEWAYS file (ETC GATEWAYS)
DTCRTD4932I *******************************************************
DTCRTD4925I Start of ETC GATEWAYS processing
DTCRTD4945I ifwithnet:  compare with PORTER
DTCRTD4947I netmatch 9.127.68.22 and 9.127.68.21
DTCRTD4936I Adding passive host route 9.127.68.22 via gateway 9.127.68.21, metric 1
DTCRTD4883I DELETE destination 9.127.68.22, router 9.127.68.21, metric 1, flags UP|HOST, state INTERFACE|CHANGED, timer 0
DTCRTD4921I Deleting route to interface PORTER? (timed out?)
DTCRTD4883I ADD destination 9.127.68.22, router 9.127.68.21, metric 1, flags UP|HOST, state PASSIVE|INTERFACE|CHANGED, timer 0
DTCRTD4945I ifwithnet:  compare with STOUT
DTCRTD4947I netmatch 9.127.68.26 and 9.127.68.25
DTCRTD4936I Adding passive host route 9.127.68.26 via gateway 9.127.68.25, metric 1
DTCRTD4883I DELETE destination 9.127.68.26, router 9.127.68.25, metric 1, flags UP|HOST, state INTERFACE|CHANGED, timer 0
DTCRTD4921I Deleting route to interface STOUT? (timed out?)
DTCRTD4883I ADD destination 9.127.68.26, router 9.127.68.25, metric 1, flags UP|HOST, state PASSIVE|INTERFACE|CHANGED, timer 0
DTCRTD4926I End of  ETC GATEWAYS processing
DTCRTD4849I RouteD Server started
```

Figure 83 (Part 2 of 3). A Sample RouteD Server Trace

```
=============== Sending packet to client (length=24)
0000  01010000 00000000 00000000 00000000   00000000 00000010 00000000 00000000
0020(32)
DTCRTD4899I     REQUEST to 9.127.32.255 -> 520 ver 1  Wed Jan 27 08:39:18 1999
=============== RIP net info (length=20)
0000  00000000 00000000 00000000 00000000   00000010 00000000 00000000 00000000
0020(32)
DTCRTD4903I         (request for full tables)
=============== Sending packet to client (length=24)
0000  01010000 00000000 00000000 00000000   00000000 00000010 00000000 00000000
0020(32)
DTCRTD4899I     REQUEST to 9.127.68.22 -> 520 ver 1  Wed Jan 27 08:39:21 1999
=============== RIP net info (length=20)
0000  00000000 00000000 00000000 00000000   00000010 00000000 00000000 00000000
0020(32)
DTCRTD4903I         (request for full tables)
=============== Sending packet to client (length=24)
0000  01010000 00000000 00000000 00000000   00000000 00000010 00000000 00000000
0020(32)
DTCRTD4899I     REQUEST to 9.127.68.26 -> 520 ver 1  Wed Jan 27 08:39:21 1999
=============== RIP net info (length=20)
0000  00000000 00000000 00000000 00000000   00000010 00000000 00000000 00000000
0020(32)
DTCRTD4903I         (request for full tables)
DTCRTD4829I Waiting for incoming packets
=============== Received packet from client (length=4)
0000  02010000 00000002 00eb24b0 00d33410   00000000 00000000 00000000 00000000
0020(32)
DTCRTD4899I     RESPONSE from 9.127.32.29 -> 520 ver 1  Wed Jan 27 08:39:21 1999
=============== Received packet from client (length=24)
0000  02010000 00020000 00000000 00000000   00000000 00000004 00000000 00000000
0020(32)
DTCRTD4899I     RESPONSE from 9.127.32.252 -> 520 ver 1  Wed Jan 27 08:39:21 1999
=============== RIP net info (length=20)
0000  00020000 00000000 00000000 00000000   00000004 00000000 00000000 00000000
0020(32)
DTCRTD4902I         destination 0.0.0.0 metric 4
DTCRTD4882I Wed Jan 27 08:39:23 1999:
DTCRTD4883I ADD destination 0.0.0.0, router 9.127.32.252, metric 5, flags UP|GATEWAY, state CHANGED|DEFAULT, timer 0
DTCRTD4829I Waiting for incoming packets
=============== Received packet from client (length=24)
0000  02010000 00020000 00000000 00000000   00000000 00000004 00000000 00000000
0020(32)
```

Figure 83 (Part 3 of 3). A Sample RouteD Server Trace

```
DTCRTD4899I     RESPONSE from 9.127.32.251 -> 520 ver 1  Wed Jan 27 08:39:24 1999
=============== RIP net info (length=20)
0000  00020000 00000000 00000000 00000000   00000004 00000000 00000000 00000000
0020(32)
DTCRTD4902I         destination 0.0.0.0 metric 4
DTCRTD4829I Waiting for incoming packets
=============== Received packet from client (length=24)
0000  02010000 00020000 00000000 00000000   00000000 00000004 00000000 00000000
0020(32)
DTCRTD4899I     RESPONSE from 9.127.32.249 -> 520 ver 1  Wed Jan 27 08:39:36 1999
=============== RIP net info (length=20)
0000  00020000 00000000 00000000 00000000   00000004 00000000 00000000 00000000
0020(32)
DTCRTD4902I         destination 0.0.0.0 metric 4
DTCRTD4829I Waiting for incoming packets
=============== Received packet from client (length=4)
0000  02010000 00020000 00000000 00000000   00000000 00000005 00000000 00000000
0020(32)
DTCRTD4899I     RESPONSE from 9.127.32.29 -> 520 ver 1  Wed Jan 27 08:39:38 1999
DTCRTD4829I Waiting for incoming packets
=============== Received packet from client (length=24)
0000  02010000 00020000 00000000 00000000   00000000 00000004 00000000 00000000
0020(32)
DTCRTD4899I     RESPONSE from 9.127.32.250 -> 520 ver 1  Wed Jan 27 08:39:41 1999
=============== RIP net info (length=20)
0000  00020000 00000000 00000000 00000000   00000004 00000000 00000000 00000000
0020(32)
DTCRTD4902I         destination 0.0.0.0 metric 4
```

Figure 83 (Part 4 of 3). A Sample RouteD Server Trace

Chapter 12. Network File System

This chapter describes debugging facilities for NFS. Included are descriptions of traces as well as the different procedures implemented for TCP/IP VM.

General NFS Debugging Features

NFS has several features for debugging. Here is a general list of some of the debugging features.

1. Several levels of trace information are available. You can ask to write trace information to the VM NFS server machine console. Use the M start up parameter or the SMSG MASK command to set the mask and write trace information to the server machine console. Several mask values result in console information:

   500 Displays information about processing to decode names, particularly the name translation that takes place for SFS and minidisk files when the **names=trans** option is used on mount.

   501 Shows NFS requests (e.g., nfsread or lookup) received by the VM NFS server, and the responses to those requests. This shows the high level flow of requests between NFS client and server.

   502 Displays information related to mount requests, including PCNFSD and translation table processing.

   503 Displays information about initialization and SMSG REFRESH CONFIG processing.

   504 Displays error messages describing the errors received by the VM NFS server when processing SFS and BFS files and directories. These are the error codes given on routines such as DMSOPEN for SFS files, and the open() function call for BFS files.

   505 Displays information related to internal tasks being dispatched.

   506 Displays information related to NFS requests, but with more details than the M 501 trace.

   507 Causes the VM NFS server to call VMDUMP and write information to the console for all SFS and BFS errors except 'file not found'. In addition to the 507 mask value, the VMNFS DUMP_REQ file must contain the correct value. See note 6 on page 162.

   508 Displays information related to sockets used in the VM NFS server.

   509 Displays file I/O related information.

   510 Displays buffers related to sockets used in the VM NFS server.

   999 Includes all of the above except mask value 510.

   The M parameter may be used multiple times on the start up command. For example, you can specify the following in the DTCPARMS file:

   ```
   :parms.M 501 M 504
   ```

   You may specify only one mask value at a time on a MASK command delivered via CP SMSG, but the settings are cumulative. Specifying 'SMSG VMNFS M MASK 0' clears all previously set mask values.

2. VMNFS maintains information and usage data about client mounts. You can see this information using the SMSG VMNFS M QUERY command. 'SMSG VMNFS M QUERY' shows you summary counts for the entire VM NFS server. Use the DETAILS option on the 'SMSG VMNFS M QUERY RESOURCE' command to see usage counts for individual mount points.

   Note that sometimes the display can contain misleading information. The counts are reset if the VM NFS server is restarted. A negative mount count could be seen if an UNMOUNT is done following a server restart. Also, in response to a person's request to MOUNT, or for any other service, the NFS client may send several requests to the server. (Duplicate requests may be sent depending on network speed, for example.)

3. The VM NFS server maintains a limited amount of host error information for SFS and BFS directories. This can assist in determining the real reason for an NFSERR_IO return code (for example) given to an NFS client. See the SMSG VMNFS M ERROR command in the *TCP/IP User's Guide* for more information.

4. Console messages about invalid calls to program number 200006 are suppressed, unless the mask controlling internal tracing (M 505) is active. These calls are emitted by AIX® Version 3 clients.

5. The SIGERROR function will automatically write the internal trace table to disk (file name SIGERROR.TRACEV) if the trace mask is non-zero. A save-area traceback will also be written to the console when the trace mask is non-zero, or when the call to SIGERROR is other than the normal termination of the VM NFS server by an external interrupt.

6. In the event of a programming logic error in the NFS server machine, facilities exist to enable a virtual machine dump (in VMDUMP format) to be taken. During abnormal termination or other error events, the default handling is for no storage dumps to be taken. To enable the taking of a dump, simply create and place a file with the following file name and file type on the A-disk of the NFS server virtual machine.

   ```
   VMNFS DUMP_REQ
   ```

   The first line of this file should contain the mask value of X'FFFFFFFF'. This will enable dumps for all classes of errors within the NFS server machine. If you are experiencing problems with NFS and have called the IBM Software Support Center for assistance, it is likely that you may be requested to produce a storage dump in the above mentioned manner to help aid with problem isolation. Comments may be added to the VMNFS DUMP_REQ file to keep as a history log. Comments may be in any form, as long as the first line contains the mask value.

NFS Protocol

The VM NFS server supports NFS protocol, program 100003, at the Version 2 and Version 3 levels. These are described by RFCs 1094 and 1813.

Mount Protocol

The VM NFS server supports MOUNT protocol, program 100005, at the Version 1 and Version 3 levels. These are also described by RFCs 1094 and 1813.

In addition to procedures 0-5 described in the RFCs, VM defines Mount protocol procedure 6 for MOUNTPW.

PCNFSD Protocol

The VM NFS server supports PCNFS protocol, program 150001, at the Version 1 and Version 2 levels. Only procedures PCNFSD_NULL (0) and PCNFSD_AUTH (Version 1 – 2, Version 2 – 13) are supported.

Activating Traces

In the NFS server virtual machine, tracing is activated by specifying either the **G** or **g** option on the :parms tag for VMNFS in the DTCPARMS file.

```
:parms.G
```

The following demonstrates the use of the trace option when used with the VMNFS command:

```
►►──VMNFS──G─────────────────────────────────────────────────►◄
```

```
►►──VMNFS──g─────────────────────────────────────────────────►◄
```

Note that the trace option is not delimited from the command by a left parenthesis.

The trace output is written to the VMNFS LOG file (on the server's A disk). The log file contains the calls and responses processed by VMNFS. Each entry written to the log file consists of the following two records:

- a header record specifying the client address and message length
- a record containing the actual message.

The log file is normally not closed until the server has been terminated. Once started, VMNFS waits for client requests, but the program may be terminated manually by an external interrupt created by the CP command EXTERNAL. It is possible to close the log file without terminating the VM NFS server by using a CMS command sent by an authorized user to the VMNFS virtual machine with SMSG:

```
SMSG VMNFS M CMS FINIS * * A
```

The VMNFS module also supports the use of a **D** or **d** option. The tracing provided by **d** is a superset of that provided by **g**, therefore, there is no requirement to specify both. This option causes various debugging messages to be written to the server's spooled console, and generates the same log file on disk as the **g** option. These messages indicate the results of activities performed by the NFS server, such as task dispatching operations. There can be many messages during normal operation of the VM NFS server, which can make it tedious to locate more interesting messages among the mass of debug messages. The D option is therefore most useful in circumstances where it is necessary to learn whether any client requests are received by the server, because this option causes console output for each such request.

The VMNFS LOG file generated by running with tracing activated contains binary data. A utility program, PRINTLOG, is provided to format the VMNFS LOG file into a VMNFS PRINT file, suitable for examination. A sample of formatted output is shown in Figure 84 on page 167.

Trace Tables

An internal trace facility is called from various places in the code to record information about the details of processing client requests. Data is written to a table in storage, with enough descriptive information included to make it possible to extract and format useful information without many dependencies on the actual storage address at which the program is loaded or on the particular order or location of the routines that are combined to produce the executable file.

There are actually two internal trace tables. The original one contains fixed-length entries and is located from pointers that have the external name TRACEPTR. The newer facility is more versatile, and uses variable-length entries. These features gave rise to the name TRACEV. The external name TRACEVAD identifies a pointer to a structure defining the newer trace table.

The original trace routine is still called, but from fewer locations because many of the original calls to "trace" were changed to call "tracev" in later releases. Both of the internal trace tables share the characteristic that they wrap: new information is written over old data when the capacity of the table is exceeded.

In order to make better use of the available space in the tracev table, calls are assigned to various classes and a mask is used to select which classes of call will result in trace data actually recorded in the table. Calls to tracev that specify classes that have zero mask bits return immediately and no data is saved as a result of those calls. This mask is a 32-bit field that has the external name TRACEV@M (the internal name is tracev_m). The mask is zero by default, in order to eliminate most of the trace overhead in the majority of times when no one is interested in the data.

The command TWRITE may be sent by CP SMSG to the VMNFS virtual machine to request it to write the current contents of the trace tables to a disk file or SFS directory. The default fileid for this file is TRACEV FILE A1, but another name may be specified in the TWRITE command. For example:

```
CP SMSG VMNFS M T DARK TDATA G
```

will write the file DARK TDATA G1. If a disk file with the specified (or default) name exists when the TWRITE command is issued, the old file is erased before the new data is written to disk. The TVPRINT Utility can be used to decode some of this file's data into a readable format.

There are several ways to set the tracev mask field. The command line option M may be used, or the mask field may be dynamically set during operation of the VM NFS server by use of a MASK command delivered using CP SMSG. The mask value 0xFFFFFFFF enables all tracing. See file TRACEV.H for trace classes and related information.

The default mask value may be changed by re-compiling the TRACEV.C file and rebuilding the VMNFS executable file. For example:

```
CC TRACEV C (DEFINE TMASK(0XFFFFFFFF)
```

will enable all tracing by default.

The trace data file (for example, TRACEV DATA) contains binary information. Care must be taken when transmitting it so that no data transformations are performed by code-sensitive programs such as mail processing agents.

Trace Output

The VMNFS PRINT file provides complete information about messages that have been sent and received. This information includes the name of the programs and procedures called and the associated versions, IP addresses, and ports used. The file includes authentication information (passwords) used by clients to identify themselves to the NFS server, and therefore may be subject to local security controls pertaining to such information.

Figure 84 on page 167 shows a sample of an NFS trace of a mount request that is rejected because of invalid authentication data. When the NFS server starts, a series of 8 messages are exchanged with the Portmapper. These messages are written to the log file in a somewhat different format than transactions with NFS clients, but the PRINTLOG program understands this. There are two messages sent to Portmap to unregister the NFS and MOUNT programs (in case VMNFS is restarting), then two messages to register these programs. Each call message is followed by its reply message. Only the last of these 4 interactions (messages 7 and 8) are shown in this sample.

Some of the message fields are described below to assist the reader in understanding the format of the VMNFS PRINT file. For a complete description of the NFS message formats, consult RFC 1057 and RFC 1094 (see Chapter 10, “RPC Programs” on page 143).

- For message 9:

Offset	Field Description
X'0000'	XID, X'290D3D97'
X'0004'	X'00000000' This is a call message.
X'0008'	RPC version 2.
X'000C'	Program number, X'186A5'=100005 (MOUNT).
X'0010'	Program version 1.
X'0014'	Procedure number 6 (a procedure added to the MOUNT program so that VMNFS can service the mountpw request from a client).
X'0018'	Credential authentication type is 0 (null).
X'001C'	Length of authentication data is zero.
X'0020'	Verifier authentication type is 0 (null).
X'0024'	Length of authentication data is zero.
X'0028'	Counted string argument length is 19 characters.
X'002C'	Start of string data.

- For message 10:

Offset	Field Description
X'0000'	XID
X'0004'	This is a reply message.
X'0008'	Reply status = accepted message.
X'000C'	RPC accepted message status = executed successfully.
X'0010'	Verifier authentication type 0 (null).
X'0014'	Authentication length is zero.
X'0018'	Value of the called procedure is zero, indicating successful execution.

- For message 11:

Offset	Field Description
X'0014'	Procedure number 1 (add mount)
X'0018'	Credential authentication type is 1 (Unix).
X'001C'	Length of authentication data is 32 bytes.
X'0040'	Verifier authentication type is 0 (null).
X'0044'	Length of authentication data is zero.
X'0048'	Counted string argument length is 14 characters.
X'004C'	Start of string data.

- For message 12:

Offset	Field Description
X'0018'	Value of the called procedure is 13, NFSERR_ACCES (access denied).

```
  Sent to       014.000.000.000  port 111  length 56  time 811
Message number    7
 0000   00000004  00000000  00000002  000186A0     E.............f.E
A...............A
 0010   00000002  00000001  00000000  00000000     E...............E
A...............A
 0020   00000000  00000000  000186A3  00000002     E.........ft....E
A...............A
 0030   00000011  00000801                         E........       E
A........       A

234881024 111 28 811
Message number    8
 0000   00000004  00000001  00000000  00000000     E...............E
A...............A
 0010   00000000  00000000  00000001               E............   E
A............   A

  Received from 129.034.138.022  port 2298  length 64  time 973
  XID 290D3D97  program 100005  procedure 6
Message number    9
 0000   290D3D97  00000000  00000002  000186A5     E...p..........fvE
A).=............A
 0010   00000001  00000006  00000000  00000000     E...............E
A...............A
 0020   00000000  00000000  00000013  72657865     E...............E
A...........rexeA
 0030   63642E31  39312C70  3D726561  64697400     E.........../....E
Acd.191,p=readit.A

  Sent to       129.034.138.022  port 2298  length 28  time 973
  XID 290D3D97  reply_stat 0  accept_stat 0  NFS stat 0
Message number   10
 0000   290D3D97  00000001  00000000  00000000     E...p............E
A).=............A
 0010   00000000  00000000  00000000               E............   E
A............   A

  Received from 129.034.138.022  port 813  length 92  time 4
  XID 290223BE  program 100005  procedure 1
Message number   11
 0000   290223BE  00000000  00000002  000186A5     E.............fvE
A).#............A
 0010   00000001  00000001  00000001  00000020     E...............E
A.............. A
 0020   290C2594  00000006  6E667372  696F0000     E...m....>....?..E
A).%.....nfsrio..A
 0030   00000000  00000000  00000001  00000000     E...............E
A...............A
 0040   00000000  00000000  0000000E  72657865     E...............E
A...........rexeA
 0050   63642E31  39312C72  3D6E0000               E.........>..   E
Acd.191,r=n..   A

  Sent to       129.034.138.022  port 813  length 28  time 4
  XID 290223BE  reply_stat 0  accept_stat 0  NFS stat 13
Message number   12
 0000   290223BE  00000001  00000000  00000000     E...............E
A).#............A
 0010   00000000  00000000  0000000D               E............   E
A............   A
```

Figure 84. A Sample of an NFS Trace of a Bad Mount

Chapter 13. Remote Printing Traces

The following sections describe the tracing capabilities available in the client and server functions provided with the Remote Printing implementation in TCP/IP for VM.

Remote Printing Client Traces

The client interface to Remote Printing is through the following series of commands:

- **LPR** – Route a specific file to a designated, possibly remote, printer.
- **LPQ** – Interrogate the print queue on the designated printer.
- **LPRM** – Remove a job from the print queue on the designated printer.

Activating Remote Printing Client Traces

In the client virtual machine, tracing is activated by specifying the **TRACE** parameter in addition to the usual processing parameters on command invocation. The following demonstrates the use of the **TRACE** parameter for each of the client Remote Printing commands:

```
►►──LPR──fn──ft──┬────┬──┬───────────────┬──TRACE──────────────────────────►◄
                 └─fm─┘  └─print_options─┘
```

```
►►──LPQ──┬───────┬──┬──────────────┬──TRACE──────────────────────────►◄
         └─jobid─┘  └─printer_info─┘
```

```
►►──LPRM──┬───────┬──┬──────────────┬──TRACE──────────────────────────►◄
          └─jobid─┘  └─printer_info─┘
```

Note that the above examples are meant only to highlight the specification of the **TRACE** parameter. They are not meant to be all inclusive examples of the parameters available for use. Refer to *TCP/IP for VM: User's Guide* for information on the full parameter set available for the commands.

Remote Printing Client Trace Output

The output from the client traces shows the sequence of interactions with the Remote Printing server. Transferred data is not traced.

Figure 85 shows an example of output received from a client trace of the LPR command. Trace output from the other client commands is similar. In the trace, the output has been artificially separated to highlight the various processing sections involved during command execution.

```
---------- Section 1 ----------
lpr doit exec a (trace
Printer name from global variable PRINTER = "FSC3820"
Host name from global variable PRTHOST = "VM1"
lpr to printer "FSC3820" at host "VM1"
Requesting TCP/IP service at 06/04/97 on 13:34:26
Granted    TCP/IP service at 06/04/97 on 13:34:27
---------- Section 2 ----------
Resolving VM1 at 06/04/97 on 13:34:27
Host VM1 name resolved to 9.67.58.225 at 06/04/97 on 13:34:27
TCP/IP turned on.
Host "VM1" Domain "TCP.ENDICOTT.IBM.COM" TCPIP Service Machine TCPIP
Trying to open with local port 721 at 06/04/91 on 13:34:27
Connection open from local port 721 to foreign port 515 at 06/04/97 on 13:34:27
Control file name is cfA164VM1
Data file name is dfA164VM1
---------- Section 3 ----------
Sending command 2 argument: "FSC3820"
Command successfully sent
Receiving ACK
   Notification: Data delivered
   ConnState:    Open
ReceiveACK:  TRUE for byte value 00
Byte size check starts on 06/04/97 at 13:34:27
Byte size check ends   on 06/04/97 at 13:34:27
Send command    starts on 06/04/97 at 13:34:27
Sending command 3 argument: "405 dfA164VM1"
Command successfully sent
Receiving ACK
   Notification: Data delivered
   ConnState:    Open
ReceiveACK:  TRUE for byte value 00
Send command    ends   on 06/04/97 at 13:34:27
---------- Section 4 ----------
Send data       starts on 06/04/97 at 13:34:27
Send data       ends   on 06/04/97 at 13:34:27
Send ACK        starts on 06/04/97 at 13:34:27
Sending ACK
ACK successfully sent
Send ACK        ends   on 06/04/97 at 13:34:27
Receiving ACK
   Notification: Data delivered
   ConnState:    Open
ReceiveACK:  TRUE for byte value 00
Data file sent.
```

Figure 85 (Part 1 of 2). A Sample of an LPR Client Trace

```
---------- Section 5 ----------
Queuing control line "HVM1"
Queuing control line "PTCPMAINT"
Queuing control line "JDOIT.EXEC"
Queuing control line "CVM1"
Queuing control line "LTCPMAINT"
Queuing control line "fdfA164VM1"
Queuing control line "UdfA164VM1"
Queuing control line "NDOIT.EXEC"
Sending command 2 argument: "74 cfA164VM1"
Command successfully sent
Receiving ACK
   Notification: Data delivered
   ConnState:    Open
ReceiveACK:  TRUE for byte value 00

---------- Section 6 ----------
Control file sent
Sending ACK
ACK successfully sent
Receiving ACK
   Notification: Data delivered
   ConnState:    Open
ReceiveACK:  TRUE for byte value 00
Control file sent.

---------- Section 7 ----------
Sending ACK
ACK successfully sent
Receiving ACK
   Notification: Connection state changed
   NewState:     Receiving only
ReceiveACK:  TRUE for byte value 00
Connection closed.
```

Figure 85 (Part 2 of 2). A Sample of an LPR Client Trace

The following provides a brief description of each of the sections identified in the above sample output:

Section 1

The LPR command is issued to print the file “DOIT EXEC A.”

Since the invocation parameters did not include the target printer, printer and host names are resolved through GLOBALV calls.

The LPR module establishes a connection with the TCP/IP virtual machine, requesting TCP/IP services.

Section 2

The host name “VM1” is resolved to its IP address.

A connection to the Remote Printing server virtual machine (LPSERVE) is established. This server had previously performed a passive open on port 515. The source port will be in the range 721 to 731, inclusive.

Unique names for the control and data files to be shipped to the server are generated. These names will conform to a specific format as follows:

- will begin with “cfA” (control file) or “dfA” (data file)
- followed by a unique three digit number in range 000 - 999 (to be used as the job number for the print request)
- followed by the host name of the system which constructs the files.

Section 3

A “Receive a printer job” command (command code 2) is sent to the server, specifying the printer name “FSC3820.”

After successfully sending the command, the client waits for, and receives, the server's (positive) acknowledgement.

The client computes the size of the file to be printed (in octets) and sends a “Receive data file” subcommand (command code 3) to the server, specifying file size (405) and data file name (dfA164VM1).

After successfully sending the command, the client waits for, and receives, the server's (positive) acknowledgement.

Section 4

The client processes the entire data file, sending 405 octets to the server across the established connection.

When all data has been sent, an octet of binary zeros is sent as an ACK (indication) that the file being sent is complete.

After successfully sending the ACK, the client waits for, and receives, the server's (positive) acknowledgement.

Section 5

The client constructs a control file according to the standard format, computes its size in octets, and sends a “Receive control file” subcommand (command code 2) to the server, specifying file size (74) and control file name (cfA164VM1).

After successfully sending the command, the client waits for, and receives, the server's (positive) acknowledgement.

Section 6

The client processes the entire control file, sending 74 octets to the server across the established connection. Note that the trace line `Control file sent` (without a trailing period) is written out when the transfer of the control data is complete.

When all data has been sent, a byte (octet) of binary zeros is sent as an ACK (indication) that the file being sent is complete.

After successfully sending the ACK, the client waits for, and receives, the server's (positive) acknowledgement.

Completion of control file processing is signified by the trace line `Control file sent.` (with a trailing period).

Section 7

After transferring all of the data and control information, an octet of binary zeros is sent as a final ACK (indication) that the processing is complete.

After successfully sending the ACK, the client waits for, and receives, the server's (positive) acknowledgement.

The connection state changes from “Open” to “Receiving only” after the final ACK.

The connection with the server is subsequently closed, and the file transfer is considered complete.

Remote Printing Server Traces

The Remote Printing server is activated during processing performed in the LPSERVE virtual machine when its PROFILE EXEC executes the LPD command.

Activating Remote Printing Server Traces

In the server virtual machine, tracing is activated by one of the following mechanisms:

- specifying **TRACE** as a parameter on the LPD command invocation,
- including the **DEBUG** statement in the LPD CONFIG file, or
- by means of the **TRACE ON** command of the SMSG interface to the Remote Printing server.

Remote Printing Server Trace Output

The output from the server traces shows the sequence of interactions with the clients as well as server-specific processing. Transferred data is not traced.

Figure 86 shows an abridged example of output received from a server trace. In the trace, the output has been artificially separated to highlight the various processing sections involved during command execution. The first section deals with trace output pertaining to initialization processing. The remaining sections of the trace depict the server processing associated with the the corresponding LPR Client trace described previously.

```
---------- Section 1 ----------
IBM LPD Version V2R4 on 06/04/97 at 13:31:09

LPD starting with port 515
Starting TCP/IP service connection
TCP/IP turned on.
Host "VM1" Domain "TCP.ENDICOTT.IBM.COM" TCPIP Service Machine TCPIP
Host VM1 name resolved to 9.67.58.225
RSCS name is RSCS.
   LOCAL added with address 191
   FSC3820 added with address 191
   FSD3820 added with address 191
   FSE3820 added with address 191
   lp added with address 191
Host "RIOS" resolved to 9.67.30.50.  Printer name is "lp".
   PUNCH added with address 191
   ...End of Printer chain...
Passive open on port 515
06/04/97 13:31:10 Ready
```

Figure 86 (Part 1 of 4). A Sample of a Remote Printing Server Trace

```
---------- Section 2 ----------
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Connection state changed
New connection state Trying to open on connection 1 with reason OK.
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Connection state changed
New connection state Open on connection 1 with reason OK.
Passive open on port 515
Connection open.  Reading command.
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Data delivered
Timer cleared for connection 1
New command  2 data "FSC3820".
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification FSend response
---------- Section 3 ----------
Reading additional data on 1
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Data delivered
Timer cleared for connection 1
New subcommand  3 operands "405 dfA164VM1".
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification FSend response
Reading additional data on 1
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Data delivered
Timer cleared for connection 1
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Data delivered
Timer cleared for connection 1
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification FSend response
---------- Section 4 ----------
Reading additional data on 1
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Data delivered
Timer cleared for connection 1
New subcommand  2 operands "74 cfA164VM1".
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification FSend response
Reading additional data on 1
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Data delivered
Timer cleared for connection 1

     :        :           :      :      :         :      :

GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Data delivered
Timer cleared for connection 1
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification FSend response
```

Figure 86 (Part 2 of 4). A Sample of a Remote Printing Server Trace

```
---------- Section 5 ----------
Reading additional data on 1
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Data delivered
Timer cleared for connection 1
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Connection state changed
New connection state Sending only on connection 1 with reason OK.
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification FSend response
Closing connection 1
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Connection state changed
New connection state Connection closing on connection 1 with reason OK.
GetNextNote with ShouldWait of  TRUE
GetNextNote returns.  Connection 1 Notification Connection state changed
New connection state Nonexistent on connection 1 with reason OK.
End Connection 1 for OK.
---------- Section 6 ----------
06/04/91 13:34:42 Job 164 received         FSC3820  9.67.58.225
Job 164 added to work queue
06/04/91 13:34:42 Job 164 scheduled        FSC3820  9.67.58.225
Released storage at 00351000
ProcessWork starting on job queue
      Work Queue start
         164 JOBstartPRINTING
      Work Queue end
   Job 164 for FSC3820 dispatched in state JOBstartPRINTING
06/04/91 13:34:42 Job 164 printing         FSC3820  9.67.58.225
PRINTER 020 defined
Spooling 020 this way " TO TCPUSR5".
Tagging 020 with "BTP311S6 N23R1  ".
ProcessWork end with queue
      Work Queue start
         164 JOBcontinuePRINTING
      Work Queue end
---------- Section 7 ----------
GetNextNote with ShouldWait of FALSE
ProcessWork starting on job queue
      Work Queue start
         164 JOBcontinuePRINTING
      Work Queue end
   Job 164 for FSC3820 dispatched in state JOBcontinuePRINTING
flpNewBlock: State build      IsAtEof FALSE
flpNewBlock: State check last IsAtEof FALSE
flpNewBlock: State call       IsAtEof FALSE

    :          :    :            :      :

flpNewBlock: State build      IsAtEof FALSE
flpNewBlock: State check last IsAtEof TRUE
flpNewBlock: State call       IsAtEof TRUE
06/04/91 13:34:47 Job 164 sent             FSC3820  9.67.58.225
ProcessWork end with queue
      Work Queue start
         164 JOBfinishPRINTING
      Work Queue end
GetNextNote with ShouldWait of FALSE
```

Figure 86 (Part 3 of 4). A Sample of a Remote Printing Server Trace

```
---------- Section 8 ----------
ProcessWork starting on job queue
      Work Queue start
         164 JOBfinishPRINTING
      Work Queue end
   Job 164 for FSC3820 dispatched in state JOBfinishPRINTING
Job 164 removed from work queue
06/04/91 13:34:47 Job 164 purged           FSC3820  9.67.58.225
ProcessWork end with queue
      Work Queue start
      Work Queue end
GetNextNote with ShouldWait of  TRUE
```

Figure 86 (Part 4 of 4). A Sample of a Remote Printing Server Trace

The following provides a brief description of each of the phases identified in the above sample output:

Section 1

The Remote Printing server announces the start of initialization activities.

The server establishes a connection with the TCP/IP virtual machine, requesting TCP/IP services.

The host name “VM1” is resolved to its IP address.

The configuration file is processed to build the control tables representing the supported printers (and possibly punches). Note that system names are resolved to their respective IP addresses at initialization time.

The server records the date and time that it completes initialization plus the port it is listening on in the console log.

Section 2

The server establishes a connection with the client requesting remote printing services.

A “Receive a printer job” command (command code 2) is received from the client with a specified printer name of “FSC3820.”

The server validates the printer name and its availability and sends an acknowledgement to the client.

Section 3

A “Receive data file” subcommand (command code 3) is received from the client with a specified file size of 405 octets and a data file name of “dfA164VM1.”

The server acknowledges the receipt of this subcommand from the client.

The 405 octets of data are received, followed by the receipt of an octet of binary zeros signifying the end of file transfer.

The server acknowledges the receipt of the “end of file” indicator from the client.

Section 4

A “Receive control file” subcommand (command code 2) is received from the client with a specified file size of 74 octets and a control file name of “cfA164VM1.”

The server acknowledges the receipt of this subcommand from the client.

The 74 octets of data are received, followed by the receipt of an octet of binary zeros signifying the end of file transfer.

The server acknowledges the receipt of the “end of file” indicator from the client.

Section 5

A “final” octet of binary zeros is received from the client to signify the end of all data transmission.

The connection state is modified from an “Open” to a “Sending only” status.

The server acknowledges the receipt of the “end of transmission” indicator from the client.

The connection state is marked “Nonexistent” and the connection with the client is terminated, marking the completion of the “file transfer” portion of the operation.

Section 6

The received print job is placed onto the queue for the designated printer. The printer name was passed as an argument on the “Receive a printer job” command (FSC3820). The “job id” is taken from the arguments passed to the server on the “Receive data file” and “Receive control file” subcommands. The IP address of the system on which the printer is located was determined (and saved) during server initialization.

The placement of an entry on the printer queue triggers the ProcessWork routine to receive control.

The status of the job is modified from “scheduled” to “JOBstartPRINTING.”

A virtual printer is defined and initialized according to the parameters either passed with the print request or extracted from the configuration file entry for the target printer.

Section 7

The actual “printing” of the job is initiated and its status is modified from “JOBstartPRINTING” to “JOBcontinuePRINTING.”

The file to be printed is processed on a block-by-block basis. Note that the example shows an abridged version of the tracing for this phase of the operation.

When an end-of-file condition is encountered, the status is is modified from “JOBcontinuePRINTING” to “JOBfinishPRINTING.”

Section 8

The “JOBfinishPRINTING” status causes the job to be removed from the work queue and purged.

The virtual printer defined for processing the print request is detached.

A final interrogation of the work queue indicates that there is no more work to be performed.

The print server returns to a passive wait state, awaiting the next print request.

For additional information on the command codes and the format of the control file lines, see RFC 1179, *Line Printer Daemon Protocol*.

Chapter 14. Remote Execution Protocol Traces

The following sections describe the tracing capabilities available in the client and server functions provided with the Remote Execution Protocol implementation in TCP/IP for VM.

Remote Execution Protocol Client Traces

The client interface to the Remote Execution Protocol is through the REXEC command. This command provides the capability to execute a specified command on a foreign host and receive the results on the local host.

Activating Remote Execution Protocol Client Traces

In the client virtual machine, tracing is activated by specifying the **-d** parameter in addition to the usual processing parameters on command invocation. The following demonstrates the use of the **-d** parameter for the REXEC command:

```
►►──REXEC──┬────┬──┬────┬──┬──────────────┬──┬────┬──┬─────────────┬──────────►
           └─-?─┘  └─-d─┘  └─-t──timeout─┘  └─-n─┘  └─-l──userid─┘

           ┌─-s──512──┐
►──┬──────────────┬──┼──────────┼──foreignhost──command────────────────────►◄
   └─-p──password─┘  └─-s──port─┘
```

Specification of the **-d** parameter will cause the trace output to be written to the client's console. Note that the trace processing does not suppress passwords supplied with the command or extracted from a NETRC DATA file, so the resultant trace output file should be treated as "company confidential" material.

The above example is intended only to highlight the specification of the parameter necessary to activate tracing. Refer to the *TCP/IP User's Guide* for information on the usage of the other parameters.

Remote Execution Protocol Client Trace Output

Figure 87 shows an example of the output received from a client trace of the REXEC command, specifying a "q n" (Query Names) command to be executed on the remote host. The entered command and the response are highlighted in order to differentiate that data from the trace information.

```
rexec -d -l guest -p guest vm1 q n
parms is -d -l guest -p guest vm1 q n
Variables have the following assignments:
fhost   : vm1
userid  : guest
passwd  : guest
command : q n
calling GetHostResol with vm1
Connecting to vm1 , port REXEC (512)
```

Figure 87 (Part 1 of 2). A Sample of a Remote Execution Client Trace

```
Passive Conn - OK on local port          601
passive open complete on port            0
Active Conn - OK on local port          601
active  open complete on port            1
rexec invoked
sending: 601 guest guest q n
D2 len          20
getnextnote until DD
Connection state changed
Trying to open
Connection state changed
Open
Data delivered
Bytes in          1
Data delivered
Bytes in        374
OPERATOR - 601, NETVPPI  - DSC, GCS5     - DSC, GCS4     - DSC
GCS3     - DSC, GCS2     - DSC, GCS      - DSC, SQLDBA   - DSC
X25IPI   - DSC, TCPMAINT - 602, LPSERVE  - DSC, ADM_SERV - DSC
VMKERB   - DSC, VMNFS    - DSC, NAMESRV  - DSC, PORTMAP  - DSC
SMTP     - DSC, FTPSERVE - DSC, REXECD   - DSC, SNMPD    - DSC
SNMPQE   - DSC, TCPIP    - DSC, RXAGENT1 - DSC
VSM - TCPIP
Connection state changed
Sending only
 returning from REXEC_UTIL
rexec complete
```

Figure 87 (Part 2 of 2). A Sample of a Remote Execution Client Trace

Remote Execution Protocol Server Traces

The Remote Execution Protocol server (REXECD) is activated during processing performed in the server virtual machine when its PROFILE EXEC executes the REXECD command.

Activating Remote Execution Protocol Server Traces

In the server virtual machine, tracing is activated by specifying the **-d** parameter in addition to the usual processing parameters on command invocation. The following demonstrates the use of the **-d** parameter for the REXECD command:

```
                                              ┌──────────────┐       ┌─-t 240────────┐
►►──REXECD──┬─────┬──┬────┬──┬────┬───────────▼──────────────┴──┬────┴───────────────┴──►
            └─-?──┘  └─-d─┘  └─-r─┘  └─-s agent_id─┘              └─-t timeout─┘

   ┌─-e 512──────┐  ┌─-h 514──────┐
►──┴─────────────┴──┴─────────────┴───────────────────────────────────────────────────►◄
      └─-e port─┘      └─-h port─┘
```

Specification of the **-d** parameter will cause the trace output to be written to the server's console.

The above example is intended only to highlight the specification of the parameter necessary to activate tracing. Refer to the *TCP/IP Planning and Customization* for information on the usage of the other parameters.

Remote Execution Protocol Server Trace Output

Figure 88 shows an abridged example of the output received from a server trace. The section of the trace shown depicts the server processing which transpired when the “q n” command was issued from the client and correlates with the trace information from the client trace shown previously.

```
 .
 .
Connection: 0
Notification: Connection state changed
        New state: Trying to open
        Reason: OK
Connection: 0
Notification: Connection state changed
        New state: Open
        Reason: OK
Tcp passive open for rexec conn 2
Connection: 0
Notification: Data delivered
        Bytes delivered:  20
        Push flag:        1
active connection: 3using first free agent agent RXAGENT1  is free
                   cmd - MSG RXAGENT1  q n
len - 16
Notification: IUCV interrupt
        IUCV interrupt incountered at 160600
received IUCV interrupt - from user RXAGENT1
iucv type is - pending connectionNotification: IUCV interrupt
        IUCV interrupt incountered at 160600
received IUCV interrupt - from user
iucv type is - pending   (priority) msgclearing actconn 3
                   Notification: IUCV interrupt
        IUCV interrupt incountered at 160600
received IUCV interrupt - from user
iucv type is - sever connectionclose conn = 0close actconn 3
                   RXAGENT1  to fpool
                   clearing actconn 3
                   Connection: 0
Notification: Connection state changed
        New state: Receiving only
        Reason: OK
Connection: 3
Notification: Connection state changed
        New state: Receiving only
        Reason: OK
Connection: 0
Notification: Connection state changed
        New state: Nonexistent
        Reason: Foreign host aborted the connection
bye to conn = 0
destroy actconn 3
Connection: 3
Notification: Connection state changed
        New state: Nonexistent
        Reason: Foreign host aborted the connection
bye to conn = 3
 .
 .
```

Figure 88. A Sample of a Remote Execution Protocol Server Trace

Chapter 15. TFTP Client Traces

TCP/IP for VM implements a Trivial File Transfer Protocol (TFTP) client function. The client interface is through the TFTP command. This command provides a simple method to get files from, and send files to, a foreign host. TFTP cannot list directories and has no provision for user authentication. The following sections describe how to activate and interpret TFTP client traces.

Activating Traces

In the client virtual machine, tracing is activated (and deactivated) by means of the **TRACE** subcommand once a TFTP session has been established. The subcommand acts as a toggle switch to enable or disable the tracing of TFTP packets. When tracing is enabled, information is displayed about each TFTP packet that is sent or received.

Trace Output

Figure 89 on page 183 shows an example of a TFTP session that includes the output obtained from executing the TFTP **TRACE** subcommand. An explanation of the trace data format follows the example.

```
tftp elmer
Command:
trace
Packet tracing is enabled.
Command:
get config.sys config.sys
Sending:  ( 22) <RRQ> config.sys NETASCII
Received: (516) <DATA> Block Number = 1
Sending:  (  4) <ACK> Block Number = 1
Received: (516) <DATA> Block Number = 2
Sending:  (  4) <ACK> Block Number = 2
Received: (516) <DATA> Block Number = 3
Sending:  (  4) <ACK> Block Number = 3
Received: (111) <DATA> Block Number = 4
Sending:  (  4) <ACK> Block Number = 4
1643 bytes transferred in 4.825 seconds. Transfer rate 0.341 Kbytes/sec.
Command:
get startup.cmd startup.cmd
Sending:  ( 23) <RRQ> startup.cmd NETASCII
Received: ( 36) <DATA> Block Number = 1
Sending:  (  4) <ACK> Block Number = 1
32 bytes transferred in 3.399 seconds. Transfer rate 0.009 Kbytes/sec.
Command:
get autoexec.bat autoexec.bat
Sending:  ( 24) <RRQ> autoexec.old NETASCII
Received: (140) <DATA> Block Number = 1
Sending:  (  4) <ACK> Block Number = 1
136 bytes transferred in 4.475 seconds. Transfer rate 0.030 Kbytes/sec.
Command:
quit
Ready; T=0.14/0.36 17:54:57
```

Figure 89. A Sample of a TFTP Client Trace

All trace entries for TFTP have the same general format:

direction (size) kind per-packet-information

where:

Field	Description
direction	Is either *Sending* or *Received*.
(size)	Is the number of bytes in the packet.
kind	Is the type of TFTP packet. The TFTP packet types are: RRQ Read request WRQ Write request Data Data packet ACK Acknowledgement packet Error Error packet.
per-packet-information	Is other data contained in the packet. The type of information displayed about each packet is: RRQ Foreign filename, transfer mode WRQ Foreign filename, transfer mode Data Block number ACK Block number Error Error number, error text (if any).

Chapter 16. TFTPD Traces

TCP/IP for VM implements a Trivial File Transfer Protocol Daemon (TFTPD) function. The daemon interface is through the TFTPD command. The following sections describe how to activate and interpret TFTPD traces.

Activating Traces

In the daemon virtual machine, tracing is activated (and deactivated) by means of the **TRACE** subcommand once a TFTPD session has been established. The subcommand acts as a toggle switch to enable or disable the tracing of TFTPD operations. When tracing is enabled, information is displayed about major operation checkpoints. For example, trace output is created when read requests are received and complete or when errors are detected.

You can also use the **TRACE** operand on the TFTPD command to enable the tracing of TFTPD operations.

Trace Output

Figure 90 shows an example of a TFTPD session that includes the output obtained from executing the TFTPD **TRACE** subcommand. An explanation of the trace data format follows the example.


```
TRACE
TFTPD Ready;
1000 9.100.20.99     1685 (........) 05/15/97 09:27:50 READ REQUEST ACCEPT SENT
     O M 8192     /QIBM/ProdData/NetworkStation/kernel
1000 9.100.20.43     1065 (........) 05/15/97 09:28:11 READ REQUEST ACCEPT SENT
     O H 8192     /QIBM/ProdData/NetworkStation/kernel
1500 9.100.20.99     1685 (........) 05/15/97 09:28:12 READ COMPLETED
     PKTS=252      FILE SIZE=2044868
1000 9.100.20.99     1662 (........) 05/15/97 09:28:20 READ REQUEST ACCEPT SENT
     O M 8192     /QIBM/ProdData/NetworkStation/StationConfig/standard.nsm
1000 9.100.20.99     1663 (........) 05/15/97 09:28:20 READ REQUEST ACCEPT SENT
     O M 8192     /QIBM/ProdData/NetworkStation/StationConfig/required.nsm
1500 9.100.20.99     1663 (........) 05/15/97 09:28:21 READ COMPLETED
     PKTS=3        FILE SIZE=1916
1000 9.100.20.99     1664 (........) 05/15/97 09:28:21 READ REQUEST ACCEPT SENT
     O M 8192     /QIBM/ProdData/NetworkStation/StationConfig/control.nsm
1000 9.100.20.99     1665 (........) 05/15/97 09:28:21 READ REQUEST ACCEPT SENT
     O M 8192     /QIBM/ProdData/NetworkStation/SysDefaults/ibmwall.xbm
1500 9.100.20.99     1665 (........) 05/15/97 09:28:21 READ COMPLETED
     PKTS=3        FILE SIZE=3041
1000 9.100.20.99     1666 (........) 05/15/97 09:28:21 READ REQUEST ACCEPT SENT
     O M 8192     /QIBM/ProdData/NetworkStation/SysDefaults/ibmwall.xbm
1500 9.100.20.99     1666 (........) 05/15/97 09:28:21 READ COMPLETED
     PKTS=3        FILE SIZE=3041
1500 9.100.20.99     1664 (........) 05/15/97 09:28:21 READ COMPLETED
     PKTS=3        FILE SIZE=1042
4000 9.100.20.99     1667 (........) 05/15/97 09:28:21 FILE NOT VALID RESPONSE
     /QIBM/ProdData/NetworkStation/StationConfig/hosts.nsm
1500 9.100.20.99     1662 (........) 05/15/97 09:28:21 READ COMPLETED
     PKTS=3        FILE SIZE=174
1000 9.100.20.99     1678 (........) 05/15/97 09:28:26 READ REQUEST ACCEPT SENT
     O M 8192     /QIBM/ProdData/NetworkStation/mods/libxm.nws
1500 9.100.20.43     1065 (........) 05/15/97 09:28:36 READ COMPLETED
     PKTS=252      FILE SIZE=2044868
1500 9.100.20.99     1678 (........) 05/15/97 09:28:36 READ COMPLETED
     PKTS=155      FILE SIZE=1252482
1000 9.100.20.99     1680 (........) 05/15/97 09:28:36 READ REQUEST ACCEPT SENT
     O M 8192     /QIBM/ProdData/NetworkStation/mods/actlogin.nws
1000 9.100.20.99     1681 (........) 05/15/97 09:28:37 READ REQUEST ACCEPT SENT
     O M 8192     /QIBM/ProdData/NetworkStation/mods/export.nws
1500 9.100.20.99     1681 (........) 05/15/97 09:28:37 READ COMPLETED
     PKTS=7        FILE SIZE=36671
5100 9.100.20.99     1680 (........) 05/15/97 09:28:38 ERROR DATAGRAM RECEIVED
     ERROR=0 File read terminated by client
```

Figure 90. A Sample of a TFTPD Client Trace

Formats of TFTPD Trace Records

TFTPD trace entries identify 5 basic events and TCP/IP errors:

- Acceptance of a read or write request
- Resending of packets due to a timeout
- Dropping of a client due to resend limit being exceeded
- Sending or reception of error packets
- Socket related errors.

The first line of the trace entry contains:

- A 4 digit trace code
- A description of the trace code
- Time and date stamp and
- Client identification information (when the entry relates to a client). This can include:
  - IP address of the client

– Port number used by the client
– User ID associated with the client.

Depending upon the trace entry, additional lines of information may be displayed; such lines are indented under the first line.

The following example shows the format of the first line of a client related trace entry.

```
code xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss description of trace code
```

where:

code
: is a 4 digit trace code.

xxx.xxx.xxx.xxx
: is the IP address of the client in dotted decimal notation.

port
: is the port that the client is using.

userid
: is the user ID associated with the IP address; This association is determined by the TFTPD USERLIST file. If the client IP address is not listed in this file, then "........" is displayed.

mm/dd/yy
: is the date portion of the timestamp, where *mm* is the month, *dd* is the day and *yy* is the year.

hh:mm:ss
: is the time portion of the timestamp, where *hh* is the hour (in 24 hour format), *mm* is the minutes and *ss* is the seconds.

description of trace code
: is a 25 character description of the trace code

TFTPD Trace Codes:

The trace codes are:

1000 A read request was accepted.
1500 A read operation has completed.
2000 A write request was accepted.
2500 A write operation has completed.
3000 Timeout; a response was resent.
3500 Timeout; the timeout limit was reached, and the client dropped.
4000 A file not valid response was sent.
4100 A missing BLKSIZE response was sent.
4200 An Access Violation response was sent.
4300 A Bad XFER (transfer) Mode response was sent.
5000 A spurious ACK was received and has been ignored.
5100 An error datagram was received.
5200 An unknown datagram was received.
6100 An unexpected RECVFROM error occurred.
6200 An unexpected SENDTO error occurred.
6300 An unexpected SOCKINIT error occurred.
6301 An unexpected SOCKET error occurred.
6302 An unexpected IOCTL error occurred.
6303 An unexpected BIND error occurred.
6304 An unexpected SELECT error occurred.

6305 An unexpected CANCEL error occurred.

TFTPD Trace Entry: 1000

This trace code is the result of accepting a READ request.

```
1000 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss READ ACCEPTED DATA SENT
     x c  blksize       pathname
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186. The additional lines consist of:

x indicates the transfer mode, "N" for NETASCII and "O" for OCTET mode.

c is a hit or miss indicator, indicating whether the file was in cache when requested (a hit) or whether it had to be loaded (a miss).
"H" indicates that the file was in cache.
"M" indicates that the file was not in cache.

Note: A miss would be indicated for a file in cache that is marked for a drop by the **DROPFILE** subcommand. Subsequent read requests would require a new copy of the file to be obtained.

blksize
is the blocksize being used for the transfer.

pathname
is the name of the file being transferred.

TFTPD Trace Entry: 1500

This trace code is the result of receiving an ACK associated with a client read operation. The ACK indicates the client received the last packet of a transmitted file.

```
1500 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss READ COMPLETED
     PKTS=pkts     FILE SIZE=filesize
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186. The additional lines consist of:

pkts
number of packets sent.

filesize
size of the file in bytes.

TFTPD Trace Entry: 2000

This trace code is the result of accepting a WRITE request.

```
2000 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss WRITE ACCEPTED DATA SENT
     x    blksize       pathname
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186. The additional lines consist of:

x indicates the transfer mode, "N" for NETASCII and "O" for OCTET mode.

blksize
is the blocksize being used for the transfer.
pathname
is the name of the file that is being transferred.

TFTPD Trace Entry: 2500

This trace code is the result of receiving the DATA packet on a client write request.

```
2500 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss WRITE COMPLETED
     PKTS=pkts     FILE SIZE=filesize
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186. The additional lines consist of:

pkts
number of packets sent.
filesize
size of the file, in bytes.

TFTPD Trace Entry: 3000

This trace code is the result of determining that time has expired for a client to send or receive a packet so that the response must be resent.

```
3000 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss TIMEOUT - RESPONSE RESENT
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186.

TFTPD Trace Entry: 3500

This trace code is the result of the TFTPD daemon determining that a timeout occurred, but that the maximum number of resends was reached so the client was dropped.

```
3500 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss TIMEOUT - CLIENT DROPPED
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186.

TFTPD Trace Entry: 4000

This trace code is the result of the TFTPD daemon determining that the file to be sent to the client was not valid.

```
4000 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss FILE NOT VALID RESPONSE
     pathname
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186. The additional lines consist of:

pathname
is the name of the file that was not valid.

TFTPD Trace Entry: 4100

This trace code is the result of the TFTPD daemon receiving a request which contained the BLKSIZE parameter, but no value for that parameter.

```
4100 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss MISSING BLKSIZE RESPONSE
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186.

TFTPD Trace Entry: 4200

This trace code is the result of the TFTPD daemon receiving a read request for a file that the client was not permitted to access.

```
4200 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss ACCESS VIOLATION RESPONSE
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186.

TFTPD Trace Entry: 4300

This trace code is the result of the TFTPD daemon receiving a READ or WRITE request with the transfer mode parameter specified, but not valid.

```
4300 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss BAD XFER MODE RESPONSE
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186.

TFTPD Trace Entry: 5000

This trace code is the result of the TFTPD daemon receiving an unexpected ACK which it ignored.

```
5000 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss SPURIOUS ACK IGNORED
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186.

TFTPD Trace Entry: 5100

This trace code is the result of the TFTPD daemon receiving an error datagram from a client.

```
5100 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss ERROR DATAGRAM RECEIVED
     ERROR=errnum errdesc
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186. The additional lines consist of:

errnum
: is the error number received from the client.

errdesc
: is the error description sent by the client in the error datagram.

TFTPD Trace Entry: 5200

This trace code is the result of the TFTPD daemon receiving an unknown datagram.

```
5200 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss UNKNOWN DATAGRAM RECEIVED
```

The first line of the entry was explained in “Formats of TFTPD Trace Records” on page 186.

TFTPD Trace Entry: 6100

This trace code is the result of the TFTPD daemon encountering an unexpected error on a SOCKET RECVFROM operation.

```
6100                                 mm/dd/yy hh:mm:ss BAD RECVFROM ERROR
     RC=rc    ERRNO=errno
```

The first line of the entry was explained in “Formats of TFTPD Trace Records” on page 186. The additional lines consist of:

rc is the return code set by the RECVFROM function.

errno
is the error number set by the RECVFROM function.

TFTPD Trace Entry: 6200

This trace code is the result of the TFTPD daemon encountering an unexpected error on a SOCKET SENDTO operation.

```
6200 xxx.xxx.xxx.xxx port (userid  ) mm/dd/yy hh:mm:ss BAD SENDTO ERROR
     RC=rc    ERRNO=errno
```

The first line of the entry was explained in “Formats of TFTPD Trace Records” on page 186. The additional lines consist of:

rc is the return code set by the SENDTO function.

errno
is the error number set by the SENDTO function.

TFTPD Trace Entry: 6300

This trace code is the result of the TFTPD daemon encountering an unexpected error on a SOCKET initialization operation.

```
6300                                 mm/dd/yy hh:mm:ss BAD SOCKINIT ERROR
     RC=rc    REASON=reason   SOCKETS=socket
```

The first line of the entry was explained in “Formats of TFTPD Trace Records” on page 186. The additional lines consist of:

rc is the return code set by the Socket Initialize function.

reason
is the reason code set by the Socket Initialize function.

socket
is the socket number (if any) returned by the Socket Initialize function.

TFTPD Trace Entry: 6301

This trace code is the result of the TFTPD daemon encountering an unexpected error on a SOCKET SOCKET operation.

```
6301                              mm/dd/yy hh:mm:ss BAD SOCKET ERROR
     SOCKET=socket       ERRNO=errno
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186. The additional lines consist of:

socket
is the socket number.

errno
is the error number set by the SOCKET function.

TFTPD Trace Entry: 6302

This trace code is the result of the TFTPD daemon encountering an unexpected error on a SOCKET IOCTL operation.

```
6302                              mm/dd/yy hh:mm:ss BAD IOCTL ERROR
     RC=rc     ERRNO=errno
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186. The additional lines consist of:

rc is the return code set by the IOCTL function.

errno
is the error number set by the IOCTL function.

TFTPD Trace Entry: 6303

This trace code is the result of the TFTPD daemon encountering an unexpected error on a SOCKET BIND operation.

```
6303                              mm/dd/yy hh:mm:ss BAD BIND ERROR
     RC=rc     ERRNO=errno
```

The first line of the entry was explained in "Formats of TFTPD Trace Records" on page 186. The additional lines consist of:

rc is the return code set by the BIND function.

errno
is the error number set by the BIND function.

TFTPD Trace Entry: 6304

This trace code is the result of the TFTPD daemon encountering an unexpected error on a SOCKET SELECT operation.

```
6304                                 mm/dd/yy hh:mm:ss BAD SELECT ERROR
     RC=rc     ERRNO=errno
```

The first line of the entry was explained in “Formats of TFTPD Trace Records” on page 186. The additional lines consist of:

rc is the return code set by the SELECT function.

errno
is the error number set by the SELECT function.

TFTPD Trace Entry: 6305

This trace code is the result of the TFTPD daemon encountering an unexpected error on a SOCKET CANCEL operation.

```
6305                                 mm/dd/yy hh:mm:ss BAD CANCEL ERROR
     RC=rc     ERRNO=errno
```

The first line of the entry was explained in “Formats of TFTPD Trace Records” on page 186. The additional lines consist of:

rc is the return code set by the CANCEL function.

errno
is the error number set by the CANCEL function.

Chapter 17. BOOT Protocol Daemon (BOOTPD) Traces

TCP/IP for VM implements the BOOTP daemon to respond to client requests for boot information using information contained in a BOOTP machine file. The daemon interface is through the BOOTPD command. The following sections describe how to activate and interpret BOOTPD traces.

Activating Traces

In the daemon virtual machine, tracing is activated (and deactivated) by means of the TRACE command once the BOOTP daemon has been installed and a BOOTPD session has been established. The subcommand acts as a toggle switch to enable or disable tracing of BOOTPD operations. When tracing is enabled, information is displayed about each BOOTPD packet that is sent or received.

You can also use the **TRACE** operand on the BOOTPD command to enable tracing of BOOTPD operations.

Trace Output

Figure 91 on page 196 shows an example of a BOOTPD session that includes the output obtained from executing the BOOTPD TRACE subcommand. An explanation of the trace data format follows the example.

```
TRACE
BOOTPD Ready;
9000 Time: 09:18:36.744586 ON 19970515
1100 FORWARDED REQUEST RECEIVED FROM 67 AT 9.100.20.110 THRU 9.100.30.75:
     OP      = 1        CIADDR  = 0.0.0.0
     HTYPE   = 6        YIADDR  = 0.0.0.0
     HLEN    = 6        SIADDR  = 0.0.0.0
     HOPS    = 1        GIADDR  = 9.100.20.110
     XID     = 000001BE CHADDR  = 0000E5E82CFF
     SNAME   =
     FILE    =
  OPTIONS    = 638253632B0E49424D414354205620312E302E30FF00000000200011FFFA81440
               00090000000000000000000FFFB367E0000000081007DE8CC000000FF
3100 REPLYING TO GATEWAY BY 67 AT 9.100.20.110 THRU 9.100.30.75
     OP      = 2        CIADDR  = 0.0.0.0
     HTYPE   = 6        YIADDR  = 9.100.20.43
     HLEN    = 6        SIADDR  = 9.100.30.75
     HOPS    = 1        GIADDR  = 9.100.20.110
     XID     = 000001BE CHADDR  = 0000E5E82CFF
     SNAME   =
     FILE    = /QIBM/ProdData/NetworkStation/kernel
  OPTIONS    = 638253630104FFFFFF000204FFFFB9B00304096414FD04040964144B050409641
               9FC0604096414FC0F10656E6469636F74742E69626D2E636F6D11012FFF
9000 Time: 09:21:27.423501 ON 19970515
1100 FORWARDED REQUEST RECEIVED FROM 67 AT 9.100.20.110 THRU 9.100.30.75:
     OP      = 1        CIADDR  = 0.0.0.0
     HTYPE   = 6        YIADDR  = 0.0.0.0
     HLEN    = 6        SIADDR  = 0.0.0.0
     HOPS    = 1        GIADDR  = 9.100.20.110
     XID     = 000001BD CHADDR  = 0000E5E8DC61
     SNAME   =
     FILE    =
  OPTIONS    = 638253632B0E49424D414354205620312E302E30FF00000000200011FFF8F3380
               00090000000000000000000FFFB367E0000000081007DE8CC000000FF
3100 REPLYING TO GATEWAY BY 67 AT 9.100.20.110 THRU 9.100.30.75
     OP      = 2        CIADDR  = 0.0.0.0
     HTYPE   = 6        YIADDR  = 9.100.20.99
     HLEN    = 6        SIADDR  = 9.100.30.75
     HOPS    = 1        GIADDR  = 9.100.20.110
     XID     = 000001BD CHADDR  = 0000E5E8DC61
     SNAME   =
     FILE    = /QIBM/ProdData/NetworkStation/kernel
  OPTIONS    = 638253630104FFFFFF000204FFFFB9B00304096414FD04040964144B050409641
               9FC0604096414FC0F10656E6469636F74742E69626D2E636F6D11012FFF
```

Figure 91. A Sample of a BOOTPD Client Trace

BOOTPD Trace Records

BOOTPD trace entries identify 5 basic events:

- Time at which BootPD began processing a set of requests
- Reception of a datagram from a client or gateway
- Declining to respond to a client or gateway due to some error or limit
- Forwarding of a request to another BootP daemon
- The attempt to respond to a client or gateway.

BOOTPD Trace Record Format

The first line of trace entry consists of a trace code followed by a description of the event along with other pertinent information. Additional lines of information may be displayed, indented under the first line.

BOOTPD Trace Codes

The trace codes are:

1000 Received a request sent by a client

1100 Received a request which was forwarded by a BootP daemon

1900 Unrecognized request was received; the opcode was neither request or reply.

3000 A BootP reply was sent to a client

3100 A BootP reply was sent to another BootP daemon, to be passed to a client.

32xx Request is being forwarded to another BootP daemon. The **xx** subcodes indicate the reasons for forwarding:

- **01** Forwarding was specified, but no entry exists in the machine table.
- **02** Always forward was specified.
- **03** Client specified a server to which to forward the request.
- **00** Reason for forwarding was not known.

40xx The BootP daemon is declining to respond to a request. The **xx** subcodes that follow indicate the reasons for declining to respond:

- **01** Entry was not found in the machine table.
- **02** Request received on an adapter that was partially excluded, for which the entry matches the exclusion criteria.
- **03** Unrecognized packet opcode was received
- **04** Could not forward because the hop count expired.
- **05** Could not determine the client IP address.
- **06** Could not determine the bootfile pathname.
- **07** Target server is on the same cable.
- **08** Unable to determine the adapter over which to reply.
- **00** Reason for declining is not known.

9000 Time Stamp, including the time and date in standard format.

Trace events which relate to the transmission of BOOT requests or replies, include information about the packet.

```
OP     = opcode    CIADDR  = ipaddr
HTYPE  = htype     YIADDR  = ipaddr
HLEN   = hlen      SIADDR  = ipaddr
HOPS   = hops      GIADDR  = ipaddr
XID    = xid       CHADDR  = chaddr
SNAME  = servname
FILE   = bootfile
VEND   = venddata
```

where

**OP = ***opcode*
: indicates the operation code: 1 for a request or 2 for a reply.

**CIADDR = ***ipaddr*
: indicates the client IP address, if specified by the client.

**HTYPE = ***htype*
: indicates the network hardware type.

**YIADDR = ***ipaddr*
: indicates the IP address of the client.

**HLEN = ***hlen*
: indicates the length of the hardware address.

SIADDR = *ipaddr*
indicates the Server IP address.

HOPS = *hops*
indicates the current hops count.

GIADDR = *ipaddr*
indicates the gateway IP address.

XID = *xid*
indicates the current transaction ID specified by the client.

CHADDR = *chaddr*
indicates the client hardware address. This field may be a maximum of 16 bytes long.

SNAME = *servname*
indicates the Server Host Name. This field may be a maximum of 64 bytes long.

FILE = *bootfile*
indicates the boot file name. This field may be a maximum of 128 bytes long.

VEND = *venddata*
indicates the current contents of the vendor-specific area. This field may be a maximum of 64 bytes long.

Chapter 18. Dynamic Host Configuration Protocol Daemon (DHCPD) Traces

TCP/IP for VM implements the DHCP daemon to respond to client requests for boot information using information contained in a DHCP machine file. The daemon interface is through the DHCPD command and DHCPD subcommands. The following sections describe how to activate and interpret DHCPD traces.

Activating Traces

In the daemon virtual machine, tracing is activated (and deactivated) by means of the TRACE subcommand once the DHCPD daemon has been installed and a DHCPD session has been established. The TRACE subcommand acts as a toggle switch to enable or disable tracing of DHCPD operations. When tracing is enabled, information is displayed about each DHCPD packet that is sent or received.

You can also use the **TRACE** operand on the DHCPD command to enable tracing of DHCPD operations.

Trace Output

Figure 92 shows an example of a DHCPD session that includes the output obtained from executing the DHCPD TRACE subcommand. An explanation of the trace data format follows the example.

```
9000 TIME: 13:58:46.115502 ON 19970819
1100 FORWARDED REQUEST RECEIVED FROM 67 AT 9.100.20.88 THRU 9.100.30.75:
     OP      = 1        CIADDR  = 9.100.20.126    DHCPTYPE = DHCPDISCOVER
     HTYPE   = 6        YIADDR  = 0.0.0.0
     HLEN    = 6        SIADDR  = 0.0.0.0
     HOPS    = 1        GIADDR  = 9.100.20.88
     SECS    = 100      FLAGS   = 0
     XID     = 00000A56 CHADDR  = 0000E5E83CC0
     SNAME   =
     FILE    =
  OPTIONS    = 638253633501013902024040D0C49424D4E534D20312E302E303C1349424D204E6
               574776F726B2053746174696F6EFF
5300 ICMP ECHO REQUEST TO 9.100.20.126 THRU 9.100.30.75
9000 TIME: 13:58:51.669942 ON 19970819
5000 ICMP TIMER EXPIRED
3100 REPLYING TO GATEWAY BY 67 AT 9.100.20.88 THRU 9.100.30.75
     OP      = 2        CIADDR  = 0.0.0.0         DHCPTYPE = DHCPOFFER
     HTYPE   = 6        YIADDR  = 9.100.20.126
     HLEN    = 6        SIADDR  = 9.100.30.75
     HOPS    = 0        GIADDR  = 9.100.20.88
     SECS    = 0        FLAGS   = 0
     XID     = 00000A56 CHADDR  = 0000E5E83CC0
     SNAME   =
     FILE    =
  OPTIONS    = 638253633501023304000012C360409641E4B0104FFFFFF000204FFFFC7C0030
               4096414FD040C09641E4B09010A0D09820303050409821 9FC0604098219FC0C05
               4144414D470F10656E6469636F74742E69626D2E636F6D3A04000000963B04000
               000FF420747444C564D4B3443242F5149424D2F50726F64446174612F4E657477
               6F726B53746174696F6E2F6B65726E656CFF
```

Figure 92. A Sample of a DHCPD Client Trace

DHCPD Trace Records

DHCPD trace entries identify 6 basic events:

- Time at which DHCPD began processing a set of requests
- Reception of a datagram from a client or gateway
- Declining to respond to a client or gateway due to some error or limit
- Forwarding of a request to another DHCP/BootP daemon
- The attempt to respond to a client or gateway.
- Timer expiration and related activities

DHCPD Trace Record Format

The first line of trace entry consists of a trace code followed by a description of the event along with other pertinent information. Additional lines of information may be displayed, indented under the first line.

DHCPD Trace Codes

The DHCPD trace codes are:

1000 Received a request sent by a client

1100 Received a request that was forwarded by a BootP daemon

1900 Unrecognized request was received; the opcode was neither request or reply

3000 A BootP/DHCP reply was sent to a client

3100 A BootP/DHCP reply was sent to another BootP/DHCP daemon, to be passed to a client

32xx Request is being forwarded to another BootP/DHCP daemon. The *xx* subcodes that follow indicate the reasons for forwarding:

- **01** Forwarding was specified, but no entry exists in the machine table
- **02** Always forward was specified
- **03** Client specified a server to which to forward the request
- **00** Reason for forwarding was not known

40xx The DHCP daemon is declining to respond to a request. The *xx* subcodes that follow indicate the reasons for declining to respond:

- **01** Entry was not found in the machine table
- **02** Request received on an adapter that was partially excluded, for which the entry matches the exclusion criteria
- **03** Unrecognized packet opcode was received
- **04** Could not forward because the hop count expired
- **05** Could not determine the client IP address
- **06** Could not determine the bootfile pathname
- **07** Target server is on the same cable
- **08** Unable to determine the adapter over which to reply
- **09** SupportBootP is NO
- **10** Client is on a different subnet than the requested address
- **11** Requested address is restricted
- **12** Requested address is in use by another client
- **13** Internal error
- **14** Requested address is differs from machine table entry
- **15** No address is available
- **16** SupportUnlistedClients is NO
- **17** Client is not recognized
- **18** Client is not in a valid state
- **19** Request is not correctly formatted

20	Not selected as the server
21	Ignore any DHCPOffer messages
22	Address is being declined
23	Address is being released
24	Ignore any DHCPAck messages
25	Ignore any DHCPNack messages
26	Nothing possible for DHCPInform
27	Client statement specified: NONE
28	Waiting for ICMP Echo to complete
29	No address available
30	Client is on a subnet that is not supported
31	Unrecognized DHCP message type
00	Reason for declining is not known

5000 ICMP Timer expired with a response reply due

5100 Received an ICMP Echo reply

5300 Sending an ICMP Echo request

5500 Lease expired for an address

9000 Time Stamp, including the time and date in standard format

9900 Indicates the time when DHCPD concluded a particular unit of work

9950 The start time when a packet is attempted to be sent (in the SendPacket routine)

9951 The time when a packet send completes

Trace events which relate to the transmission of BOOT requests or replies, include information about the packet.

```
    OP      = opcode    CIADDR  = ipaddr  DHCPTYPE = msgtype
    HTYPE   = htype     YIADDR  = ipaddr
    HLEN    = hlen      SIADDR  = ipaddr
    HOPS    = hops      GIADDR  = ipaddr
    XID     = xid       CHADDR  = chaddr
    SNAME   = servname
    FILE    = bootfile
 OPTIONS    = optiondata
```

where

OP = *opcode*
indicates the operation code: 1 for a request or 2 for a reply.

CIADDR = *ipaddr*
indicates the client IP address, if specified by the client.

DHCPTYPE = *msgtype*
indicates the type of DHCP message. This parameter is shown only for DHCP protocol requests and replies.

HTYPE = *htype*
indicates the network hardware type.

YIADDR = *ipaddr*
indicates the IP address of the client.

HLEN = *hlen*
indicates the length of the hardware address.

SIADDR = *ipaddr*
indicates the Server IP address.

HOPS = *hops*
indicates the current hop count.

GIADDR = *ipaddr*
indicates the gateway IP address.

XID = *xid*
indicates the current transaction ID specified by the client.

CHADDR = *chaddr*
indicates the client hardware address. This field may be a maximum of 16 bytes long.

SNAME = *servname*
indicates the Server Host Name. This field may be a maximum of 64 bytes long. When “SNAME” is followed by “(O),” the field contains configuration options instead of only SNAME data. The data shown is a hexadecimal representation of the contents of the field.

FILE = *bootfile*
indicates the boot file name. This field may be a maximum of 128 bytes long. When “FILE” is followed by “(O),” the field contains configuration options instead of only FILE data. The data shown is a hexadecimal representation of the contents of the field.

OPTIONS = *optiondata*
indicates the current contents of the option area. This field may be a maximum of 64 bytes long.

Chapter 19. Hardware Trace Functions

This chapter describes PCCA and CETI devices. These devices support Local Area Networks (LANs).

You can trace LAN events in two ways: sniffer traces and CCW traces. Sniffer traces are attached directly to LANs, and are not dependent on the operating system. This chapter describes the CCW traces, which are the most common I/O traces implemented on IBM/370-based LANs.

PCCA Devices

The following sections describe the PCCA block structure, control messages, LAN messages, token-ring frames, and 802.2 LLC frames.

PCCA Block Structure

You should understand the PCCA block structure to interpret CCW traces. The PCCA block is a series of messages. Figure 93 shows the PCCA block structure. The first two bytes of each message is an integer value that determines the offset in the block of the next message. The last offset value, X'0000', designates the end of the message. The first two bytes of each data packet indicate the LAN and adapter numbers.

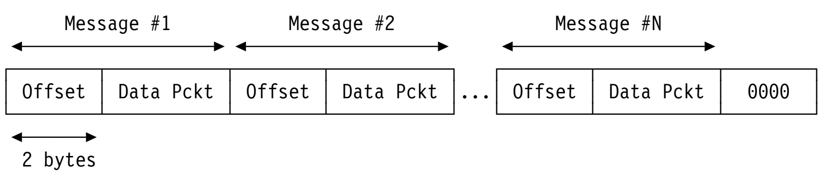


Figure 93. PCCA Block Structure

The PCCA block can be divided into two modes. Figure 94 shows a sample of a PCCA block with a series of messages. All highlighted halfwords in Figure 94 are offset fields in the block and denote the beginning of the new message. The last offset is X'0000'.

```
3C TRAPID ENTRY       **MP** 3C080000 01000000 E3C3D740 40404040           CP
       TRAPID = TCP, TRAPSET = IOSET, IODATA = 500
       TRAPTYPE = IO, USER = TCPIP, I/O OLD PSW = 0FC318
       DEVICE ADDRESS = 561, CSW = E05590C0 0C000000,
    -> CCW(1) = 01559028 240000AA, CCW ADDRESS = 5590B8, ** IDA **
       -> IDAW(1) = 14A020,
          DATA = 001C0000 01000000 00030100 00380000  *................*
                 0003D3C3 E2F100D7 C6B800D7 00380000  *..LCS1.PF..P....*
                 04000000 00030100 00380000 0003D3C3  *..............LC*
                 E2F100D7 C6B800D7 00540000 01000000  *S1.PF..P........*
                 00030200 00380000 0003D3C3 E2F100D7  *..........LCS1.P*
                 C6B800D7 00700000 04000000 00030200  *F..P............*
                 00380000 0003D3C3 E2F100D7 C6B800D7  *......LCS1.PF..P*
                 008C0000 01000000 00030201 00380000  *................*
                 0003D3C3 E2F100D7 C6B800D7 00A80000  *..LCS1.PF..P.y..*
                 04000000 00030201 00380000 0003D3C3  *..............LC*
                 E2F100D7 C6B800D7 0000               *S1.PF..P..      *
20 TOD STAMP          **MP** 20000000 00000000 A298CC1A 19EA1000           CP
```

Figure 94. A Sample of a PCCA Control Message Block

Control Messages

Control messages perform functions, such as starting the LAN and obtaining the hardware addresses of the LAN adapters. Figure 95 shows the structure of a PCCA control message, which has three fields.

The following are descriptions of the fields shown in Figure 95.

- Net Type (1 byte); X'00' for control messages

  This field helps to determine whether the packet is used for control or LAN operations.

- Adapter Number (1 byte); X'00', ignored for control messages
- Control field
  - Control command (1 byte)
    - X'00' Control Timing (sent by PCCA)
    - X'01' Start LAN
    - X'02' Stop LAN
    - X'04' LAN Stats
    - X'08' Shutdown
  - Control flags (1 byte)
    - X'00' From host
    - X'01' From PCCA
  - Control sequence (1 halfword)
  - Return code (1 halfword)
  - Net type_2 (1 byte)

    This is the net type of the adapter referred to by the control packet.
  - Adapter number_2 (1 byte)

    This is the number of the adapter referred to by the control packet.
  - Count (1 halfword)

    This occurs at startup. It is used for block size or a count of items in the data field (general control packet has 56 bytes, X'38').
  - Control reserved
  - Ignored (1 halfword)
  - Hardware address (6 bytes).

X'00'	X'00'	Control information

Figure 95. PCCA Control Message Structure

LAN Messages

LAN messages are used to send and receive LAN information or data to and from other LANs. PCCA LAN messages have three fields.

- Net Type (1 byte)
  - X'01' for Ethernet and 802.3
  - X'02' for token-ring

  - X'07' for FDDI networks

- Adapter Number (1 byte), X'00' or X'01'
- Data for the adapter.

Figure 96 shows a sample of a trace started by a CPTRAP IO command issued on a VM/SP6 system.

LAN No.	ADP No.	Data to send on a LAN

Figure 96. PCCA LAN Messages Structure

PCCA token-ring packets conform to the canonical 802 standards if they are specified in a PROFILE TCPIP file. If the PCCA packet is sent to a token-ring, use the 802.x or Ethernet layout.

Token-Ring Frames

Figure 97 shows the most common layout for token-ring packets. The components of the token-ring packet are:

- SD - Starting delimiter (1 byte)
- AC - Access control (1 byte)
- FD - Frame control (1 byte)
- DA - Destination address (6 bytes)
- SA - Source address (6 bytes)
- Data - Data field, including LLC frame (variable length)
- ED - End of frame (6 bytes).

Trace output does not include the starting delimiter or the end of frame.

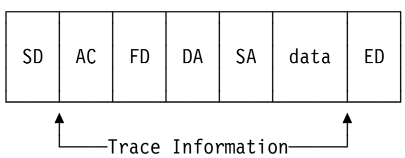


Figure 97. Common Layout of a Token-Ring Packet

CCW traces provide all fields from AC to Data fields for token-ring frames.

Note: When the first bytes of the source address are ORed with X'80', the frame contains routing information.

802.2 LLC Frame

An 802.2 LLC frame incorporates token-ring and 802.3 packets. This frame is a SNAP fashion frame for internet protocols and has the following layout:

1. DSAP and SSAP (2 bytes) X'AAAA' designates a SNAP frame
2. Control field (1 byte)
3. Origin/Port (1 byte)
4. Ether type, which has the values:
   - X'0800' IP protocol
   - X'0806' ARP protocol
   - X'8035' RARP protocol.

The data fields for the upper protocol follow the LLC frame.

CCW

There are three main sections of CCW trace output:

- CSW/CCW
- Hexadecimal representation of data
- EBCDIC character representation of data.

Table 22 lists the functions of the PCCA CCW codes.

Table 22. PCCA CCW Codes

Code	Function
X'01'	Write PCCA.
X'02'	Read PCCA.
X'03'	Nop PCCA.
X'04'	Sense PCCA.
X'C3'	Set X mode PCCA.
X'E4'	Sense ID PCCA.

The length of the CCW data field is usually X'5000' for runtime operations, and the CSW count cannot be zero.

Samples of CCW Traces

Figure 98 and Figure 99 show samples of traces started by a CPTRAP IO command issued on a VM/SP6 system. The data output, which is in hexadecimal format, is displayed in four columns. X'3C' entries represent the CCW and data. X'20' entries are the Time Of Day clock stamp associated with the CCW. For more information on CPTRAP, see the *CP System Commands Guide*.

Figure 98 is a sample of a VM CCW trace for I/O 560-561. The layout for this trace is:

Offset	Field Description
X'0038'	PCCA offset
X'02'	PCCA, network type (token-ring)
X'00'	PCCA, adapter number
X'6040'	Token-ring, AC and FD
X'FFFFFFFFFFFF'	Token-ring, destination address (broadcast)
X'90005A6BB806'	Token-ring, source address (ORed with 8000)
X'8220'	Token-ring, routing information
X'AAAA'	802.2 DSAP and SSAP (snap mode)
X'03'	802.2 control field
X'000000'	802.2 Prot/Org code
X'0806'	802.2 ether type (ARP type)
X'0006'	Beginning of ARP packet
X'0000'	Last offset, PCCA packet end delimiter.

```
3C TRAPID ENTRY        **MP** 3C080000 00900000 E3C3D740 40404040           CP
       TRAPID = TCP, TRAPSET = IOSET, IODATA = 500
       TRAPTYPE = IO, USER = TCPIP, I/O OLD PSW = 0F5C40
       DEVICE ADDRESS = 561, CSW = E05590C0 0C000000,
    -> CCW(1) = 01559028 2400003A, CCW ADDRESS = 5590B8, ** IDA **
       -> IDAW(1) = 14A020,
          DATA = 00380200 6040FFFF FFFFFFFF 90005A6B  *....- ........!,*
                 B8068220 AAAA0300 00000806 00060800  *..b.............*
                 06040001 10005A6B B8060943 3AE9C534  *......!,.....ZE.*
                 00D7C530 09433AEA 0000               *.PE.......      *
20 TOD STAMP           **MP** 20000000 00000000 A298CC1D B04DE000           CP
```

Figure 98. A Sample of an ARP Frame on a PCCA Token-Ring

Figure 99 shows a sample trace of an IP/ICMP packet on a PCCA token-ring. The layout for this trace is:

Offset	Field Description
X'0068'	PCCA offset
X'02'	PCCA, network type
X'00'	PCCA, adapter number
X'6040'	Token-ring, AC and FD
X'10005A250858'	Token-ring, destination address
X'000000000000'	Token-ring, source address
X'AAAA03000000'	802.2 frame
X'0800'	802.2 ether type (IP)
X'**45**'	Beginning of IP packet (version and IP header length)
X'00'	IP type of service
X'004D'	IP total length
X'002B'	IP datagram identification
X'0000'	IP flags and fragment offset
X'3C'	Time to live
X'**11**'	IP protocol (ICMP)
X'05A3'	Header checksum
X'09433AE9'	Source IP address
X'09432B64'	Destination IP address
X'0000'	Last offset, PCCA packet end delimiter.

```
3C TRAPID ENTRY        **MP** 3C080000 00C00000 E3C3D740 40404040           CP
       TRAPID = TCP, TRAPSET = IOSET, IODATA = 500
       TRAPTYPE = IO, USER = TCPIP, I/O OLD PSW = 0F5C40
       DEVICE ADDRESS = 561, CSW = E05590C0 0C000000,
    -> CCW(1) = 01559028 2400006A, CCW ADDRESS = 5590B8, ** IDA **
       -> IDAW(1) = 14A020,
          DATA = 00680200 60401000 5A250858 00000000  *....- ..!.......*
                 0000AAAA 03000000 08004500 004D002B  *.............(..*
                 00003C11 05A30943 3AE90943 2B640400  *.....t...Z......*
                 00350039 ED000001 01000001 00000000  *................*
                 00000652 414C564D 4D085443 50495044  *.....<.((...&.&.*
                 45560752 414C4549 47480349 424D0343  *.....<.......(..*
                 4F4D0000 010001C3 0000               *|(.....C..      *
20 TOD STAMP           **MP** 20000000 00000000 A298CC1E 01BE0000           CP
```

Figure 99. A Sample of an IP/ICMP Packet on a PCCA Token-Ring

Figure 100 on page 208 shows a sample of PCCA block encapsulating an IP/TCP packet on an Ethernet LAN. The trace was run on a VM/SP5 system. The data output, which is in hexadecimal format, is displayed in three columns. In SP4-5

CCW traces, ignore the first three words. The following is a description of the highlighted fields that mark the beginning of blocks or packets:

Field	Description
X'00F6'	Next message offset
X'45'	Starting of IP packet
X'0616'	Starting of TCP packet
X'0000'	Last offset, PCCA packet end delimiter.

```
I/O          CUU =0AE0 CSW = E0930DC0 0C004F08 PSW ADDR = 20D694
17:19:41/378927
     CCW = 0291D928 24005000          (930DB8)
                    C9C4C1E6 0091B920 00000000 *IDAW.J......*
                    00F60100 00DD0102 33C102CF *.6.......A..*
                    1F600887 08004500 00E437B3 *...G.....U..*
                    00004006 397C2C4A 01102C4A *.. .........*
                    01180616 00C80000 02212F4D *.....H......*
                    E9995018 111C1D4F 0000084C *ZR..........*
                    00000100 00003C00 00000250 *............*
                    0000BC00 00004442 53000000 *............*
                    69777331 34007361 30303130 *............*
                    00000000 00000000 54532053 *............*
                    43490000 0200FFFF FFFF0100 *............*
                    00007800 00002F75 73722F74 *............*
                    6573742F 30313233 34353637 *............*
                    000034AD 0A0020AD 0A002CFC *............*
                    F70014FC F70034FC F7007A9E *7...7...7...*
                    02004AFF F7000100 73613031 *....7.......*
                    20707264 000044AD 0A0038FC *............*
                    F70038FC F70040FC F700906D *7...7. .7...*
                    02002900 00000100 00000000 *............*
                    00000200 00000000 00000000 *............*
                    00000000 0000B601 00000000 *............*
                    00000000 F7000000 00000000 *....7.......*
```

Figure 100. A Sample of a VM/SP4-5 CCW Trace

Figure 101 shows the IP header format. For more information about IP headers, see RFC 791, which is represented with 32-bit words. This sample trace has the same IP header shown in Figure 100.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    4       5       0       0       0       0       E       4
 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    3       7       B       3       0       0       0       0
 0 0 1 1 0 1 1 1 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    4       0       0       6       3       9       7       C
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1 0 1 1 1 1 0 0
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                         44.74.1.16
    2       C       4       A       0       1       1       0
 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                         44.74.1.24
    2       C       4       A       0       1       1       8
 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 101. IP Header Format

Figure 102 shows the TCP header format.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   0       6       1       6       0       0       C       8
 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   0       0       0       0       0       2       2       1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   2       F       4       D       E       9       9       9
 0 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 1 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   5       0       1       8       1       1       C       1
 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   1       D       4       F       0       0       0       0
 0 0 0 1 1 1 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                          L
   0       8       4       C       0       0       0       0
 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 102. TCP Header Format

CETI Devices

CETI devices are Ethernet, token-ring, and X.25 internal adapters for the IBM/9370.

CETI traces provide the data that is transferred on CETI devices in the trace output. Table 23 lists the functions of the CCW command codes for CETI devices.

Table 23. CCW Command Codes for CETI Devices

Code	Functions
X'00'	TestIO.
X'03'	Nop.
X'04'	Sense adapter state.
X'08'	Transfer in channel.
X'C1'	Write data buffer.
X'C2'	Read data buffer.
X'C4'	Sense adapter state.
X'C5'	Write control block.
X'C6'	Read control block.
X'C7'	Data synchronization.
X'C9'	Write data parameters.
X'CB'	Set CETI mode on.
X'CD'	Interrupt write parameters.
X'E4'	Sense ID.

Matching CCW Traces and TCP/IP Traces

TCPIP and CCW traces can be matched in numerous ways by using the following:

- The CCW address, which is provided in PCCA and CETI traces
- The device address and first command (CCW code)
- The IP packets ID (IP traces)
- All fields identified by decimal integers in TCPIP internal traces can be converted to hexadecimal values and matched with the values in the CCW trace or text output if it is provided by the trace.

Appendix A. Return Codes

This appendix describes return codes sent by TCP/IP to the local client and return codes for User Datagram Protocol (UDP).

TCP/IP Return Codes

Table 24 describes the return codes sent by TCP/IP to servers and clients through the Virtual Machine Communication Facility (VMCF).

Table 24 (Page 1 of 2). TCP/IP Return Codes Sent to Servers and Clients

Return Message	Value	Description
OK	0	
ABNORMALcondition	–1	This indicates a VMCF error that is not fatal.
ALREADYclosing	–2	Connection is closing.
BADlengthARGUMENT	–3	Length parameter is invalid.
CANNOTsendDATA	–4	
CLIENTrestart	–5	
CONNECTIONalreadyEXISTS	–6	
DESTINATIONunreachable	–7	Returned from the remote site or gateway.
ERRORinPROFILE	–8	
FATALerror	–9	This is a fatal VMCF error.
HASnoPASSWORD	–10	Errors ...
INCORRECTpassword	–11	...in opening
INVALIDrequest	–12	
INVALIDuserID	–13	...file
INVALIDvirtualADDRESS	–14	...used
KILLEDbyCLIENT	–15	
LOCALportNOTavailable	–16	
MINIDISKinUSE	–17	...by
MINIDISKnotAVAILABLE	–18	...MonCommand
NObufferSPACE	–19	
NOmoreINCOMINGdata	–20	
NONlocalADDRESS	–21	
NOoutstandingNOTIFICATIONS	–22	
NOsuchCONNECTION	–23	
NOtcpIPservice	–24	
NOTyetBEGUN	–25	Client has not called BeginTcpIp.
NOTyetOPEN	–26	Client has not called TcpOpen.
OPENrejected	–27	

Return Message	Value	Description
PARAMlocalADDRESS	–28	Invalid...
PARAMstate	–29	...values...
PARAMtimeout	–30	...specified...
PARAMunspecADDRESS	–31	...in connection
PARAMunspecPORT	–32	...information record
PROFILEnotFOUND	–33	
RECEIVEstillPENDING	–34	
REMOTEclose	–35	Foreign client is closing.
REMOTEreset	–36	
SOFTWAREerror	–37	This is a WISCNET software error.
TCPipSHUTDOWN	–38	
TIMEOUTconnection	–39	
TIMEOUTopen	–40	
TOOmanyOPENS	–41	
UNAUTHORIZEDuser	–43	
UNEXPECTEDsyn	–44	
UNIMPLEMENTEDrequest	–45	
UNKNOWNhost	–46	There is a lack of information in the tables.
UNREACHABLEnetwork	–47	
UNSPECIFIEDconnection	–48	
VIRTUALmemoryTOOsmall	–49	
WRONGsecORprc	–50	The request does not have the necessary security or priority.
X25tooCongested	–51	No virtual circuits are available.
YOURend	–55	
ZEROresources	–56	

Table 24 (Page 2 of 2). TCP/IP Return Codes Sent to Servers and Clients

UDP Error Return Codes

Table 25 describes errors that are specific to UDP.

Table 25. UDP Error Return Codes

Return Message	Value	Description
UDPlocalADDRESS	–57	Invalid local address.
UDPunspecADDRESS	–59	Unspecified local address.
UDPunspecPORT	–60	Unspecified local port.
UDPzeroRESOURCES	–61	No space available to continue.
FSENDstillPENDING	–62	TcpFSend is still outstanding.

Appendix B. Related Protocol Specifications

IBM is committed to industry standards. The internet protocol suite is still evolving through Requests for Comments (RFC). New protocols are being designed and implemented by researchers, and are brought to the attention of the internet community in the form of RFCs. Some of these are so useful that they become a recommended protocol. That is, all future implementations for TCP/IP are recommended to implement this particular function or protocol. These become the *de facto* standards, on which the TCP/IP protocol suite is built.

Many features of TCP/IP for VM are based on the following RFCs:

RFC	Title	Author
768	*User Datagram Protocol*	J.B. Postel
791	*Internet Protocol*	J.B. Postel
792	*Internet Control Message Protocol*	J.B. Postel
793	*Transmission Control Protocol*	J.B. Postel
821	*Simple Mail Transfer Protocol*	J.B. Postel
822	*Standard for the Format of ARPA Internet Text Messages*	D. Crocker
823	*DARPA Internet Gateway*	R.M. Hinden, A. Sheltzer
826	*Ethernet Address Resolution Protocol: or Converting Network Protocol Addresses to 48.Bit Ethernet Address for Transmission on Ethernet Hardware*	D.C. Plummer
854	*Telnet Protocol Specification*	J.B. Postel, J.K. Reynolds
856	*Telnet Binary Transmission*	J.B. Postel, J.K. Reynolds
857	*Telnet Echo Option*	J.B. Postel, J.K. Reynolds
877	*Standard for the Transmission of IP Datagrams over Public Data Networks*	J.T. Korb
885	*Telnet End of Record Option*	J.B. Postel
903	*Reverse Address Resolution Protocol*	R. Finlayson, T. Mann, J.C. Mogul, M. Theimer
904	*Exterior Gateway Protocol Formal Specification*	D.L. Mills
919	*Broadcasting Internet Datagrams*	J.C. Mogul
922	*Broadcasting Internet Datagrams in the Presence of Subnets*	J.C. Mogul
950	*Internet Standard Subnetting Procedure*	J.C. Mogul, J.B. Postel
952	*DoD Internet Host Table Specification*	K. Harrenstien, M.K. Stahl, E.J. Feinler
959	*File Transfer Protocol*	J.B. Postel, J.K. Reynolds
974	*Mail Routing and the Domain Name System*	C. Partridge
1009	*Requirements for Internet Gateways*	R.T. Braden, J.B. Postel
1013	*X Window System Protocol, Version 11: Alpha Update*	R.W. Scheifler
1014	*XDR: External Data Representation Standard*	Sun Microsystems Incorporated

RFC	Title	Author
1027	*Using ARP to Implement Transparent Subnet Gateways*	S. Carl-Mitchell, J.S. Quarterman
1032	*Domain Administrators Guide*	M.K. Stahl
1033	*Domain Administrators Operations Guide*	M. Lottor
1034	*Domain Names—Concepts and Facilities*	P.V. Mockapetris
1035	*Domain Names—Implementation and Specification*	P.V. Mockapetris
1042	*Standard for the Transmission of IP Datagrams over IEEE 802 Networks*	J.B. Postel, J.K. Reynolds
1044	*Internet Protocol on Network System's HYPERchannel: Protocol Specification*	K. Hardwick, J. Lekashman
1055	*Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP*	J.L. Romkey
1057	*RPC: Remote Procedure Call Protocol Version 2 Specification*	Sun Microsystems Incorporated
1058	*Routing Information Protocol*	C.L. Hedrick
1091	*Telnet Terminal-Type Option*	J. VanBokkelen
1094	*NFS: Network File System Protocol Specification*	Sun Microsystems Incorporated
1118	*Hitchhikers Guide to the Internet*	E. Krol
1122	*Requirements for Internet Hosts-Communication Layers*	R.T. Braden
1123	*Requirements for Internet Hosts-Application and Support*	R.T. Braden
1155	*Structure and Identification of Management Information for TCP/IP-Based Internets*	M.T. Rose, K. McCloghrie
1156	*Management Information Base for Network Management of TCP/IP-based Internets*	K. McCloghrie, M.T. Rose
1157	*Simple Network Management Protocol (SNMP),*	J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin
1179	*Line Printer Daemon Protocol*	The Wollongong Group, L. McLaughlin III
1180	*TCP/IP Tutorial,*	T. J. Socolofsky, C.J. Kale
1183	*New DNS RR Definitions* (Updates RFC 1034, RFC 1035)	C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris,
1187	*Bulk Table Retrieval with the SNMP*	M.T. Rose, K. McCloghrie, J.R. Davin
1188	*Proposed Standard for the Transmission of IP Datagrams over FDDI Networks*	D. Katz
1198	*FYI on the X Window System*	R.W. Scheifler
1207	*FYI on Questions and Answers: Answers to Commonly Asked Experienced Internet User Questions*	G.S. Malkin, A.N. Marine, J.K. Reynolds
1208	*Glossary of Networking Terms*	O.J. Jacobsen, D.C. Lynch
1213	*Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II,*	K. McCloghrie, M.T. Rose
1215	*Convention for Defining Traps for Use with the SNMP*	M.T. Rose

RFC	Title	Author
1228	*SNMP-DPI Simple Network Management Protocol Distributed Program Interface*	G.C. Carpenter, B. Wijnen
1229	*Extensions to the Generic-Interface MIB*	K. McCloghrie
1230	*IEEE 802.4 Token Bus MIB IEEE 802 4 Token Bus MIB*	K. McCloghrie, R. Fox
1231	*IEEE 802.5 Token Ring MIB IEEE 802.5 Token Ring MIB*	K. McCloghrie, R. Fox, E. Decker
1267	*A Border Gateway Protocol 3 (BGP-3)*	K. Lougheed, Y. Rekhter
1268	*Application of the Border Gateway Protocol in the Internet*	Y. Rekhter, P. Gross
1269	*Definitions of Managed Objects for the Border Gateway Protocol (Version 3)*	S. Willis, J. Burruss
| 1293	*Inverse Address Resolution Protocol*	T. Bradley, C. Brown
1270	*SNMP Communications Services*	F. Kastenholz, ed.
1323	*TCP Extensions for High Performance*	V. Jacobson, R. Braden, D. Borman
1325	*FYI on Questions and Answers: Answers to Commonly Asked New Internet User Questions*	G.S. Malkin, A.N. Marine
1350	*TFTP Protocol*	K.R. Sollins
1351	*SNMP Administrative Model*	J. Davin, J. Galvin, K. McCloghrie
1352	*SNMP Security Protocols*	J. Galvin, K. McCloghrie, J. Davin
1353	*Definitions of Managed Objects for Administration of SNMP Parties*	K. McCloghrie, J. Davin, J. Galvin
1354	*IP Forwarding Table MIB*	F. Baker
1356	*Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode*	A. Malis, D. Robinson, R. Ullmann
1374	*IP and ARP on HIPPI*	J. Renwick, A. Nicholson
1381	*SNMP MIB Extension for X.25 LAPB*	D. Throop, F. Baker
1382	*SNMP MIB Extension for the X.25 Packet Layer*	D. Throop
1387	*RIP Version 2 Protocol Analysis*	G. Malkin
1389	*RIP Version 2 MIB Extension*	G. Malkin
1390	*Transmission of IP and ARP over FDDI Networks*	D. Katz
1393	*Traceroute Using an IP Option*	G. Malkin
1397	*Default Route Advertisement In BGP2 And BGP3 Versions of the Border Gateway Protocol*	D. Haskin
1398	*Definitions of Managed Objects for the Ethernet-like Interface Types*	F. Kastenholz
1440	*SIFT/UFT:Sender-Initiated/Unsolicited File Transfer*	R. Troth
| 1483	*Multiprotocol Encapsulation over ATM Adaptation Layer 5*	J. Heinanen
1540	*IAB Official Protocol Standards*	J.B. Postel
1647	*TN3270 Enhancements*	B. Kelly
1700	*Assigned Numbers*	J.K. Reynolds, J.B. Postel
| 1723	*RIP Version 2 — Carrying Additional Information*	G. Malkin

RFC	Title	Author
1813	*NFS Version 3 Protocol Specification*	B. Callaghan, B. Pawlowski, P. Stauback, Sun Microsystems Incorporated
2225	*Classical IP and ARP over ATM*	M. Laubach, J. Halpern

These documents can be obtained from:

Government Systems, Inc.
Attn: Network Information Center
14200 Park Meadow Drive
Suite 200
Chantilly, VA 22021

Many RFCs are available online. Hard copies of all RFCs are available from the NIC, either individually or on a subscription basis. Online copies are available using FTP from the NIC at `nic.ddn.mil`. Use FTP to download the files, using the following format:

```
RFC:RFC-INDEX.TXT
RFC:RFCnnnn.TXT
RFC:RFCnnnn.PS
```

Where:

nnnn	Is the RFC number.
`TXT`	Is the text format.
`PS`	Is the PostScript format.

You can also request RFCs through electronic mail, from the automated NIC mail server, by sending a message to `service@nic.ddn.mil` with a subject line of `RFC` *nnnn* for text versions or a subject line of `RFC` *nnnn*`.PS` for PostScript versions. To request a copy of the RFC index, send a message with a subject line of `RFC INDEX`.

For more information, contact `nic@nic.ddn.mil`. Information is also available through http://www.internic.net.

Glossary

This glossary describes the most common terms associated with TCP/IP communication in an internet environment, as used in this book.

If you do not find the term you are looking for, see the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

For abbreviations, the definition usually consists only of the words represented by the letters; for complete definitions, see the entries for the words.

Numerics

3172. IBM Interconnect Controller.

3174. IBM Establishment Controller.

3270. Refers to a series of IBM display devices; for example, the IBM 3275, 3276 Controller Display Station, 3277, 3278, and 3279 Display Stations, the 3290 Information Panel, and the 3287 and 3286 printers. A specific device type is used only when a distinction is required between device types. Information about display terminal usage also refers to the IBM 3138, 3148, and 3158 Display Consoles when used in display mode, unless otherwise noted.

37xx Communication Controller. A network interface used to connect a TCP/IP for VM or MVS network that supports X.25 connections. NCP with X.25 NPSI must be running in the controller, and VTAM must be running on the host.

6611. IBM Network Processor.

8232. IBM LAN Station.

9370. Refers to a series of processors, namely the IBM 9373 Model 20, the IBM 9375 Models 40 and 60, and the IBM 9377 Model 90 and other models.

A

abend. The abnormal termination of a program or task.

abstract syntax. A description of a data structure that is independent of machine-oriented structures and encodings.

Abstract Syntax Notation One (ASN.1). The OSI language for describing abstract syntax.

active gateway. A gateway that is treated like a network interface in that it is expected to exchange routing information, and if it does not do so for a period of time, the route associated with the gateway is deleted.

active open. The state of a connection that is actively seeking a service. Contrast with *passive open*.

adapter. (1) A piece of hardware that connects a computer and an external device. (2) An auxiliary device or unit used to extend the operation of another system.

address. The unique code assigned to each device or workstation connected to a network. A standard internet address uses a two-part, 32-bit address field. The first part of the address field contains the network address; the second part contains the local address.

address mask. A bit mask used to select bits from an Internet address for subnet addressing. The mask is 32 bits long and selects the network portion of the Internet address and one or more bits of the local portion. It is sometimes called a subnet mask.

address resolution. A means for mapping network layer addresses onto media-specific addresses. See *ARP*.

Address Resolution Protocol (ARP). A protocol used to dynamically bind an internet address to a hardware address. ARP is implemented on a single physical network and is limited to networks that support broadcast addressing.

address space. A collection of bytes that are allocated, and in many ways managed, as a single entity by CP. Each byte within an address space is identified by a unique address. An address space represents an extent of storage available to a program. Address spaces allocated by VM range in size from 64KB to 2GB.

Advanced Interactive Executive (AIX). IBM's licensed version of the UNIX operating system.

Advanced Program-to-Program Communications (APPC). The interprogram communication service within SNA LU 6.2 on which the APPC/VM interface is based.

Advanced Research Projects Agency (ARPA). Now called DARPA, its the U.S. Government agency that funded the ARPANET.

Advanced Research Projects Agency Network (ARPANET). A packet switched network developed in the early 1970's that is the forerunner of today's Internet. It was decommissioned in June 1990.

agent. As defined in the SNMP architecture, an agent, or an SNMP server is responsible for performing the network management functions requested by the network management stations.

AIX. Advanced Interactive Executive.

American National Standard Code for Information Interchange (ASCII). (1) The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (2) The default file transfer type for FTP, used to transfer files that contain ASCII text characters.

American National Standards Institute (ANSI). An organization consisting of producers, consumers, and general interest groups that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. ANSI is sponsored by the Computer and Business Equipment Manufacturer Association and is responsible for establishing voluntary industry standards.

ANSI. American National Standards Institute.

API. Application Program Interface.

APPC. Advanced Program-to-Program Communications.

application. The use to which an information processing system is put, for example, a payroll application, an airline reservation application, a network application.

application layer. The seventh layer of the OSI (Open Systems Interconnection) model for data communication. It defines protocols for user or application programs.

Application Program Interface (API). The formally defined programming-language interface between an IBM system control program or licensed program and its user. APIs allow programmers to write application programs that use the TCP, UDP, and IP layers of the TCP/IP protocol suite.

argument. A parameter passed between a calling program and a called program.

ARP. Address Resolution Protocol.

ARPA. Advanced Research Projects Agency.

ARPANET. Advanced Research Projects Agency Network.

ASCII. (1) American National Standard Code for Information Interchange. (2) The default file transfer type for FTP, used to transfer files that contain ASCII text characters.

ASN.1. Abstract Syntax Notation One.

ASYNC. Asynchronous.

asynchronous (ASYNC). Without regular time relationship; unexpected or unpredictable with respect to the execution of program instruction. See *synchronous*.

asynchronous communication. A method of communication supported by the operating system that allows an exchange of data with remote device, using either a start-stop line or an X.25 line. Asynchronous communications include the file transfer and the interactive terminal facility support.

Athena Widgets. The X Window widget set developed by MIT for Project Athena.

Attachment Unit Interface (AUI). Connector used with thick Ethernet that often includes a drop cable.

AUI. Attachment Unit Interface.

attention key. A function key on terminals that, when pressed, causes an I/O interruption in the processing unit.

authentication server. The service that reads a Kerberos database to verify that a client making a request for access to an end-service is the client named in the request. The authentication server provides an authenticated client ticket as permission to access the ticket-granting server.

authenticator. Information encrypted by a Kerberos authentication server that a client presents along with a ticket to an end-server as permission to access the service.

authorization. The right granted to a user to communicate with, or to make use of, a computer system or service.

B

backbone. (1) In a local area network multiple-bridge ring configuration, a high-speed link to which rings are connected by means of bridges. A backbone can be configured as a bus or as a ring. (2) In a wide area network, a high-speed link to which nodes or data switching exchanges (DSES) are connected.

background task. A task with which the user is not currently interacting, but continues to run.

baseband. Characteristic of any network technology that uses a single carrier frequency and requires all stations attached to the network to participate in every transmission. See *broadband*.

Basic Encoding Rules (BER). Standard rules for encoding data units described in ASN.1. Sometimes incorrectly grouped under the term ASN.1, which correctly refers only to the abstract description language, not the encoding technique.

Basic Input/Output System (BIOS). A set of routines that permanently resides in read-only memory (ROM) in a PC. The BIOS performs the most basic tasks, such as sending a character to the printer, booting the computer, and reading the keyboard.

batch. (1) An accumulation of data to be processed. (2) A group of records or data processing jobs brought together for processing or transmission. (3) Pertaining to activity involving little or no user action. See *interactive*

Bayonet Neill-Concelman (BNC). A standardized connector used with Thinnet and coaxial cable.

Because It's Time NETwork (BITNET). A network of hosts that use the Network Job Entry (NJE) protocol to communicate. The network is primarily composed of universities, nonprofit organizations, and research centers. BITNET has recently merged with the Computer and Science Network (CSNET) to form the Corporation for Research and Educational Networking (CSNET). See *CSNET*.

BER. Basic Encoding Rules.

Berkeley Software Distribution (BSD). Term used when describing different versions of the Berkeley UNIX software, as in "4.3BSD UNIX".

BFS. Byte File System.

big-endian. A format for storage or transmission of binary data in which the most significant bit (or byte) comes first. The reverse convention is little-endian.

BIOS. Basic Input/Output System.

BITNET. Because It's Time NETwork.

block. A string of data elements recorded, processed, or transmitted as a unit. The elements can be characters, words, or physical records.

blocking mode. If the execution of the program cannot continue until some event occurs, the operating system suspends the program until that event occurs.

BNC. Bayonet Neill-Concelman.

BOOTPD. Bootstrap Protocol Daemon.

Bootstrap Protocol Daemon (BOOTPD). The BOOTP daemon responds to client requests for boot information using information contained in a BOOTP machine file.

bridge. (1) A router that connects two or more networks and forwards packets among them. The operations carried out by a bridge are done at the physical layer and are transparent to TCP/IP and TCP/IP routing. (2) A functional unit that connects two local area networks (LANs) that use the same logical link control (LLC) procedures but may use different medium access control (MAC) procedures.

broadband. Characteristic of any network that multiplexes multiple, independent network carriers onto a single cable. This is usually done using frequency division multiplexing. Broadband technology allows several networks to coexist on one single cable; traffic from one network does not interfere with traffic from another, because the "conversations" happen on different frequencies in the ether, similar to a commercial radio system.

broadcast. The simultaneous transmission of data packets to all nodes on a network or subnetwork.

broadcast address. An address that is common to all nodes on a network.

BSD. Berkeley Software Distribution.

bus topology. A network configuration in which only one path is maintained between stations. Any data transmitted by a station is concurrently available to all other stations on the link.

byte-ordering. The method of sorting bytes under specific machine architectures. Of the two common methods, little endian byte ordering places the least significant byte first. This method is used in Intel** microprocessors. In the second method, big endian byte ordering, the most significant byte is placed first. This method is used in Motorola microprocessors.

Byte File System (BFS). A file system in which a file consists of an ordered sequence of bytes rather than records. BFS files can be organized into hierarchical directories. Byte file systems are enrolled as file spaces in CMS file pools.

C

Carrier Sense Multiple Access with Collision Detection (CSMA/CD). The access method used by local area networking technologies such as Ethernet.

case-sensitive. A condition in which entries for an entry field must conform to a specific lowercase, uppercase, or mixed-case format to be valid.

CCITT. Comite Consultatif International Telegraphique et Telephonique.

channel. A path in a system that connects a processor and main storage with an I/O device.

channel-attached. (1) pertaining to attachment of devices directly by data channels (I/O channels)to a computer. (2) Pertaining to devices attached to a controlling unit by cables, rather than by telecommunication lines. (3) Synonymous with local, locally attached.

checksum. The sum of a group of data associated with the group and used for checking purposes.

CICS. Customer Information Control System.

Class A network. An internet network in which the high-order bit of the address is 0. The host number occupies the three, low-order octets.

Class B network. An internet network in which the high-order bit of the address is 1, and the next high-order bit is 0. The host number occupies the two low-order octets.

Class C network. An internet network in which the two high-order bits of the address are 1 and the next high-order bit is 0. The host number occupies the low-order octet.

CLAW. Common Link Access to Workstation.

client. (1) A function that requests services from a server, and makes them available to the user. (2) In MVS, an address space that is using TCP/IP services.

client-server model. A common way to describe network services and the model user processes (programs) of those services. Examples include the name server and resolver paradigm of the DNS and file server/file client relationships such as NFS and diskless hosts.

client-server relationship. Any device that provides resources or services to other devices on a network is a *server*. Any device that employs the resources provided by a server is a *client*. A machine can run client and server processes at the same time.

CLIST. Command List.

CLPA. Create Link Pack Area.

CMS. Conversational Monitor System.

Comite Consultatif International Telegraphicque et Telephonique (CCITT). The International Telegraph and Telephone Consultative Committee. A unit of the International Telecommunications Union (ITU) of the United Nations. CCITT produces technical standards, known as “recommendations,” for all internationally controlled aspects of analog and digital communication.

command. The name and any parameters associated with an action that can be performed by a program. The command is entered by the user; the computer performs the action requested by the command name.

Command List (CLIST). A list of commands and statements designed to perform a specific function for the user.

command prompt. A displayed symbol, such as [C:\] that requests input from a user.

Common Link Access to Workstation (CLAW). A continuously executing duplex channel program designed to minimize host interrupts while maximizing channel utilization.

communications adapter. A hardware feature that enables a computer or device to become a part of a data network.

community name. A password used by hosts running Simple Network Management Protocol (SNMP) agents to access remote network management stations.

compile. (1) To translate a program written in a high-level language into a machine language program. (2) The computer actions required to transform a source file into an executable object file.

compiler. A program that translates a source program into an executable program (an object program).

Computer and Science Network (CSNET). A large computer network, mostly in the U.S. but with international connections. CSNET sites include universities, research labs, and some commercial companies. It is now merged with BITNET to form CREN. See *BITNET*.

connection. (1) An association established between functional units for conveying information. (2) The path between two protocol modules that provides reliable stream delivery service. In an internet, a connection extends from a TCP module on one machine to a TCP module on the other.

Control Program (CP). The VM operating system that manages the real processor's resources and is responsible for simulating System/370s or 390s for individual users.

conversational monitor system (CMS). A virtual machine operating system that provides general interactive time sharing, problem solving, and program development capabilities, and operates only under control of the VM//ESA control program.

Corporation for Research and Educational Networking (CREN). A large computer network formed from the merging of BITNET and CSNET. See *BITNET* and *CSNET*.

CP. Control Program.

Create Link Pack Area (CLPA). A parameter specified at startup, which says to create the link pack area.

CREN. Corporation for Research and Educational Networking.

CSMA/CD. Carrier Sense Multiple Access with Collision Detection.

CSNET. Computer and Science Network.

Customer Information Control System (CICS). An IBM-licensed program that enables transactions entered at remote terminals to be processed concurrently by user written application programs. It includes facilities for building, using, and maintaining databases.

D

daemon. A background process usually started at system initialization that runs continuously and performs a function required by other processes. Some daemons are triggered automatically to perform their task; others operate periodically.

DASD. Direct Access Storage Device.

DARPA. Defense Advanced Research Projects Agency.

DATABASE 2 (DB2). An IBM relational database management system for the MVS operating system.

database administrator (DBA). An individual or group responsible for the rules by which data is accessed and stored. The DBA is usually responsible for database integrity, security, performance and recovery.

datagram. A basic unit of information that is passed across the internet, it consists of one or more data packets.

data link layer. Layer 2 of the OSI (Open Systems Interconnection) model; it defines protocols governing data packetizing and transmission into and out of each node.

data set. The major unit of data storage and retrieval in MVS, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. Synonymous with *file* in VM and OS/2.

DB2. DATABASE 2.

DBA. Database administrator.

DBCS. Double Byte Character Set.

DDN. Defense Data Network.

decryption. The unscrambling of data using an algorithm that works under the control of a key. The key allows data to be protected even when the algorithm is unknown. Data is unscrambled after transmission.

default. A value, attribute, or option that is assumed when none is explicitly specified.

Defense Advanced Research Projects Agency (DARPA). The U.S. government agency that funded the ARPANET.

Defense Data Network (DDN). Comprises the MILNET and several other Department of Defense networks.

destination node. The node to which a request or data is sent.

DHCPD. Dynamic Host Configuration Protocol Daemon.

Direct Access Storage Device (DASD). A device in which access to data is independent of where data resides on the device.

directory. A named grouping of files in a file system.

Disk Operating System (DOS). An operating system for computer systems that use disks and diskettes for auxiliary storage of programs and data.

display terminal. An input/output unit by which a user communicates with a data-processing system or sub-system. Usually includes a keyboard and always provides a visual presentation of data; for example, an IBM 3179 display.

Distributed Program Interface (DPI). A programming interface that provides an extension to the function provided by the SNMP agents.

DLL. Dynamic Link Library.

DNS. Domain Name System.

domain. In an internet, a part of the naming hierarchy. Syntactically, a domain name consists of a sequence of names (labels) separated by periods (dots).

Domain Name System (DNS). A system in which a resolver queries name servers for resource records about a host.

domain naming. A hierarchical system for naming network resources.

DOS. Disk Operating System.

dotted-decimal notation. The syntactic representation for a 32-bit integer that consists of four 8-bit numbers, written in base 10 and separated by periods (dots). Many internet application programs accept dotted decimal notations in place of destination machine names.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS.

doubleword. A contiguous sequence of bits or characters that comprises two computer words and is capable of being addressed as a unit.

DPI. Distributed Program Interface.

Dynamic Host Configuration Protocol Daemon (DHCPD). The DHCP daemon (DHCPD server) responds to client requests for boot information using information contained in a DHCP machine file. This information includes the IP address of the client, the IP address of the TFTP daemon, and information about the files to request from the TFTP daemon.

dynamic resource allocation. An allocation technique in which the resources assigned for execution of computer programs are determined by criteria applied at the moment of need.

dynamic link library (DLL). A module containing dynamic link routines that is linked at load or run time.

E

EBCDIC. Extended binary-coded decimal interchange code.

EGP. Exterior Gateway Protocol.

encapsulation. A process used by layered protocols in which a lower-level protocol accepts a message from a higher-level protocol and places it in the data portion of the low-level frame. As an example, in Internet terminology, a packet would contain a header from the physical layer, followed by a header from the network layer (IP), followed by a header from the transport layer (TCP), followed by the application protocol data.

encryption. The scrambling or encoding of data using an algorithm that works under the control of a key. The key allows data to be protected even when the algorithm is unknown. Data is scrambled prior to transmission.

ES/9370 Integrated Adapters. An adapter you can use in TCP/IP for VM to connect into Token-Ring networks and Ethernet networks, as well as TCP/IP networks that support X.25 connections.

Ethernet. The name given to a local area packet-switched network technology invented in the early 1970s by Xerox**, Incorporated. Ethernet uses a Carrier Sense Multiple Access/Collision Detection (CSMA/CD) mechanism to send packets.

EXEC. In a VM operating system, a user-written command file that contains CMS commands, other user-written commands, and execution control statements, such as branches.

extended binary-coded decimal interchange code (EBCDIC). A coded character set consisting of 8-bit coded characters.

extended character. A character other than a 7-bit ASCII character. An extended character can be a 1-bit code point with the 8th bit set (ordinal 128-255) or a 2-bit code point (ordinal 256 and greater).

Exterior Gateway Protocol (EGP). A reachability routing protocol used by gateways in a two-level internet.

eXternal Data Representation (XDR). A standard developed by Sun Microsystems, Incorporated for representing data in machine-independent format.

F

FAT. File Allocation Table.

FDDI. Fiber Distributed Data Interface. Also used to abbreviate Fiber Optic Distributed Data Interface.

Fiber Distributed Data Interface (FDDI). The ANSI standard for high-speed transmission over fiber optic cable.

Fiber Optic Network. A network based on the technology and standards that define data transmission using cables of glass or plastic fibers carrying visible light. Fiber optic network advantages are: higher transmission speeds, greater carrying capacity, and lighter, more compact cable.

file. In VM and OS/2, a named set of records stored or processed as a unit. Synonymous with *data set* in MVS.

File Allocation Table (FAT). A table used to allocate space on a disk for a file.

File Transfer Access and Management (FTAM). An application service element that enables user application processes to manage and access a file system, which may be distributed.

File Transfer Protocol (FTP). A TCP/IP protocol used for transferring files to and from foreign hosts. FTP also provides the capability to access directories. Password protection is provided as part of the protocol.

foreign host. Any machine on a network that can be interconnected.

foreign network. In an internet, any other network interconnected to the local network by one or more intermediate gateways or routers.

foreign node. See *foreign host*.

frame. The portion of a tape on a line perpendicular to the reference edge, on which binary characters can be written or read simultaneously.

FTAM. File Transfer Access and Management.

FTP. File Transfer Protocol.

fullword. A computer word. In System/370, 32 bits or 4 bytes.

G

gadget. A windowless graphical object that looks like its equivalent like-named widget but does not support the translations, actions, or pop-up widget children supplied by that widget.

gateway. (1) A functional unit that interconnects a local data network with another network having different protocols. (2) A host that connects a TCP/IP network to a non-TCP/IP network at the application layer. See also *router*.

gather and scatter data. Two related operations. During the gather operation, data is taken from multiple buffers and transmitted. In the scatter operation, data is received and stored in multiple buffers.

GC. Graphics Context.

GContext. See *Graphics Context*.

GCS. Group Control System.

GDDM. Graphical Data Display Manager.

GDDMXD. Graphical Data Display Manager interface for X Window System. A graphical interface that formats and displays alphanumeric, data, graphics, and images on workstation display devices that support the X Window System.

GDF. Graphics data file.

Graphical Display Data Manager (GDDM). A group of routines that allows pictures to be defined and displayed procedurally through function routines that correspond to graphic primitives.

Graphics Context (GC). The storage area for graphics output. Also known as *GC* and *GContext*. Used only with graphics that have the same root and depth as the graphics content.

Group Control System (GCS). A component of VM/ESA, consisting of a shared segment that you can Initial Program Load (IPL) and run in a virtual machine. It provides simulated MVS services and unique supervisor services to help support a native SNA network.

H

handle. A temporary data representation that identifies a file.

halfword. A contiguous sequence of bits or characters that constitutes half a fullword and can be addressed as a unit.

HASP. Houston automatic spooling priority system.

HDLC. High-level Data Link Control.

header file. A file that contains constant declarations, type declarations, and variable declarations and assignments. Header files are supplied with all programming interfaces.

High-level Data Link Control (HDLC). An ISO protocol for X.25 international communication.

High Performance File System (HPFS). An OS/2 file management system that supports high-speed buffer storage, long file names, and extended attributes.

hop count. The number of gateways or routers through which a packet passes on its way to its destination.

host. A computer connected to a network, which provides an access method to that network. A host provides end-user services and can be a client, a server, or a client and server simultaneously.

Houston automatic spooling priority system (HASP). A computer program that provides supplementary job management, data management, and task management functions such as control of job flow, ordering of tasks, and spooling.

HPFS. High Performance File System.

HYPERchannel Adapter. A network interface used to connect a TCP/IP for VM or MVS host into an existing TCP/IP HYPERchannel network, or to connect TCP/IP hosts together to create a TCP/IP HYPERchannel network.

I

IAB. Internet Activities Board.

ICMP. Internet Control Message Protocol.

IEEE. Institute of Electrical and Electronic Engineers.

IETF. Internet Engineering Task Force.

IGP. Interior Gateway Protocol.

include file. A file that contains preprocessor text, which is called by a program, using a standard programming call. Synonymous with *header file*.

IMS. Information Management System.

Information Management System (IMS). A database/data communication (DB/DC) system that can manage complex databases and networks.

initial program load (IPL). The initialization procedure that causes an operating system to commence operation.

instance. Indicates a label that is used to distinguish among the variations of the *principal name*. An instance allows for the possibility that the same client or service can exist in several forms that require distinct authentication.

Institute of Electrical and Electronic Engineers (IEEE). An electronics industry organization.

Integrated Services Digital Network (ISDN). A digital, end-to-end telecommunication network that supports multiple services including, but not limited to, voice and data.

interactive. Pertaining to a program or a system that alternately accepts input and then responds. An interactive system is conversational; that is, a continuous dialog exists between user and system. See *batch*.

Interior Gateway Protocol (IGP). The protocol used to exchange routing information between collaborating routers in the Internet. RIP is an example of an IGP.

Internet. The largest internet in the world consisting of large national backbone nets (such as MILNET, NSFNET, and CREN) and a myriad of regional and local campus networks all over the world. The Internet uses the Internet protocol suite. To be on the Internet, you must have IP connectivity (be able to TELNET to, or PING, other systems). Networks with only electronic mail connectivity are not actually classified as being on the Internet.

Internet Activities Board (IAB). The technical body that oversees the development of the Internet suite of protocols (commonly referred to as TCP/IP). It has two task forces (the IRTF and the IETF) each charged with investigating a particular area.

Internet address. A 32-bit address assigned to hosts using TCP/IP. An internet address consists of a network number and a local address. Internet addresses are represented in a dotted-decimal notation and are used to route packets through the network.

Internet Engineering Task Force (IETF). One of the task forces of the IAB. The IETF is responsible for solving short-term engineering needs of the Internet.

International Organization for Standardization (ISO). An organization of national standards bodies from various countries established to promote development of standards to facilitate international exchange of goods and services, and develop cooperation in intellectual, scientific, technological, and economic activity.

internet or internetwork. A collection of packet switching networks interconnected by gateways, routers, bridges, and hosts to function as a single, coordinated, virtual network.

internet address. The unique 32-bit address identifying each node in an internet. See also *address*.

Internet Control Message Protocol (ICMP). The part of the Internet Protocol layer that handles error messages and control messages.

Internet Protocol (IP). The TCP/IP layer between the higher level host-to-host protocol and the local network protocols. IP uses local area network protocols to carry packets, in the form of datagrams, to the next gateway, router, or destination host.

interoperability. The capability of different hardware and software by different vendors to effectively communicate together.

Inter-user communication vehicle (IUCV). A VM facility for passing data between virtual machines and VM components.

intrinsics X-Toolkit. A set management mechanism that provides for constructing and interfacing between composite X Window widgets, their children, and other clients. Also, intrinsics provide the ability to organize a collection of widgets into an application.

IP. Internet Protocol.

IP datagram. The fundamental unit of information passed across the Internet. An IP datagram contains source and destination addresses along with data and a number of fields that define such things as the length of the datagram, the header checksum, and flags to say whether the datagram can be (or has been) fragmented.

IPL. Initial Program Load.

ISDN. Integrated Services Digital Network.

ISO. International Organization for Standardization.

IUCV. Inter-User Communication Vehicle.

J

JCL. Job Control Language.

JES. Job Entry Subsystem.

JIS. Japanese Institute of Standards.

Job Control Language (JCL). A problem-oriented language designed to express statements in a job that are used to identify the job or describe its requirements to an operating system.

Job Entry Subsystem (JES). An IBM System/370 licensed program that receives jobs into the system and processes all output data produced by the jobs.

JUNET. The Japanese Academic and Research Network that connects various UNIX operating systems.

K

Kanji. A graphic character set consisting of symbols used in Japanese ideographic alphabets. Each character is represented by 2 bytes.

katakana. A character set of symbols used on one of the two common Japanese phonetic alphabets, which is used primarily to write foreign words phonetically. See also *kanji*.

Kerberos. A system that provides authentication service to users in a network environment.

Kerberos Authentication System. An authentication mechanism used to check authorization at the user level.

L

LaMail. The client that communicates with the OS/2 Presentation Manager to manage mail on the network.

LAN. Local area network.

Line Printer Client (LPR). A client command that allows the local host to submit a file to be printed on a remote print server.

Line Printer Daemon (LPD). The remote printer server that allows other hosts to print on a printer local to your host.

little-endian. A format for storage or transmission of binary data in which the least significant bit (or byte) comes first. The reverse convention is big-endian.

LLB. Local Location Broker.

local area network (LAN). A data network located on the user's premises in which serial transmission is used for direct data communication among data stations.

local host. In an internet, the computer to which a user's terminal is directly connected without using the internet.

Local Location Broker (LLB). In Network Computing System (NCS) Location Broker, a server that maintains information about objects on the local host and provides the Location Broker forwarding facility.

local network. The portion of a network that is physically connected to the host without intermediate gateways or routers.

logical character delete symbol. A special editing symbol, usually the at (@) sign, which causes CP to delete it and the immediately preceding character from the input line. If many delete symbols are consecutively entered, the same number of preceding characters are deleted from the input line.

Logical Unit (LU). An entity addressable within an SNA-defined network. LUs are categorized by the types of communication they support.

LPD. Line Printer Daemon.

LPR. Line Printer Client.

LU. Logical Unit.

LU-LU session. In SNA, a session between two logical units (LUs). It provides communication between two end users, or between an end user and an LU services component.

LU type. In SNA, the classification of an LU-LU session in terms of the specific subset of SNA protocols and options supported by the logical units (LUs) for that session.

M

MAC. Media Access Control.

mail gateway. A machine that connects two or more electronic mail systems (often different mail systems on different networks) and transfers messages between them.

Management Information Base (MIB). A standard used to define SNMP objects, such as packet counts and routing tables, that are in a TCP/IP environment.

mapping. The process of relating internet addresses to physical addresses in the network.

mask. (1) A pattern of characters used to control retention or elimination of portions of another pattern of characters. (2) To use a pattern of characters to control retention or elimination of another pattern of characters. (3) A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters.

Maximum Transmission Unit (MTU). The largest possible unit of data that can be sent on a given physical medium.

media access control (MAC). The method used by network adapters to determine which adapter has access to the physical network at a given time.

Message Handling System (MHS). The system of message user agents, message transfer agents, message stores, and access units that together provide OSI electronic mail.

MHS. Message Handling System.

MIB. Management Information Base.

microcode. A code, representing the instructions of an instruction set, which is implemented in a part of storage that is not program-addressable.

MILNET. Military Network.

Military Network (MILNET). Originally part of the ARPANET, MILNET was partitioned in 1984 to make it possible for military installations to have reliable network service, while the ARPANET continued to be used for research. See *DDN*.

minidisk. Logical divisions of a physical direct access storage device.

modem (modulator/demodulator). A device that converts digital data from a computer to an analog signal that can be transmitted on a telecommunication line, and converts the analog signal received to data for the computer.

Motif. see OSF/Motif.

mouse. An input device that is used to move a pointer on the screen and select items.

MTU. Maximum Transmission Unit.

multicast. The simultaneous transmission of data packets to a group of selected nodes on a network or subnetwork.

multiconnection server. A server that is capable of accepting simultaneous, multiple connections.

Multiple Virtual Storage (MVS). Implies MVS/370, the MVS/XA product, and the MVS/ESA product.

multitasking. A mode of operation that provides for the concurrent performance execution of two or more tasks.

MVS. Multiple Virtual Storage.

N

name server. The server that stores resource records about hosts.

National Science Foundation (NSF). Sponsor of the NSFNET.

National Science Foundation Network (NSFNET). A collection of local, regional, and mid-level networks in the U.S. tied together by a high-speed backbone. NSFNET provides scientists access to a number of supercomputers across the country.

NCK**. Network Computing Kernel.

NCP. Network Control Program.

NCS. Network Computing System.

NDB. Network Database.

NDIS. Network Driver Interface Specification.

Netman. This device keyword specifies that this device is a 3172 LAN Channel Station that supports IBM Enterprise-Specific SNMP Management Information Base (MIB) variables for 3172. TCP/IP for VM supports SNMP GET and SNMP GETNEXT operations to request and retrieve 3172 Enterprise-Specific MIB variables. These requests are answered only by those 3172 devices with the NETMAN option in the PROFILE TCPIP file.

NetView. A system 390-based, IBM-licensed program used to monitor, manage, and diagnose the problems of a network.

network. An arrangement of nodes and connecting branches. Connections are made between data stations. Physical network refers to the hardware that makes up a network. Logical network refers to the abstract organization overlaid on one or more physical networks. An internet is an example of a logical network.

network adapter. A physical device, and its associated software, that enables a processor or controller to be connected to a network.

network administrator. The person responsible for the installation, management, control, and configuration of a network.

Network Computing Kernel (NCK). In the Network Computing System (NCS), the combination of the remote procedure call runtime library and the Location Broker.

Network Computing System (NCS). A set of software components developed by Apollo, Incorporated, that conform to the Network Computing Architecture (NCA). NCS is made up of two parts: the nidl compiler and Network Computing Kernel (NCK). NCS is a programming tool kit that allows programmers to distribute processing power to other hosts.

Network Control Program (NCP). An IBM-licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability.

network database (NDB). An IBM-licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. NDB allows interoperability among different database systems, and uses RPC protocol with a client/server type of relationship. NDB is used for data conversion, security, I/O buffer management, and transaction management.

Network Driver Interface Specification (NDIS). An industry-standard specification used by applications as an interface with network adapter device drivers.

network elements. As defined in the SNMP architecture, network elements are gateways, routers, and hosts that contain management agents responsible for performing the network management functions requested by the network management stations.

network file system (NFS). The NFS protocol, which was developed by Sun Microsystems, Incorporated, allows computers in a network to access each other's file systems. Once accessed, the file system appears to reside on the local host.

Network Information Center (NIC). Originally there was only one, located at SRI International and tasked to serve the ARPANET (and later DDN) community. Today, there are many NICs operated by local, regional, and national networks all over the world. Such centers provide user assistance, document service, training, and more.

Network Interface Definition Language (NIDL). A declarative language for the definition of interfaces that has two forms, a Pascal-like syntax and a C-like syntax. NIDL is a component of the Network Computing Architecture.

Network Job Entry (NJE). In object distribution, an entry in the network job table that specifies the system action required for incoming network jobs sent by a particular user or group of users. Each entry is identified by the user ID of the originating user or group.

network layer. Layer 3 of the Open Systems Interconnection (OSI) model; it defines protocols governing data routing.

network management stations. As defined in the SNMP architecture, network management stations, or SNMP clients, execute management applications that monitor and control network elements.

NFS. Network file system.

NIC. Network Information Center.

NIDL. Network Interface Definition Language.

NJE. Network Job Entry.

node. (1) In a network, a point at which one or more functional units connect channels or data circuits. (2) In a network topology, the point at an end of a branch.

nonblocking mode. If the execution of the program cannot continue until some event occurs, the operating system does not suspend the program until that event occurs. Instead, the operating system returns an error message to the program.

NPSI. X.25 NCP Packet Switching Interface.

NSF. National Science Foundation.

NSFNET. National Science Foundation Network.

O

octet. A byte composed of eight binary elements.

OfficeVision (OV). IBM's new proprietary, integrated office management system used for sending, receiving, and filing electronic mail, and a variety of other office tasks. OfficeVision replaces PROFS

Offload host. Any device that is handling the TCP/IP processing for the MVS host where TCP/IP for MVS is installed. Currently, the only supported Offload host is the 3172-3.

Offload system. Represents both the MVS host where TCP/IP for MVS is installed and the Offload host that is handling the TCP/IP Offload processing.

open system. A system with specified standards and that therefore can be readily connected to other systems that comply with the same standards.

Open Systems Interconnection (OSI). (1) The interconnection of open systems in accordance with specific ISO standards. (2) The use of standardized procedures to enable the interconnection of data processing systems.

Operating System/2 (OS/2). Pertaining to the IBM licensed program that can be used as the operating system for personal computers. The OS/2 licensed program can perform multiple tasks at the same time.

OS/2. Operating System/2.

OSF/Motif. OSF/Motif is an X Window System toolkit defined by Open Software Foundation, Inc. (OSF), which enables the application programmer to include standard graphic elements that have a 3-D appearance. Performance of the graphic elements is increased with gadgets and windowless widgets.

OSI. Open Systems Interconnection.

out-of-band data. Data that is placed in a secondary channel for transmission. Primary and secondary communication channels are created physically by modulation on a different frequency, or logically by specifying a different logical channel. A primary channel can have a greater capacity than a secondary one.

OV. OfficeVision.

P

packet. A sequence of binary digits, including data and control signals, that is transmitted and switched as a composite whole.

Packet Switching Data Network (PSDN). A network that uses packet switching as a means of transmitting data.

parameter. A variable that is given a constant value for a specified application.

parse. To analyze the operands entered with a command.

passive open. The state of a connection that is prepared to provide a service on demand. Contrast with *active open*.

Partitioned data set (PDS). A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

PC. Personal computer.

PCA. Personal Channel Attach.

PC Network. A low-cost, broadband network that allows attached IBM personal computers, such as IBM 5150 Personal Computers, IBM Computer ATs, IBM PC/XTs, and IBM Portable Personal Computers to communicate and to share resources.

PDS. Partitioned data set.

PDN. Public Data Network.

PDU. Protocol data unit.

peer-to-peer. In network architecture, any functional unit that resides in the same layer as another entity.

Personal Channel Attach (PCA). see Personal System Channel Attach.

Personal Computer (PC). A microcomputer primarily intended for stand-alone use by an individual.

Personal System Channel Attach (PSCA). An adapter card to connect a micro-channel based personal computer (or processor) to a System/370 parallel channel.

physical layer. Layer 1 of the Open Systems Interconnection (OSI) model; it details protocols governing transmission media and signals.

physical unit (PU). In SNA, the component that manages and monitors the resources, such as attached links and adjacent link stations, associated with a node, as requested by an SSPC via an SSPC-PU session. An SSPC activates a session with the physical unit in order to indirectly manage, through the PU, resources of the node such as attached links.

PING. The command that sends an ICMP Echo Request packet to a host, gateway, or router with the expectation of receiving a reply.

PM. Presentation Manager.

PMANT. In OS/2, the 3270 client terminal emulation program that is invoked by the PMANT command.

polling. (1) On a multipoint connection or a point-to-point connection, the process whereby data stations are invited one at a time to transmit. (2) Interrogation of devices for such purposes as to avoid contention, to determine operational status, or to determine readiness to send or receive data.

POP. Post Office Protocol.

port. (1) An endpoint for communication between devices, generally referring to a logical connection. (2) A 16-bit number identifying a particular Transmission Control Protocol or User Datagram Protocol resource within a given TCP/IP node.

PORTMAP. Synonymous with *Portmapper*.

Portmapper. A program that maps client programs to the port numbers of server programs. Portmapper is used with Remote Procedure Call (RPC) programs.

Post Office Protocol (POP). A protocol used for exchanging network mail.

presentation layer. Layer 6 of the Open Systems Interconnections (OSI) model; it defines protocols governing data formats and conversions.

Presentation Manager (PM). A component of OS/2 that provides a complete graphics-based user interface, with pull-down windows, action bars, and layered menus.

principal name. Specifies the unique name of a user (client) or service.

PostScript. A standard that defines how text and graphics are presented on printers and display devices.

process. (1) A unique, finite course of events defined by its purpose or by its effect, achieved under defined conditions. (2) Any operation or combination of operations on data. (3) A function being performed or waiting to be performed. (4) A program in operation; for example, a daemon is a system process that is always running on the system.

Professional Office Systems (PROFS). IBM's proprietary, integrated office management system used for sending, receiving, and filing electronic mail, and a variety of other office tasks. PROFS has been replaced by OfficeVision. See *OfficeVision*.

PROFS. Professional Office Systems.

protocol. A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. Protocols can determine low-level details of machine-to-machine interfaces, such as the order in which bits from a byte are sent; they can also determine high-level exchanges between application programs, such as file transfer.

Protocol data unit (PDU). A set of commands used by the SNMP agent to request management station data.

protocol suite. A set of protocols that cooperate to handle the transmission tasks for a data communication system.

PSCA. Personal System Channel Attach.

PSDN. Packet Switching Data Network.

PU. Physical unit.

Public Data Network (PDN). A network established and operated by a telecommunication administration or by a Recognized Private Operating Agency (RPOA) for the specific purpose of providing circuit-switched, packet-switched, and leased-circuit services to the public.

Q

queue. A line or list formed by items in a system waiting for service; for example, tasks to be performed or messages to be transmitted. To arrange in, or form, a queue.

R

RACF. Resource access control facility.

RARP. Reverse Address Resolution Protocol.

read-only access. An access mode associated with a virtual disk directory that lets a user read, but not write or update, any file on the disk directory.

read/write access. An access mode associated with a virtual disk directory that lets a user read and write any file on the disk directory (if write authorized).

realm. One of the three parts of a Kerberos name. The realm specifies the network address of the principal name or instance. This address must be expressed as a fully qualified domain name, not as a “dot numeric” internet address.

recursion. A process involving numerous steps, in which the output of each step is used for the successive step.

reduced instruction-set computer (RISC). A computer that uses a small, simplified set of frequently used instructions for rapid execution.

reentrant. The attribute of a program or routine that allows the same copy of a program or routine to be used concurrently by two or more tasks.

Remote Execution Protocol (REXEC). A protocol that allows the execution of a command or program on a foreign host. The local host receives the results of the command execution. This protocol uses the REXEC command.

remote host. A machine on a network that requires a physical link to interconnect with the network.

remote logon. The process by which a terminal user establishes a terminal session with a remote host.

Remote Procedure Call (RPC). A facility that a client uses to request the execution of a procedure call from a server. This facility includes a library of procedures and an eXternal data representation.

Remote Spooling Communications Subsystem (RSCS). An IBM-licensed program that transfers spool files, commands, and messages between VM users, remote stations, and remote and local batch systems, through HASP-compatible telecommunication facilities.

Request For Comments (RFC). A series of documents that covers a broad range of topics affecting internetwork communication. Some RFCs are established as internet standards.

resolver. A program or subroutine that obtains information from a name server or local table for use by the calling program.

resource access control facility (RACF). An IBM-licensed program that provides for access control by identifying and by verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources.

resource records. Individual records of data used by the Domain Name System. Examples of resource records include the following: a host’s Internet Protocol addresses, preferred mail addresses, and aliases.

response unit (RU). In SNA, a message unit that acknowledges a request unit. It may contain prefix information received in a request unit. If positive, the response unit may contain additional information such as session parameters in response to BIND SESSION. If negative, it contains sense data defining the exception condition.

Restructured Extended Executor (REXX) language. A general purpose programming language, particularly suitable for EXEC procedures, XEDIT macros, or programs for personal computing. Procedures, XEDIT macros, and programs written in this language can be interpreted by the Procedures Language VM/REXX interpreter.

return code. (1) A code used to influence the execution of succeeding instructions. (2) A value returned to a program to indicate the results of an operation requested by that program.

Reverse Address Resolution Protocol (RARP). A protocol that maintains a database of mappings between physical hardware addresses and IP addresses.

REXEC. Remote Execution Protocol.

REXX. Restructured Extended Executor language.

RFC. Request For Comments.

RIP. Routing Information Protocol.

RISC. Reduced instruction-set computer.

router. A device that connects networks at the ISO Network Layer. A router is protocol-dependent and connects only networks operating the same protocol. Routers do more than transmit data; they also select the best transmission paths and optimum sizes for packets. In TCP/IP, routers operate at the Internetwork layer. See also *gateway*.

Routing Information Protocol (RIP). The protocol that maintains routing table entries for gateways, routers, and hosts.

routing table. A list of network numbers and the information needed to route packets to each.

RPC. Remote Procedure Call.

RSCS. Remote Spooling Communications Subsystem.

RU. Response unit.

S

SAA. Systems Application Architecture.

SBCS. Single Byte Character Set.

SDLC. Synchronous data link control.

Sendmail. The OS/2 mail server that uses Simple Mail Transfer Protocol to route mail from one host to another host on the network.

serial line. A network media that is a de facto standard, not an international standard, commonly used for point-to-point TCP/IP connections. Generally, a serial line consists of an RS-232 connection into a modem and over a telephone line.

semantics. (1) The relationships of characters or groups of characters to their meanings, independent of the manner of their interpretation and use. (2) The relationships between symbols and their meanings.

server. A function that provides services for users. A machine can run client and server processes at the same time.

SFS. Shared File System.

Shared File System (SFS). A part of CMS that lets users organize their files into groups known as *directories* and selectively share those files and directories with other users.

Simple Mail Transfer Protocol (SMTP). A TCP/IP application protocol used to transfer mail between users on different systems. SMTP specifies how mail systems interact and the format of control messages they use to transfer mail.

Simple Network Management Protocol (SNMP). A protocol that allows network management by elements, such as gateways, routers, and hosts. This protocol provides a means of communication between network elements regarding network resources.

simultaneous peripheral operations online (SPOOL). (1) (Noun) An area of auxiliary storage defined to temporarily hold data during its transfer between peripheral equipment and the processor. (2) (Verb) To use auxiliary storage as a buffer storage to reduce processing delays when transferring data between peripheral equipment and the processing storage of a computer.

single-byte character set (SBCS). A character set in which each character is represented by a one-byte code. Contrast with double-byte character set.

SMI. Structure for Management Information.

SMTP. Simple Mail Transfer Protocol.

SNA. Systems Network Architecture.

SNALINK. SNA Network Link.

SNA Network Link. An SNA network link function of TCP/IP for VM and MVS hosts running TCP/IP to communicate through an existing SNA backbone.

SNMP. Simple Network Management Protocol.

SOA. Start of authority record.

socket. (1) An endpoint for communication between processes or applications. (2) A pair consisting of TCP port and IP address, or UDP port and IP address.

socket address. An address that results when the port identification number is combined with an internet address.

socket interface. An application interface that allows users to write their own applications to supplement those supplied by TCP/IP.

SPOOL. Simultaneous peripheral operations online.

spooling. The processing of files created by or intended for virtual readers, punches, and printers. The spool files can be sent from one virtual device to another, from one virtual machine to another, and to read devices.

SQL. Structured Query Language.

SQL/DS. Structured Query Language/Data System.

start of authority record (SOA). In the Domain Name System, the resource record that defines a zone.

stream. A continuous sequence of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format.

Structured Query Language (SQL). Fourth generation English-like programming language used to perform queries on relational databases.

Structured Query Language/Data System (SQL/DS). An IBM relational database management system for the VM and VSE operating systems.

Structure for Management Information (SMI). The rules used to define the objects that can be accessed through a network management protocol. See also *MIB*.

subagent. In the SNMP architecture, a subagent provides an extension to the utility provided by the SNMP agent.

subdirectory. A directory contained within another directory in a file system hierarchy.

subnet. A networking scheme that divides a single logical network into smaller physical networks to simplify routing.

subnet address. The portion of the host address that identifies a subnetwork.

subnet mask. A mask used in the IP protocol layer to separate the subnet address from the host portion of the address.

subnetwork. Synonymous with *subnet*.

subsystem. A secondary or subordinate system, usually capable of operating independent of, or asynchronously with, a controlling system.

SYNC. Synchronous.

synchronous (SYNC). (1) Pertaining to two or more processes that depend on the occurrences of a specific event such as common timing signal. (2) Occurring with a regular or predictable time relationship. See *asynchronous*.

synchronous data link control (SDLC). A data link over which communication is conducted using the synchronous data protocol.

Systems Application Architecture (SAA). A formal set of rules that enables applications to be run without modification in different computer environments.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks.

T

TALK. An interactive messaging system that sends messages between the local host and a foreign host.

TCP. Transmission Control Protocol.

TCP/IP. Transmission Control Protocol/Internet Protocol.

Telnet. The Terminal Emulation Protocol, a TCP/IP application protocol for remote connection service. Telnet allows a user at one site to gain access to a foreign host as if the user's terminal were connected directly to that foreign host.

terminal emulator. A program that imitates the function of a particular kind of terminal.

Terminate and Stay Resident (TSR) program. A TSR is a program that installs part of itself as an extension of DOS when it is executed.

TFTPD. Trivial File Transfer Protocol Daemon.

ticket. Encrypted information obtained from a Kerberos authentication server or a ticket-granting server. A ticket authenticates a user and, in conjunction with an authenticator, serves as permission to access a service when presented by the authenticated user.

ticket-granting server. Grants Kerberos tickets to authenticated users as permission to access an end-service.

Time Sharing Option (TSO). An operating system option; for System/370 system, the option provides interactive time sharing from remote terminals

time stamp. (1) To apply the current system time. (2) The value on an object that is an indication of the system time at some critical point in the history of the object. (3) In query, the identification of the day and time when a query report was created that query automatically provides on each report.

TN3270. An informally defined protocol for transmitting 3270 data streams over Telnet.

token. In a local network, the symbol of authority passed among data stations to indicate the station temporarily in control of the transmission medium.

token-bus. See *bus topology*.

token ring. As defined in IEEE 802.5, a communication method that uses a token to control access to the LAN. The difference between a token bus and a token ring is that a token-ring LAN does not use a master controller to control the token. Instead, each computer knows the address of the computer that should receive the token next. When a computer with the token has nothing to transmit, it passes the token to the next computer in line.

token-ring network. A ring network that allows unidirectional data transmission between data stations by a token-passing procedure over one transmission medium, so that the transmitted data returns to the transmitting station.

Transmission Control Protocol (TCP). The TCP/IP layer that provides reliable, process-to-process data stream delivery between nodes in interconnected computer networks. TCP assumes that IP (Internet Protocol) is the underlying protocol.

Transmission Control Protocol/Internet Protocol (TCP/IP). A suite of protocols designed to allow communication between networks regardless of the technologies implemented in each network.

transport layer. Layer 4 of the Open Systems Interconnection (OSI) model; it defines protocols governing message structure and some error checking.

TRAP. An unsolicited message that is sent by an SNMP agent to an SNMP network management station.

Trivial File Transfer Protocol Daemon (TFTPD). The TFTP daemon (TFTPD server) transfers files between the Byte File System (BFS) and TFTP clients. TFTPD supports access to files maintained in a BFS directory structure that is mounted.

TSO. Time Sharing Option.

TSR. Terminate and stay resident. TSR usually refers to a terminate-and-stay-resident program.

U

UDP. User Datagram Protocol.

user. A function that uses the services provided by a server. A host can be a user and a server at the same time. See *client*.

User Datagram Protocol (UDP). A datagram level protocol built directly on the IP layer. UDP is used for application-to-application programs between TCP/IP hosts.

user exit. A point in an IBM-supplied program at which a user routine may be given control.

user profile. A description of a user, including user ID, user name, defaults, password, access authorization, and attributes.

V

virtual address. The address of a location in virtual storage. A virtual address must be translated into a real address to process the data in processor storage.

Virtual Machine (VM). Licensed software whose full name is Virtual Machine/Enterprise Systems Architecture (VM/ESA) It is a software operating system that manages the resources of a real processor to provide virtual machines to end users. It includes time-sharing system control program (CP), the conversational monitor system (CMS), the group control system (GCS), and the dump viewing facility (DVF).

Virtual Machine Communication Facility (VMCF). A connectionless mechanism for communication between address spaces.

Virtual Machine/System Product (VM/SP). An IBM-licensed program that manages the resources of a single computer so that multiple computing systems appear to exist. Each virtual machine is the functional equivalent of a *real* machine.

virtual storage. Storage space that can be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computing system and by the amount of auxiliary storage available, not by the actual number of main storage locations.

Virtual Telecommunications Access Method (VTAM). An IBM-licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VM. Virtual Machine.

VMCF. Virtual Machine Communication Facility.

VM/ESA. Virtual Machine/Enterprise System Architecture

VMSES/E. Virtual Machine Serviceability Enhancements Staged/Extended.

VTAM. Virtual Telecommunications Access Method.

W

WAN. Wide area network.

well-known port. A port number that has been preassigned for specific use by a specific protocol or application. Clients and servers using the same protocol communicate over the same well-known port.

wide area network (WAN). A network that provides communication services to a geographic area larger than that served by a local area network.

widget. The basic data type of the X Window System Toolkit. Every widget belongs to a widget class that contains the allowed operations for that corresponding class.

window. An area of the screen with visible boundaries through which a panel or portion of a panel is displayed.

working directory. The directory in which an application program is found. The working directory becomes the current directory when the application is started.

X

X Client. An application program which uses the X protocol to communicate windowing and graphics requests to an X Server.

XDR. eXternal Data Representation.

XEDIT. The CMS facility, containing the XEDIT command and XEDIT subcommands and macros, that lets a user create, change, and manipulate CMS files.

X Server. A program which interprets the X protocol and controls one or more screens, a pointing device, a keyboard, and various resources associated with the X Window System, such as Graphics Contexts, Pixmaps, and color tables.

X Window System. The X Window System is a protocol designed to support network transparent windowing and graphics. TCP/IP for VM and MVS provides client support for the X Window System application program interface.

X Window System API. An application program interface designed as a distributed, network-transparent, device-independent, windowing and graphics system.

X Window System Toolkit. Functions for developing application environments.

X.25. A CCITT communication protocol that defines the interface between data terminal equipment and packet switching networks.

X.25 NCP Packet Switching Interface (X.25 NPSI). An IBM-licensed program that allows users to communicate over packet switched data networks that have interfaces complying with Recommendation X.25 (Geneva** 1980) of the CCITT. It allows SNA programs to communicate with SNA equipment or with non-SNA equipment over such networks.

Z

ZAP. To modify or dump an individual text file/data set using the ZAP command or the ZAPTEXT EXEC.

ZAP disk. The virtual disk in the VM operating system that contains the user-written modifications to VTAM code.

zone. In the Domain Name System, a zone is a logical grouping of domain names that is assigned to a particular organization. Once an organization controls its own zone, it can change the data in the zone, add new tree sections connected to the zone, delete existing nodes, or delegate new subzones under its zone.

Bibliography

This bibliography lists the publications that provide information about your VM/ESA system. The VM/ESA library includes VM/ESA base publications, publications for additional facilities included with VM/ESA, and publications for VM/ESA optional features.

VM/ESA publications may be available as Adobe Portable Document Format (PDF) files, IBM BookManager® files, or printed books. For abstracts of VM/ESA publications and other library-related information, including current editions and available publication formats, see *VM/ESA: General Information*.

VM/ESA Base Publications

Evaluation

VM/ESA: Licensed Program Specifications, GC24-5744

VM/ESA: General Information, GC24-5745

Installation and Service

VM/ESA: Installation Guide, GC24-5836

VM/ESA: Service Guide, GC24-5838

VM/ESA: VMSES/E Introduction and Reference, GC24-5837

Planning and Administration

VM/ESA: Planning and Administration, SC24-5750

VM/ESA: CMS File Pool Planning, Administration, and Operation, SC24-5751

VM/ESA: Conversion Guide and Notebook, GC24-5839

VM/ESA: REXX/EXEC Migration Tool for VM/ESA, GC24-5752

VM/ESA: Running Guest Operating Systems, SC24-5755

VM/ESA: Connectivity Planning, Administration, and Operation, SC24-5756

VM/ESA: Group Control System, SC24-5757

VM/ESA: Performance, SC24-5782

Customization

IBM VM/ESA: CP Exit Customization, SC24-5672

Operation

VM/ESA: System Operation, SC24-5758

VM/ESA: Virtual Machine Operation, SC24-5759

Application Programming

VM/ESA: CP Programming Services, SC24-5760

VM/ESA: CMS Application Development Guide, SC24-5761

VM/ESA: CMS Application Development Reference, SC24-5762

VM/ESA: CMS Application Development Guide for Assembler, SC24-5763

VM/ESA: CMS Application Development Reference for Assembler, SC24-5764

VM/ESA: CMS Application Multitasking, SC24-5766

VM/ESA: REXX/VM Primer, SC24-5598

VM/ESA: REXX/VM User's Guide, SC24-5465

VM/ESA: REXX/VM Reference, SC24-5770

IBM VM/ESA: Distributed Graphical User Interface Toolkit, SC24-5724

IBM VM/ESA: Reusable Server Kernel Programmer's Guide and Reference, SC24-5852

VM/ESA: Enterprise Systems Architecture/Extended Configuration Principles of Operation, SC24-5594

VM/ESA: Programmer's Guide to the Server-Requester Programming Interface for VM, SC24-5455

VM/ESA: CPI Communications User's Guide, SC24-5595

Common Programming Interface Communications Reference, SC26-4399

Common Programming Interface Resource Recovery Reference, SC31-6821

External Security Interface (RACROUTE) Macro Reference for MVS and VM, GC28-1366

End Use

VM/ESA: CP Command and Utility Reference, SC24-5773

VM/ESA: CMS Primer, SC24-5458

VM/ESA: CMS User's Guide, SC24-5775

VM/ESA: CMS Command Reference, SC24-5776

IBM VM/ESA: Graphical User Interface Facility, SC24-5789

VM/ESA: CMS Pipelines User's Guide, SC24-5777

VM/ESA: CMS Pipelines Reference, SC24-5778

CMS/TSO Pipelines: Author's Edition, SL26-0018

VM/ESA: XEDIT User's Guide, SC24-5779

VM/ESA: XEDIT Command and Macro Reference, SC24-5780

VM/ESA: Quick Reference, SX24-5290

Diagnosis

VM/ESA: System Messages and Codes, GC24-5841

VM/ESA: Dump Viewing Facility, GC24-5853

VM/ESA: Diagnosis Guide, GC24-5854

VM/ESA: CP Diagnosis Reference, SC24-5855

VM/ESA: CP Diagnosis Reference Summary, SX24-5292

VM/ESA: CMS Diagnosis Reference, SC24-5857

Note: CP and CMS control block information is not provided in book form. This information is available on the IBM VM/ESA operating system home page (http://www.ibm.com/s390/vm).

Publications for Additional Facilities

OpenEdition® for VM/ESA

IBM OpenEdition for VM/ESA: POSIX Conformance Document, GC24-5842

IBM OpenEdition for VM/ESA: User's Guide, SC24-5727

IBM OpenEdition for VM/ESA: Command Reference, SC24-5728

IBM OpenEdition for VM/ESA: Advanced Application Programming Tools, SC24-5729

IBM OpenEdition for VM/ESA: Callable Services Reference, SC24-5726

IBM OpenEdition for VM/ESA: Sockets Reference, SC24-5741

IBM C for VM/ESA: Library Reference, SC23-3908

Debug Tool User's Guide and Reference, SC09-2137

DFSMS/VM®

VM/ESA: DFSMS/VM Function Level 221 Planning Guide, GC35-0121

VM/ESA: DFSMS/VM Function Level 221 Installation and Customization, SC26-4704

VM/ESA: DFSMS/VM Function Level 221 Storage Administration Guide and Reference, SH35-0111

VM/ESA: DFSMS/VM Function Level 221 Removable Media Services User's Guide and Reference, SC35-0141

VM/ESA: DFSMS/VM Function Level 221 Messages and Codes, SC26-4707

VM/ESA: DFSMS/VM Function Level 221 Diagnosis Guide, LY27-9589

S/390® Open Systems Adapter Support Facility for VM/ESA

Planning for the System/390 Open Systems Adapter Feature, GC23-3870

IBM VM/ESA: Open Systems Adapter Support Facility User's Guide, SC28-1992

Language Environment®

Language Environment for OS/390 & VM: Concepts Guide, GC28-1945

Language Environment for OS/390 & VM: Migration Guide, SC28-1944

Language Environment for OS/390 & VM: Programming Guide, SC28-1939

Language Environment for OS/390 & VM: Programming Reference, SC28-1940

Language Environment for OS/390 & VM: Writing Interlanguage Communication Applications, SC28-1943

Language Environment for OS/390 & VM: Debugging Guide and Run-Time Messages, SC28-1942

Publications for Optional Features

CMS Utilities Feature

VM/ESA: CMS Utilities Feature, SC24-5535

TCP/IP Feature for VM/ESA

VM/ESA: TCP/IP Function Level 320 Planning and Customization, SC24-5847

VM/ESA: TCP/IP Function Level 320 User's Guide, SC24-5848

VM/ESA: TCP/IP Function Level 320 Programmer's Reference, SC24-5849

VM/ESA: TCP/IP Function Level 320 Messages and Codes, GC24-5850

VM/ESA: TCP/IP Function Level 320 Diagnosis Guide, GC24-5851

OpenEdition Distributed Computing Environment Feature for VM/ESA

IBM OpenEdition DCE for VM/ESA: Introducing the OpenEdition Distributed Computing Environment, SC24-5735

IBM OpenEdition DCE for VM/ESA: Planning, SC24-5737

IBM OpenEdition DCE for VM/ESA: Configuring and Getting Started, SC24-5734

IBM OpenEdition DCE for VM/ESA: Administration Guide, SC24-5730

IBM OpenEdition DCE for VM/ESA: Administration Reference, SC24-5731

IBM OpenEdition DCE for VM/ESA: Application Development Guide, SC24-5732

IBM OpenEdition DCE for VM/ESA: Application Development Reference, SC24-5733

IBM OpenEdition DCE for VM/ESA: User's Guide, SC24-5738

IBM OpenEdition DCE for VM/ESA: Messages and Codes, SC24-5736

LAN File Services/ESA

Discovering LAN File Services/ESA, GK2T-5762

Introducing LAN File Services/ESA, GH24-5259

LAN File Services/ESA: Licensed Program Specifications, GH24-5260

LAN File Services/ESA: VM Guide and Reference, SH24-5264

LAN Resource Extension and Services/VM

LAN Resource Extension and Services/VM: Licensed Program Specifications, GC24-5617

LAN Resource Extension and Services/VM: General Information, GC24-5618

LAN Resource Extension and Services/VM: Guide and Reference, SC24-5622

CD-ROM

The following CD-ROM contains PDF versions of many VM/ESA publications and publications for some related IBM licensed programs. It also contains all the IBM libraries that are available in IBM BookManager format for current VM system products and current IBM licensed programs that run on VM/ESA.

IBM Online Library Omnibus Edition: VM Collection, SK2T-2067

Note: Only unlicensed publications are included.

Other TCP/IP Related Publications

This section lists other publications, outside the VM/ESA 2.4.0 library, that you may find helpful.

TCP/IP Tutorial and Technical Overview, GG24-3376

TCP/IP Illustrated, Volume 1: The Protocols, SR28-5586

Internetworking with TCP/IP Volume I: Principles, Protocols, and Architecture, SC31-6144

Internetworking With TCP/IP Volume II: Implementation and Internals, SC31-6145

Internetworking With TCP/IP Volume III: Client-Server Programming and Applications, SC31-6146

DNS and BIND in a Nutshell, SR28-4970

"MIB II Extends SNMP Interoperability," C. Vanderberg, *Data Communications*, October 1990.

"Network Management and the Design of SNMP," J.D. Case, J.R. Davin, M.S. Fedor, M.L. Schoffstall.

"Network Management of TCP/IP Networks: Present and Future," A. Ben-Artzi, A. Chandna, V. Warrier.

"Special Issue: Network Management and Network Security,"*ConneXions-The Interoperability Report*, Volume 4, No. 8, August 1990.

The Art of Distributed Application: Programming Techniques for Remote Procedure Calls, John R. Corbin, Springer-Verlog, 1991.

The Simple Book: An Introduction to Management of TCP/IP-based Internets, Marshall T Rose, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

Index

Numerics

Readers' Comments

Virtual Machine/
Enterprise Systems Architecture
TCP/IP Function Level 320
Diagnosis Guide
Version 2 Release 4.0

Publication No. GC24-5851-01

You may use this form to report errors, to suggest improvements, or to express your opinion on the appearance, organization, or completeness of this book.

Date: ________________________________

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note

Report system problems to your IBM representative or the IBM branch office serving you.
U.S. customers can order publications by calling the IBM Software Manufacturing Solutions at **1-800-879-2755**.

In addition to using this postage-paid form, you may send your comments by:

FAX	1-607-752-2327	Internet	pubrcf@vnet.ibm.com
IBM Mail	USIB2L8Z@IBMMAIL	IBMLink	GDLVME(PUBRCF)

Would you like a reply? ___**YES** ___ **NO** If yes, please tell us the type of response you prefer.

___ Electronic address: ________________________________

___ FAX number: ________________________________

___ Mail: (Please fill in your name and address below.)

Name ________________________________ Address ________________________________

Company or Organization ________________________________

Phone No. ________________________________

Readers' Comments
GC24-5851-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Department G60
International Business Machines Corporation
Information Development
1701 North Street
ENDICOTT NY 13760-5553

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

File Number: S370/4300/30XX-50
Program Number: 5654-030

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC24-5851-01

Spine information:

VM/ESA TCP/IP Function Level 320 Diagnosis Guide *Version 2 Release 4.0*