

z/VM
7.3

TCP/IP Diagnosis Guide



Note:

Before you use this information and the product it supports, read the information in [“Notices” on page 213.](#)

This edition applies to version 7, release 3 of IBM® z/VM® (product number 5741-A09) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-06-17

© **Copyright International Business Machines Corporation 1987, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures.....	ix
Tables.....	xv
About This Document.....	xvii
Intended Audience.....	xvii
Conventions and Terminology.....	xvii
How the Term “internet” Is Used in This Document.....	xvii
How Numbers Are Used in This Document.....	xvii
Syntax, Message, and Response Conventions.....	xvii
Where to Find More Information.....	xx
Links to Other Documents and Websites.....	xxii
How to provide feedback to IBM.....	xxiii
Summary of Changes for z/VM: TCP/IP Diagnosis Guide.....	xxv
GC24-6328-73, z/VM 7.3 (June 2025)	xxv
GC24-6328-73, z/VM 7.3 (September 2023).....	xxv
GC24-6328-73, z/VM 7.3 (September 2022).....	xxv
Chapter 1. Diagnosis Overview.....	1
Chapter 2. Problem Identification.....	3
Categories that Help Identify the Problem.....	3
Abend.....	4
Message.....	4
Loop.....	5
Wait State.....	6
Incorrect Output.....	6
Performance.....	7
Documentation.....	8
Guidelines for Machine Readable Documentation.....	9
Necessary Documentation.....	10
Additional Documentation.....	10
Problem Resolution.....	11
Severe Problem Resolution.....	11
Customer Worksheet.....	11
Problem Category.....	11
Background Information.....	12
Additional Information.....	12
Chapter 3. TCP/IP VM Structures and Internetworking Overview.....	13
VM Structure.....	13
Virtual Machines.....	13
Virtual Machine Communication Facility.....	14
Inter-User Communication Vehicle.....	15
*CCS and Logical Device Service Facility.....	15
Overview of Internetworking.....	15
Bridges.....	17

Maximum Transmission Unit (MTU).....	17
Token Ring IEEE 802.5.....	18
IEEE 802.3.....	19
Ethernet - DIX V2.....	20
Sub-Network Access Protocol (SNAP).....	20
Internet Addressing.....	21
Direct Routing.....	24
Indirect Routing.....	25
Simplified IP Datagram Routing Algorithm.....	25
Subnetting.....	25
Simplified IP Datagram Routing Algorithm with Subnets.....	26
Static Routing.....	27
Dynamic Routing.....	28
Dynamic Routing Tables.....	28
Example of Network Connectivity.....	29
Chapter 4. Server Initialization.....	31
CMS Servers.....	31
Diagnosis Method 1.....	31
Diagnosis Method 2.....	31
GCS Servers.....	31
Chapter 5. TCP/IP Procedures.....	33
TCP/IP Internals.....	33
Internal Procedures.....	33
Queues.....	35
Internal Activities.....	36
Input/Output.....	39
IUCV Links.....	39
Chapter 6. Diagnosing the Problem.....	41
Unable to Connect to TCP/IP Node.....	41
Description of the Problem.....	41
Symptom.....	41
Problem Determination.....	41
PING — Sending an Echo Request to a Foreign Host.....	42
Resolving the PING Command Problems.....	42
Chapter 7. TCP/IP Traces.....	45
Debugging in VM.....	45
Executing Traces.....	45
Activating Traces.....	45
First-Level Trace.....	45
Second-Level Trace.....	46
Directing Output.....	47
TCP/IP Packet Tracing.....	47
Native TCP/IP Stack Packet Trace.....	47
TCP/IP Stack Packet Trace with TRSOURCE.....	48
Process Names.....	49
Single Process Names.....	49
Group Process Names.....	91
Commonly Used Trace Options.....	98
Connection State.....	104
Connection State As Known by TCP.....	104
Connection State As Known by Pascal or VMCF Applications.....	106
Connection State As Known by Socket Applications.....	106
Traceroute Function (TRACERTE).....	107

Chapter 8. Using IPFORMAT Packet Trace Formatting Tool.....	109
IPFORMAT Command Overview.....	109
IPFORMAT Command.....	109
IPFORMAT Configuration File.....	111
Using IPFORMAT to View Packet Data.....	112
The Packet Summary View.....	112
The Packet Detail View.....	114
IPFORMAT VIEW Function Keys.....	117
Packet Summary PF Keys.....	117
Packet Detail PF Keys.....	118
IPFORMAT Subcommands.....	119
FILTER Subcommand.....	119
VIEW Subcommand.....	121
HEADER Subcommand.....	121
SAVE Subcommand.....	122
APPEND Subcommand.....	123
Chapter 9. FTP Traces.....	125
FTP Connection.....	125
FTP Client Traces.....	127
Activating Traces.....	127
Trace Output.....	127
FTP Server Traces.....	132
Activating Traces.....	132
Trace Output.....	133
Chapter 10. Simple Mail Transfer Protocol Traces.....	137
SMTP Client Traces.....	137
Activating Traces.....	137
Obtaining Queue Information.....	137
SMTP Server Traces.....	138
Activating Traces.....	138
Chapter 11. RPC Programs.....	145
General Information about RPC.....	145
RPC Call Messages.....	145
RPC Reply Messages.....	146
Accepted Reply Messages.....	146
Rejected Reply Messages.....	147
RPC Support.....	148
Portmapper.....	148
Portmapper Procedures.....	148
Chapter 12. Diagnosing MPRoute Problems.....	149
Categorizing MPRoute Problems.....	149
Abends.....	149
MPRoute Connection Problems.....	149
Routing Failures.....	150
Using Privileged MPRoute SMSG Commands.....	151
MPRoute Traces and Debug Information.....	151
Starting MPRoute Tracing and Debugging from the z/VM Console.....	151
Starting MPRoute Tracing and Debugging using the SMSG Command.....	152
Destination of MPRoute Trace and Debug Output.....	153
Sample MPRoute Trace Output.....	153

Chapter 13. SSL Server Diagnosis.....	163
SSL component Flow.....	164
SSL Server Traces.....	165
Diagnosing Problems.....	166
Symptom - The SSL Server Does Not Initialize.....	166
Symptom - Parameters Are Not Correctly Passed to the SSL Server.....	167
Symptom - Protected Application Server Shuts Down at Startup.....	167
Symptom - Connection to a Protected Application Server Cannot be Established.....	168
Symptom - Connections Close Due to Errors.....	168
Symptom - Incorrect Input or Output.....	169
Trace Output.....	169
Trace Normal.....	169
Trace Connections NODATA.....	170
Trace Connections DATA.....	171
Trace FLOW.....	172
Displaying Local Host Information.....	174
Chapter 14. Network File System.....	177
VM NFS Client Support.....	177
Activating Traces for NFS Client.....	177
VM NFS Server Support.....	177
NFS Protocol.....	177
Mount Protocol.....	177
PCNFSD Protocol.....	177
General NFS Debugging Features.....	177
Activating Traces for NFS Server.....	179
Additional Trace Options.....	179
Chapter 15. Remote Printing Traces.....	185
Remote Printing Client Traces.....	185
Activating Remote Printing Client Traces.....	185
Remote Printing Client Trace Output.....	185
Chapter 16. Remote Execution Protocol Traces.....	189
Remote Execution Protocol Client Traces.....	189
Activating Remote Execution Protocol Client Traces.....	189
Remote Execution Protocol Client Trace Output.....	189
Remote Execution Protocol Server Traces.....	190
Activating Remote Execution Protocol Server Traces.....	190
Remote Execution Protocol Server Trace Output.....	191
Chapter 17. Hardware Trace Functions.....	193
PCCA Devices.....	193
PCCA Block Structure.....	193
CCW.....	195
Matching CCW Traces and TCP/IP Traces.....	200
NETSTAT OSAINFO.....	200
Appendix A. Return Codes.....	203
TCP/IP Return Codes.....	203
UDP Error Return Codes.....	204
Appendix B. Related Protocol Specifications.....	207
Notices.....	213

Programming Interface Information.....	214
Trademarks.....	214
Terms and Conditions for Product Documentation.....	214
IBM Online Privacy Statement.....	215
Bibliography.....	217
Where to Get z/VM Information.....	217
z/VM Base Library.....	217
z/VM Facilities and Features.....	218
Prerequisite Products.....	220
Related Products.....	220
Other TCP/IP Related Publications.....	220
Index.....	223

Figures

1. Overview of the Diagnosis Procedure.....	1
2. Pascal Execution Error.....	4
3. The TCP/IP Layered Architecture for VM.....	13
4. The sequence of a Server Startup.....	14
5. Networks with a Gateway Forming an Internet.....	16
6. Routers and Bridges within an Internet.....	17
7. Relationship of MTU to Frame Size.....	18
8. Format of an IEEE 802.5 Token-Ring Frame.....	19
9. Format of an IEEE 802.3 Frame.....	20
10. Format of an Ethernet V2 Frame.....	20
11. SNAP Header.....	20
12. Classes of IP Addresses.....	21
13. Determining the Class of an IP Address.....	22
14. Routing and Bridging.....	24
15. General IP Routing Algorithm.....	25
16. Routing Algorithm with Subnets.....	27
17. Example of Resolving a Subnet Route.....	27
18. Example of Network Connectivity.....	29
19. Format of the User Field for a CONNECT Request.....	40
20. Format of the User Field for a Local IUCV CONNECT Request.....	40
21. A Sample of an ARP Trace (Part 1 of 2).....	50
22. A Sample of an ARP Trace (Part 2 of 2).....	51
23. A Sample of an ARP Trace Using MORETRACE.....	51

24. A Sample of a CCS Trace.....	52
25. A Sample of a Congestion Trace.....	52
26. A Sample of a CONSISTENCYCHECKER Trace.....	53
27. A Sample of a DENIALOFSERVICE in the TRACE Statement.....	54
28. A Sample of a DENIALOFSERVICE in the MORETRACE Statement.....	54
29. A Sample of a DROPPED in the TRACE Statement.....	55
30. A Sample of an ICMP Trace.....	55
31. A Sample of an IGMP Trace.....	56
32. A Sample of an INITIALIZE Trace Using MORETRACE (Part 1 of 2).....	57
33. A Sample of an INITIALIZE Trace Using MORETRACE (Part 2 of 2).....	58
34. A Sample of an IPDOWN Trace.....	58
35. A Sample of an IPDOWN Trace Using MORETRACE.....	59
36. A Sample of an IPUP Trace.....	59
37. A Sample of an IPUP Trace Using MORETRACE.....	59
38. A Sample of a MONITOR Trace Using MORETRACE (Part 1 of 2).....	60
39. A Sample of a MONITOR Trace Using MORETRACE (Part 2 of 2).....	61
40. A Sample of a MULTICAST Trace.....	62
41. A Sample of a NOTIFY Trace.....	63
42. A Sample of a NOTIFY Trace Using MORETRACE.....	64
43. A Sample of an OSD Trace.....	65
44. A Sample of a PARSE-TCP Trace Using MORETRACE and LESSTRACE.....	65
45. A Sample of a PING Trace.....	66
46. A Sample of a QDIO Trace.....	67
47. A Sample of a ROUNDTRIP Trace.....	67
48. A Sample of a SCHEDULER Trace.....	67

49. A Sample of a SCHEDULER Trace Using MORETRACE (Part 1 of 2).....	68
50. A Sample of a SCHEDULER Trace Using MORETRACE (Part 2 of 2).....	69
51. A Sample of a SHUTDOWN Trace.....	70
52. A Sample of an SNMPDPI Trace.....	70
53. A Sample of a SOCKET Trace.....	71
54. A Sample of an SSL Trace.....	72
55. A Sample of a TCPDOWN Trace.....	72
56. A Sample of a TCPDOWN Trace Using MORETRACE.....	73
57. A Sample of a TCPUP Trace.....	74
58. A Sample of a TCPUP Trace Using MORETRACE (Part 1 of 3).....	75
59. A Sample of a TCPUP Trace Using MORETRACE (Part 2 of 3).....	76
60. A Sample of a TCPUP Trace Using MORETRACE (Part 3 of 3).....	77
61. A Sample of a TCPREQUEST Trace.....	78
62. A Sample of a TCPREQUEST Trace Using MORETRACE.....	79
63. A Sample of a TELNET Trace (Part 1 of 2).....	82
64. A Sample of a TELNET Trace (Part 2 of 2).....	83
65. A Sample of a TELNET Trace Using MORETRACE (Part 1 of 3).....	84
66. A Sample of a TELNET Trace Using MORETRACE (Part 2 of 3).....	85
67. A Sample of a TELNET Trace Using MORETRACE (Part 3 of 3).....	86
68. A Sample of a TIMER Trace.....	87
69. A Sample of a TIMER Trace Using MORETRACE.....	88
70. A Sample of a UDPREQUEST Trace.....	89
71. A Sample of a UDPREQUEST Trace Using MORETRACE.....	90
72. A Sample of a UDPUP Trace Using MORETRACE.....	91
73. A Sample of an IUCV Trace (Part 1 of 2).....	93

74. A Sample of an IUCV Trace (Part 2 of 2).....	94
75. A Sample of a PCCA Trace (Part 1 of 2).....	95
76. A Sample of a PCCA Trace (Part 2 of 2).....	96
77. A Sample of a PCCA Trace Using MORETRACE (Part 1 of 2).....	97
78. A Sample of a PCCA Trace Using MORETRACE (Part 2 of 2).....	98
79. Packet Summary of IPv4 Packets.....	113
80. Packet Summary of a mix of IPv4 and IPv6 packets.....	114
81. Packet Detail of an ICMP Packet (Part 1 of 3).....	115
82. Packet Detail of an ICMP Packet (Part 2 of 3).....	116
83. Packet Detail of an ICMP Packet (Part 3 of 3).....	117
84. The FTP Model.....	125
85. A Sample of an FTP Client Trace (Part 1 of 2).....	129
86. A Sample of an FTP Client Trace (Part 2 of 2).....	130
87. A Sample of an FTP Server Trace (Part 1 of 4).....	133
88. A Sample of an FTP Server Trace (Part 2 of 4).....	134
89. A Sample of an FTP Server Trace (Part 3 of 4).....	135
90. A Sample of an FTP Server Trace (Part 4 of 4).....	136
91. Sample Outout form a Mail Queue Query.....	137
92. SMTP Reply Codes.....	140
93. A Sample of an SMTP Server Trace Using the DEBUG Statement.....	141
94. Sample LOG Output.....	141
95. A Sample of an SMTP Resolver Trace.....	142
96. A Sample of a Notification Trace.....	143
97. A Sample of a Connection Activity Trace.....	143
98. RPC Call Message Structure.....	146

99. Structure of an RPC Accepted Reply Message.....	147
100. Structure of an RPC Rejected Reply Message.....	147
101. SSL Client and Server Environment.....	163
102. TCP/IP Stack View of connection.....	164
103. SSL processing flow.....	164
104. A Sample of an NFS Trace of a Bad Mount.....	183
105. A Sample of an LPR Client Trace (Part 1 of 2).....	186
106. A Sample of an LPR Client Trace (Part 2 of 2).....	187
107. A Sample of a Remote Execution Client Trace.....	190
108. A Sample of a Remote Execution Protocol Server Trace.....	191
109. PCCA Block Structure.....	193
110. A Sample of a PCCA Control Message Block.....	193
111. PCCA Control Message Structure.....	194
112. PCCA LAN Messages Structure.....	195
113. Common Layout of a Token-Ring Packet.....	195
114. A Sample of an ARP Frame on a PCCA Token-Ring.....	197
115. A Sample of an IP/ICMP Packet on a PCCA Token-Ring.....	198
116. A Sample of a VM/SP4-5 CCW Trace.....	199
117. IP Header Format.....	199
118. TCP Header Format.....	200

Tables

1. Examples of Syntax Diagram Conventions.....	xviii
2. Usage of TCP/IP for z/VM Applications, Functions, and Protocols.....	xx
3. TCP/IP Component ID Number.....	10
4. Relationship between RC Field and Maximum I-Field Value.....	19
5. IPv6 Address Format.....	23
6. TCP/IP Internal Procedures.....	33
7. TCPIP Queues.....	35
8. TCP/IP Internal Activities.....	36
9. Telnet Commands from RFC 854.....	80
10. Telnet Command Options from RFC 1060.....	80
11. Commonly-used Trace Options.....	99
12. TCP Connection States.....	104
13. Connection Pseudo-states.....	106
14. Packet Summary PF Keys.....	117
15. Packet Detail PF Keys.....	118
16. SMTP Commands.....	138
17. RPC Credentials.....	146
18. RPC Accept_stat Values.....	147
19. RPC Auth_stat Values.....	148
20. Portmapper Procedures.....	148
21. PCCA CCW Codes.....	196
22. TCP/IP Return Codes Sent to Servers and Clients.....	203
23. UDP Error Return Codes.....	204

About This Document

This document provides information for diagnosing problems that occur in the IBM z/VM Transmission Control Protocol/Internet Protocol (TCP/IP) networks.

Intended Audience

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

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

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

Conventions and Terminology

This topic describes important style conventions and terminology used in this document.

How the Term “internet” Is Used in This Document

In this document, 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.

How Numbers Are Used in This Document

In this document, 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.

Syntax, Message, and Response Conventions

The following topics provide information on the conventions used in syntax diagrams and in examples of messages and responses.

How to Read Syntax Diagrams

Special diagrams (often called *railroad tracks*) are used to show the syntax of external interfaces.

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

Within the syntax diagram, items on the line are required, items below the line are optional, and items above the line are defaults. See the examples in [Table 1 on page xviii](#).

Table 1. Examples of Syntax Diagram Conventions

Syntax Diagram Convention	Example
Keywords and Constants <p>A keyword or constant appears in uppercase letters. In this example, you must specify the item KEYWORD as shown.</p> <p>In most cases, you can specify a keyword or constant in uppercase letters, lowercase letters, or any combination. However, some applications may have additional conventions for using all-uppercase or all-lowercase.</p>	<p>» KEYWORD «</p>
Abbreviations <p>Uppercase letters denote the shortest acceptable abbreviation of an item, and lowercase letters denote the part that can be omitted. If an item appears entirely in uppercase letters, it cannot be abbreviated.</p> <p>In this example, you can specify KEYWO, KEYWOR, or KEYWORD.</p>	<p>» KEYWOrd «</p>
Symbols <p>You must specify these symbols exactly as they appear in the syntax diagram.</p>	<p>* Asterisk</p> <p>:</p> <p>Colon</p> <p>,</p> <p>Comma</p> <p>=</p> <p>Equal Sign</p> <p>-</p> <p>Hyphen</p> <p>()</p> <p>Parentheses</p> <p>.</p> <p>Period</p>
Variables <p>A variable appears in highlighted lowercase, usually italics.</p> <p>In this example, <i>var_name</i> represents a variable that you must specify following KEYWORD.</p>	<p>» KEYWOrd — <i>var_name</i> «</p>

Table 1. Examples of Syntax Diagram Conventions (continued)

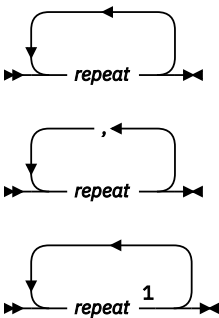
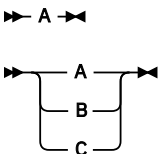
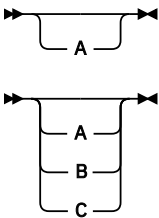
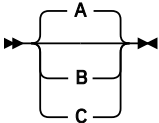
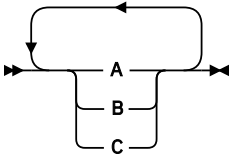
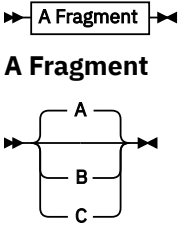
Syntax Diagram Convention	Example
<p>Repetitions</p> <p>An arrow returning to the left means that the item can be repeated.</p> <p>A character within the arrow means that you must separate each repetition of the item with that character.</p> <p>A number (1) by the arrow references a syntax note at the bottom of the diagram. The syntax note tells you how many times the item can be repeated.</p> <p>Syntax notes may also be used to explain other special aspects of the syntax.</p>	 <p>Notes:</p> <p>¹ Specify <i>repeat</i> up to 5 times.</p>
<p>Required Item or Choice</p> <p>When an item is on the line, it is required. In this example, you must specify A.</p> <p>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.</p>	
<p>Optional Item or Choice</p> <p>When an item is below the line, it is optional. In this example, you can choose A or nothing at all.</p> <p>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.</p>	
<p>Defaults</p> <p>When an item is above the line, it is the default. The system will use the default unless you override it. You can override the default by specifying an option from the stack below the line.</p> <p>In this example, A is the default. You can override A by choosing B or C.</p>	
<p>Repeatable Choice</p> <p>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.</p> <p>In this example, you can choose any combination of A, B, or C.</p>	

Table 1. Examples of Syntax Diagram Conventions (continued)	
Syntax Diagram Convention	Example
Syntax Fragment 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."	

Examples of Messages and Responses

Although most examples of messages and responses are shown exactly as they would appear, some content might depend on the specific situation. The following notation is used to show variable, optional, or alternative content:

- xxx**
Highlighted text (usually italics) indicates a variable that represents the data that will be displayed.
- []**
Brackets enclose optional text that might be displayed.
- { }**
Braces enclose alternative versions of text, one of which will be displayed.
- |**
The vertical bar separates items within brackets or braces.
- ...**
The ellipsis indicates that the preceding item might be repeated. A vertical ellipsis indicates that the preceding line, or a variation of that line, might be repeated.

Where to Find More Information

For more information about related publications, see [“Bibliography” on page 217](#).

Table 2 on [page xx](#) shows where to find specific information about TCP/IP for z/VM applications, functions, and protocols.

Table 2. Usage of TCP/IP for z/VM Applications, Functions, and Protocols

Applications, Functions, and Protocols	Topic	Document
eXternal Data Representation (XDR)	Usage	z/VM: TCP/IP Programmer's Reference
File Transfer Protocol (FTP)	Setting Up the Server	z/VM: TCP/IP Planning and Customization TCP/IP for z/VM Program Directory
	Usage	z/VM: TCP/IP User's Guide
	Commands	z/VM: TCP/IP User's Guide
MPROUTE	Setting Up the Server	z/VM: TCP/IP Planning and Customization
NETSTAT	Usage	z/VM: TCP/IP User's Guide

Table 2. Usage of TCP/IP for z/VM Applications, Functions, and Protocols (continued)

Applications, Functions, and Protocols	Topic	Document
Network File System (NFS)	Setting Up the Server	z/VM: TCP/IP Planning and Customization <i>TCP/IP for z/VM Program Directory</i>
	Usage	z/VM: TCP/IP User's Guide
OSF/Motif	Usage	z/VM: TCP/IP Programmer's Reference
PING	Usage	z/VM: TCP/IP Planning and Customization <i>TCP/IP for z/VM Program Directory</i> <i>z/VM: TCP/IP User's Guide</i>
Portmapper	Setting Up the Server	z/VM: TCP/IP Planning and Customization <i>TCP/IP for z/VM Program Directory</i>
	Usage	z/VM: TCP/IP Programmer's Reference z/VM: TCP/IP User's Guide
Remote Execution Protocol (REXEC)	Setting Up the Server	z/VM: TCP/IP Planning and Customization <i>TCP/IP for z/VM Program Directory</i>
	Usage	z/VM: TCP/IP User's Guide
Remote Printing	Setting Up the Server	z/VM: TCP/IP Planning and Customization <i>TCP/IP for z/VM Program Directory</i>
	Usage	z/VM: TCP/IP User's Guide
Remote Procedure Calls (RPC)	Usage	z/VM: TCP/IP Programmer's Reference
Resolver	CMS Program Interface	z/VM: TCP/IP Programmer's Reference
	Configuration Parameters	z/VM: TCP/IP Planning and Customization <i>TCP/IP for z/VM Program Directory</i>
RPCGEN command	Usage	z/VM: TCP/IP Programmer's Reference

Table 2. Usage of TCP/IP for z/VM Applications, Functions, and Protocols (continued)

Applications, Functions, and Protocols	Topic	Document
Simple Mail Transfer Protocol (SMTP)	Setting Up the Server	z/VM: TCP/IP Planning and Customization <i>TCP/IP for z/VM Program Directory</i>
	Usage	z/VM: TCP/IP User's Guide
	Interface to SMTP	z/VM: TCP/IP Programmer's Reference
Simple Network Management Protocol (SNMP)	Setting Up the Server and Agent	z/VM: TCP/IP Planning and Customization <i>TCP/IP for z/VM Program Directory</i>
	Usage	z/VM: TCP/IP Planning and Customization <i>TCP/IP for z/VM Program Directory</i> z/VM: TCP/IP User's Guide
SNMP Distributed Program Interface (DPI)	Usage	z/VM: TCP/IP Programmer's Reference
Socket Calls	Usage	z/VM: TCP/IP Programmer's Reference
Secure Socket Layer (SSL)	Setting Up the Server	z/VM: TCP/IP Planning and Customization
Telnet	Setting Up the Server	z/VM: TCP/IP Planning and Customization <i>TCP/IP for z/VM Program Directory</i>
	Usage	z/VM: TCP/IP User's Guide
	Commands	z/VM: TCP/IP User's Guide

Links to Other Online Documents

The online version of this document contains links to other online documents. These links are to editions that were current when this document was published. However, due to the nature of some links, if a new edition of a linked document has been published since the publication of this document, the linked document might not be the latest edition. Also, a link from this document to another document works only when both documents are in the same directory.

Links to Other Documents and Websites

The PDF version of this document contains links to other documents and websites. A link from this document to another document works only when both documents are in the same directory or database, and a link to a website works only if you have access to the Internet. A document link is to a specific edition. If a new edition of a linked document has been published since the publication of this document, the linked document might not be the latest edition.

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. See [How to send feedback to IBM](#) for additional information.

Summary of Changes for z/VM: TCP/IP Diagnosis Guide

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line (|) to the left of the change.

GC24-6328-73, z/VM 7.3 (June 2025)

This edition includes changes to support product changes provided or announced after the general availability of z/VM 7.3.

[7.3 PH65377] z/VM TCP/IP support for EQDIO

With the PTF for APAR PH65377, z/VM 7.3 provides a native network device driver for the z/VM TCP/IP stack that uses EQDIO adapters for network transport.

For more information, see:

- [*z/VM: TCP/IP Planning and Customization*](#)
- [*z/VM: TCP/IP User's Guide*](#)
- [*z/VM: TCP/IP Messages and Codes*](#)
- [*z/VM: Installation Guide*](#)
- [*z/VM: Performance*](#)

GC24-6328-73, z/VM 7.3 (September 2023)

This edition supports product changes that were provided or announced after the general availability of z/VM 7.3.

[PH56199, VM66698] System SSL z/OS 2.5 equivalence

With the PTFs for APARs PH56199 (TCP/IP) and VM66698 (LE), z/VM 7.3 provides an update to the cryptographic services library, which includes certificate diagnostic enhancements and improved algorithmic support and allows for enablement of TLS 1.3, for secure connectivity to the z/VM platform.

GC24-6328-73, z/VM 7.3 (September 2022)

This edition supports the general availability of z/VM 7.3. Note that the publication number suffix (-73) indicates the z/VM release to which this edition applies.

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

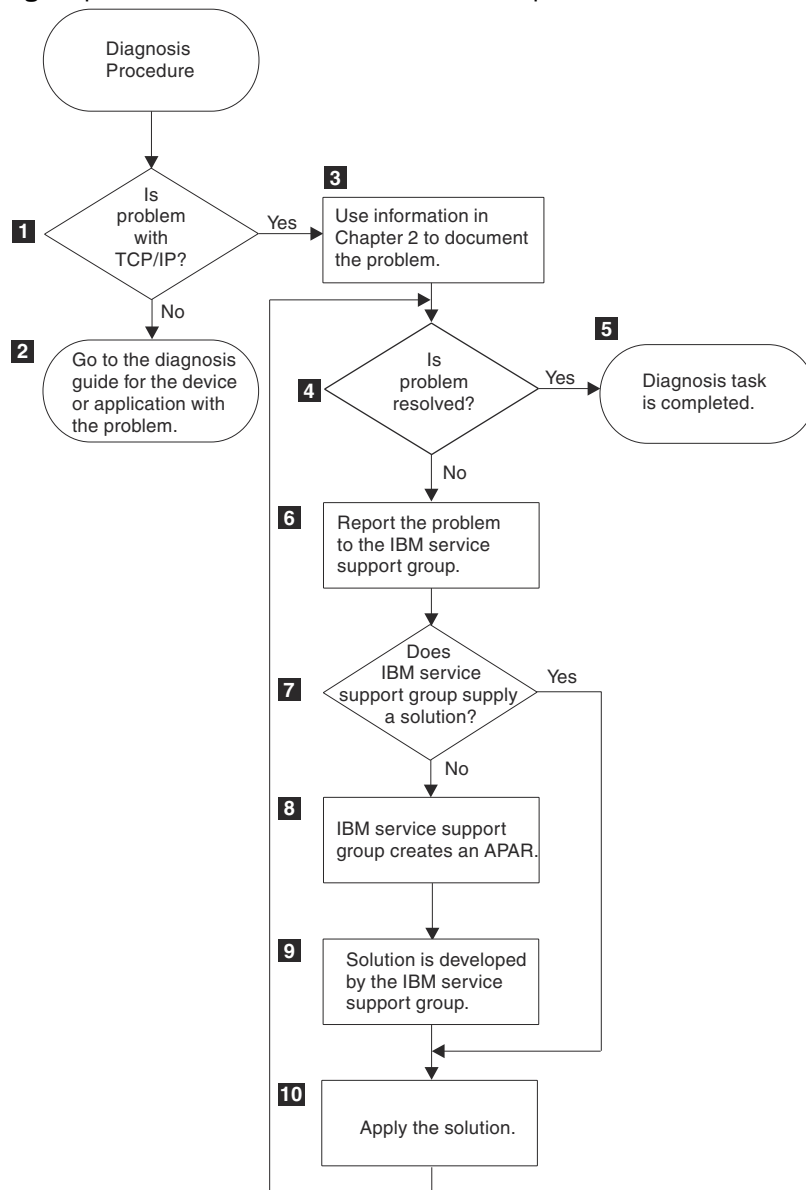


Figure 1. Overview of the Diagnosis Procedure

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

Various messages outputted 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,”](#) on [page 3](#), 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 [TCP/IP for z/VM](https://www.ibm.com/vm/related/tcpip) (<https://www.ibm.com/vm/related/tcpip>).

Note: Preventive service planning (PSP) buckets for many IBM products, including z/VM, are no longer created or updated. For more information, see:

- [z/VM Preventive Service](https://www.vm.ibm.com/service/prevserv.html#heading-link2) (<https://www.vm.ibm.com/service/prevserv.html#heading-link2>)
- [IBM servers supported by z/VM](#)

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 its 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 z/VM home page at TCP/IP for z/VM (<https://www.ibm.com/vm/related/tcpip>).

Note: Preventive service planning (PSP) buckets for many IBM products, including z/VM, are no longer created or updated. For more information, see:

- [z/VM Preventive Service \(https://www.vm.ibm.com/service/prevserv.html#heading-link2\)](https://www.vm.ibm.com/service/prevserv.html#heading-link2)
- [IBM servers supported by z/VM](#)

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 4](#). 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 for Machine Readable Documentation” on page 9](#))
- Client or server dump, if applicable.

You might also need to gather the following documentation:

- TCP/IP initial configuration file (PROFILE TCPIP, or its equivalent)
- Dynamic configuration (OBEYFILE) files
- Console listing
- TCPIP DATA file
- Channel control word (CCW) trace with data
- TCP/IP trace
- Customized DTCPARMS file
- RSU Service Level
- For out-of-storage abends, the size of the virtual machine and the output from the Query Segment command

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 4](#).

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 for Machine Readable Documentation” on page 9](#))
- Branch Trace if appropriate.

You might also need to gather the following documentation:

- TCP/IP initial configuration file (PROFILE TCPIP, or its equivalent)
- Dynamic configuration (OBEYFILE) files
- TCPIP DATA 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 for Machine Readable Documentation”](#) on page 9)
- Dump of the client or server virtual machine if appropriate.

You might also need to gather the following documentation:

- TCP/IP initial configuration file (PROFILE TCPIP, or its equivalent)
- Dynamic configuration (OBEYFILE) files
- TCPIP DATA 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 for Machine Readable Documentation”](#) on page 9)
- 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 for Machine Readable Documentation”](#) on page 9)
- 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 involved.
 - Any modifications made to your operating system, input/output (I/O) generation, or the connection interface, such as 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 z/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 VMDUMP 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.
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 3. 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 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 3](#) 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,” on page 3](#), 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 4](#).

Message

The message problem category describes a problem identified by a message. These problems are explained in [“Message” on page 4](#).

Loop

Loop problems refer to an operation that repeats endlessly. These problems are explained in [“Loop” on page 5](#).

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 6](#).

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 6](#).

Performance

A performance problem is characterized by slow response time or slow throughput. These problems are explained in [“Performance” on page 7](#).

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 8](#).

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 10](#), 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 on page 13 represents the TCP/IP layered architecture for the VM environment.

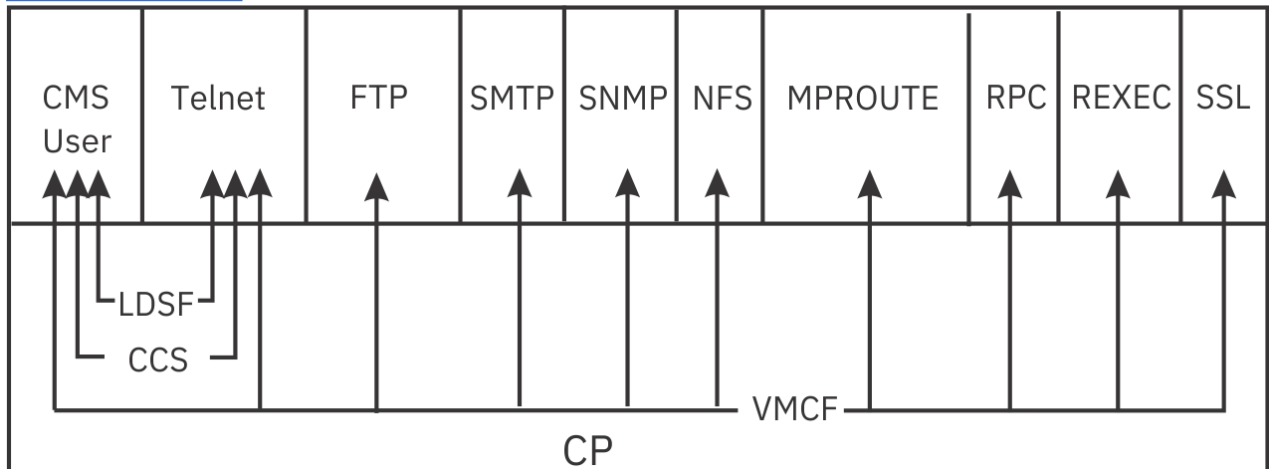


Figure 3. The TCP/IP Layered Architecture for VM

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 on page 14](#).

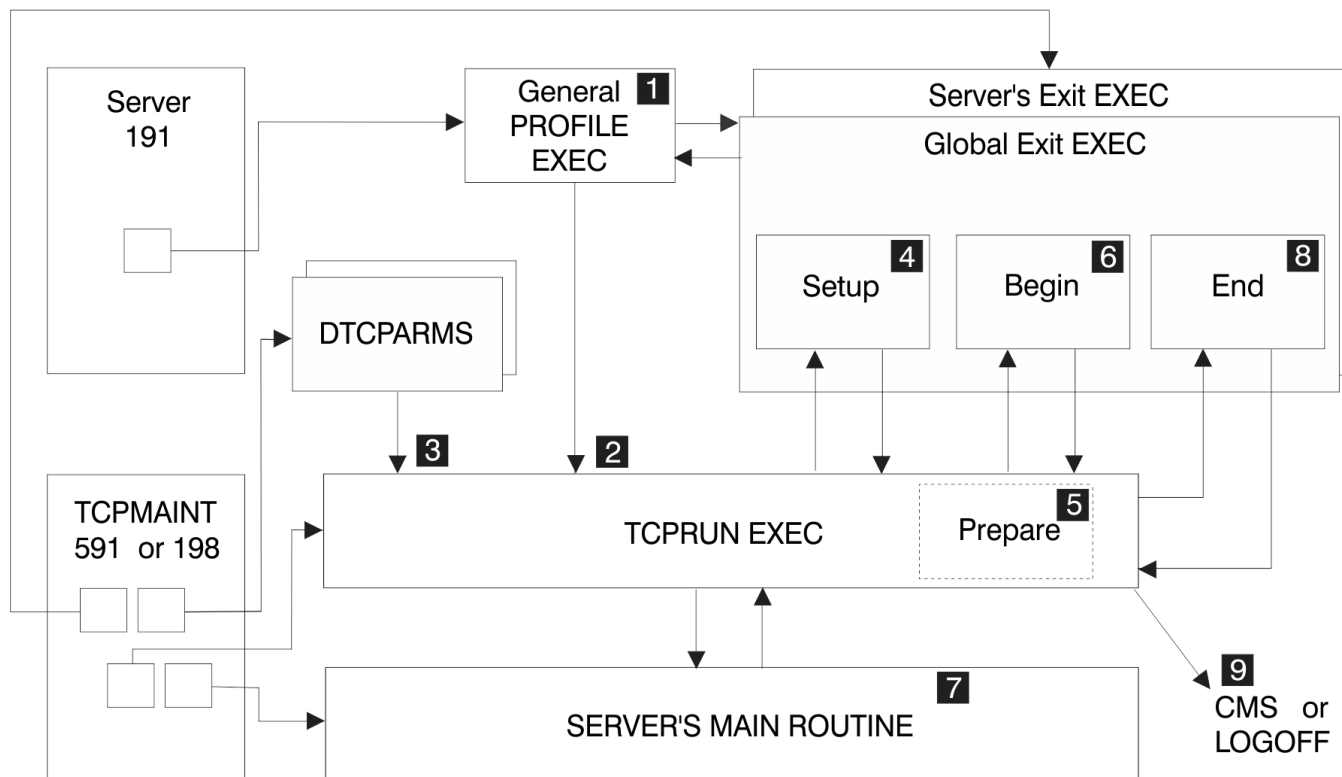


Figure 4. The sequence of a Server Startup

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 TCP/IP 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 on page 16](#) shows a simple internet with a gateway.

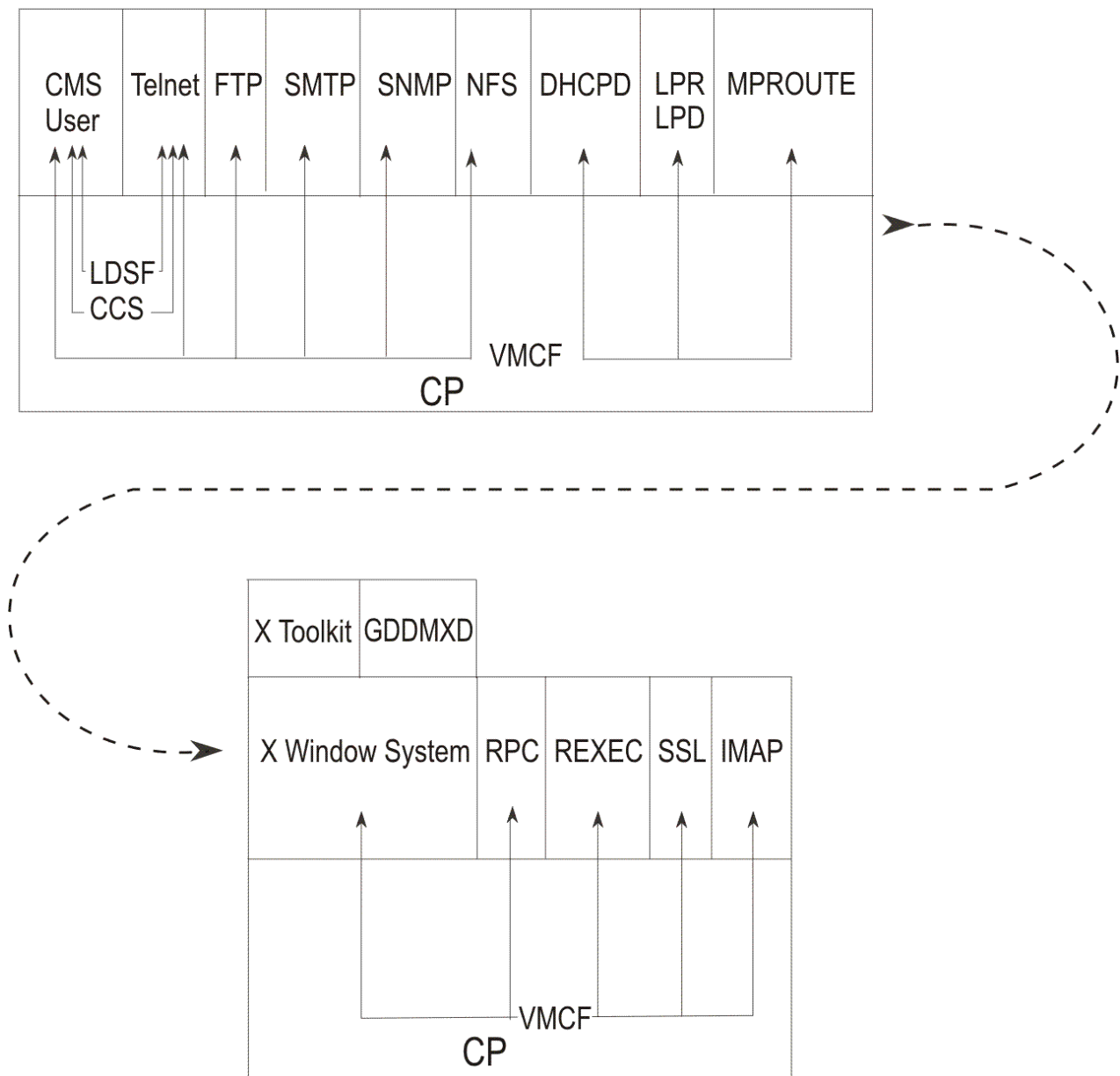


Figure 5. Networks with a Gateway Forming an Internet

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 on page 17 depicts a router and a bridge. The router connects Network 1 to Network 2 to form an internet.

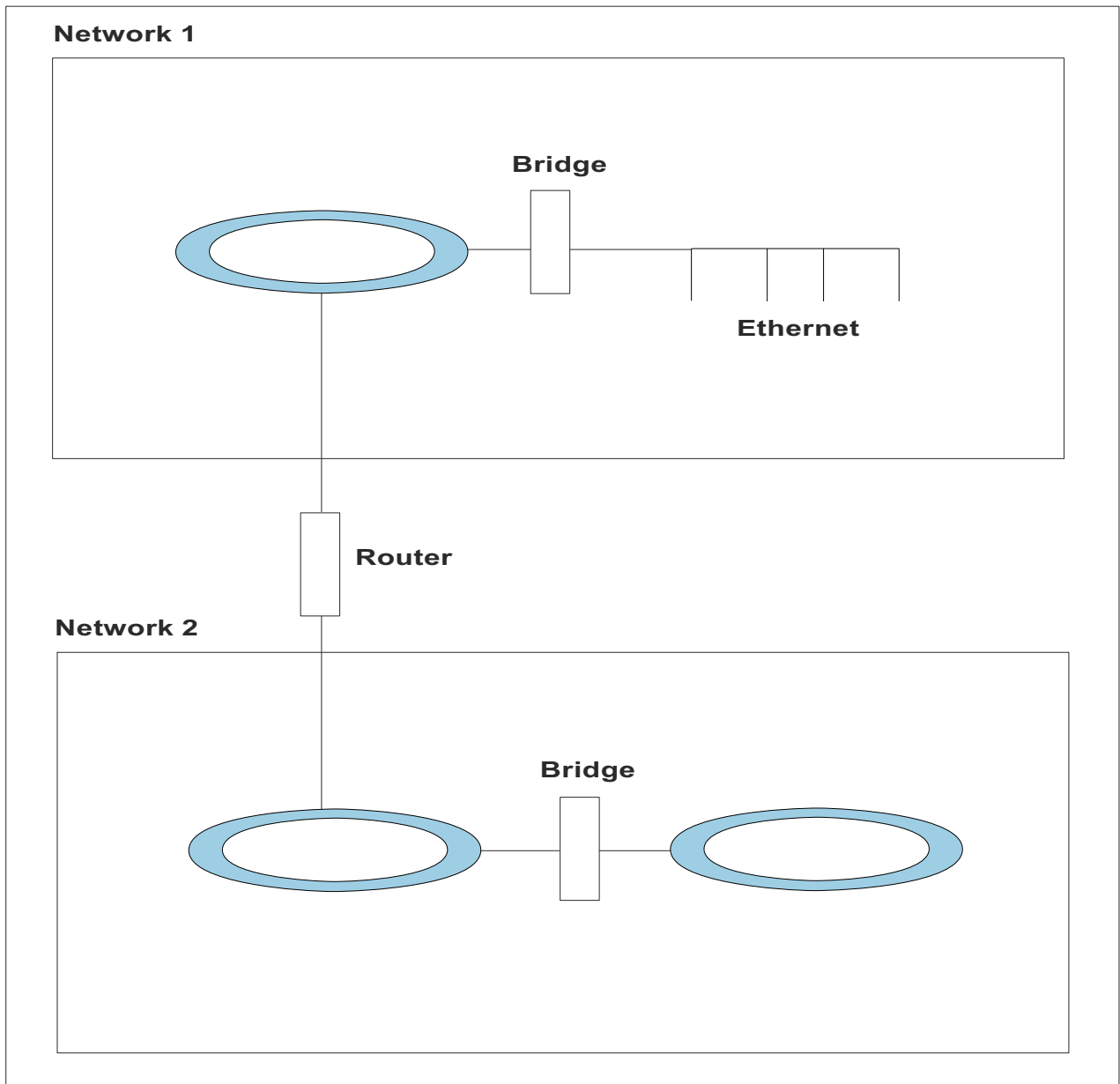
Internet A

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 on page 18](#) shows the relationship of MTU to frame size.

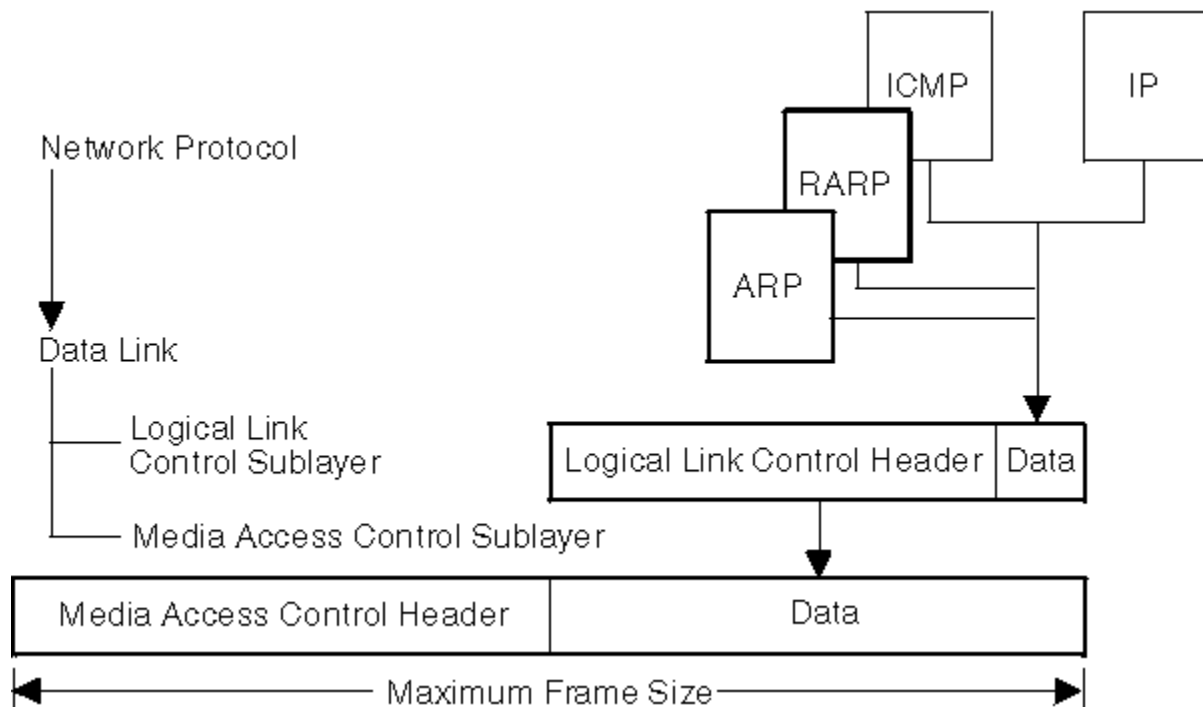


Figure 7. Relationship of MTU to Frame Size

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.

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 19](#) 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 19](#).

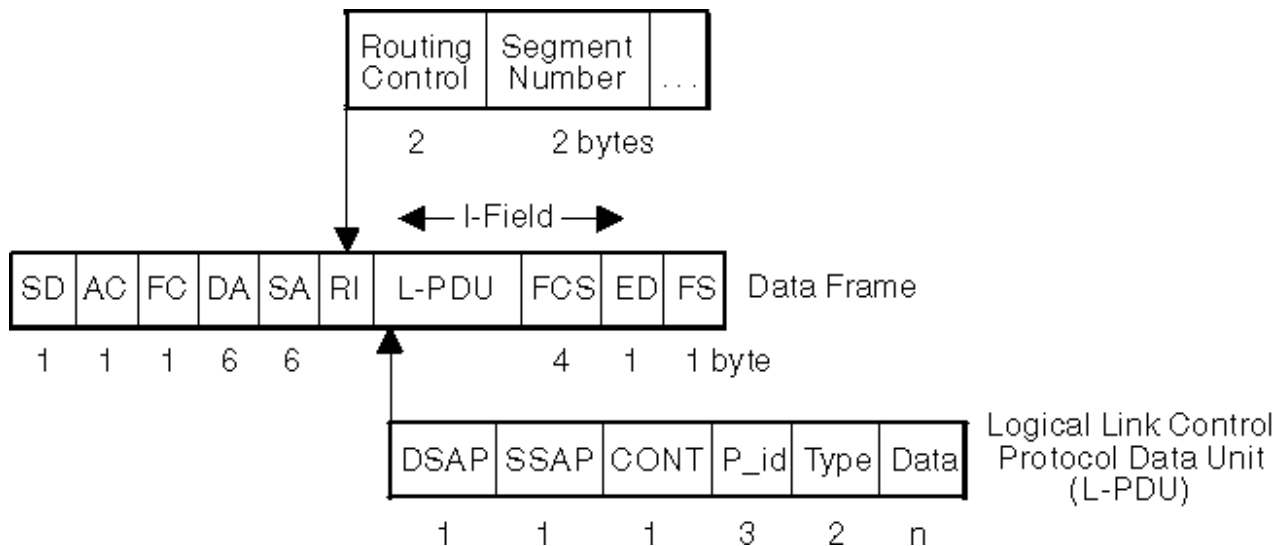


Figure 8. Format of an IEEE 802.5 Token-Ring Frame

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 4 on page 19 shows the relationship between the RC field and the maximum I-field values.

Table 4. 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	11 407
x110 xxxx xxxx xxxx	17 800

In Figure 8 on page 19, 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 than the maximum I-field. (Note that the L-PDU contains a SNAP header, as described in “Sub-Network Access Protocol (SNAP)” on page 20) 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 on page 20.

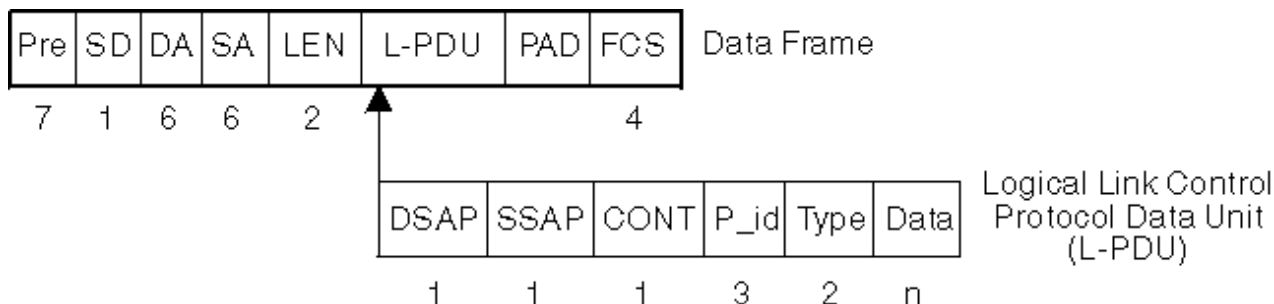


Figure 9. Format of an IEEE 802.3 Frame

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 on page 20](#).

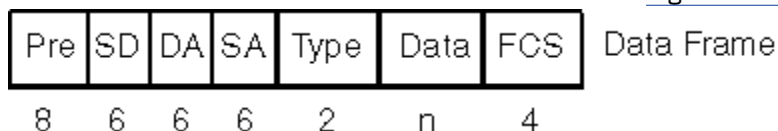


Figure 10. Format of an Ethernet V2 Frame

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 on page 20](#).
LLC with SNAP Header

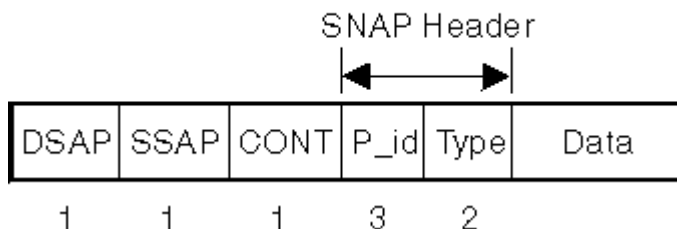


Figure 11. SNAP Header

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

Internet Addressing

Hosts on an internet are identified by their *IP address*. *Internet Protocol* (IPv4 and IPv6) 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 (IPv4) and RFC 2460 (IPv6).

IPv4 Addressing

An IPv4 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 IPv4 address that is used for each of these addresses is determined by the *class of address*. There are four commonly used classes of IPv4 address (see [Figure 12 on page 21](#)).

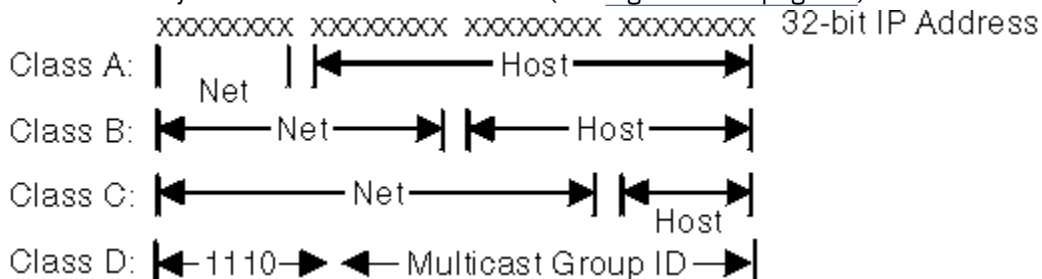


Figure 12. Classes of IP Addresses

The class of address of the IPv4 network is determined from the first 4 bits in the first octet of the IP address. [Figure 13 on page 22](#) shows how the class of address is determined.

32-bit address		xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx
Class A		0xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx
	min	00000000			
	max	01111111			
	range	0 - 127	(decimal notation)		
Class B		10xxxxxx	xxxxxxx	xxxxxxx	xxxxxxx
	min	10000000			
	max	10111111			
	range	128 - 191	(decimal notation)		
Class C		110xxxxx	xxxxxxx	xxxxxxx	xxxxxxx
	min	11000000			
	max	11011111			
	range	192 - 223	(decimal notation)		
Class D		1110xxxx	xxxxxxx	xxxxxxx	xxxxxxx
	min	11100000			
	max	11101111			
	range	224 - 239	255.255.255		

Figure 13. Determining the Class of an IP Address

As shown in [Figure 13 on page 22](#), 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 '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*.

IPv6 Addressing

One problem that IPv6 solves is the limited number of addresses available in IPv4. IPv6 uses a 128-bit address space, which has no practical limit on global addressability and provides 340 282 366 920 938 463 463 374 607 431 768 211 456 addresses. Currently, this is enough addresses so that every person can have a single IPv6 network with as many as 18 000 000 000 000 000 000 nodes on it, and still the address space would be almost completely unused.

There are three conventional forms for representing IPv6 addresses as text strings:

- The preferred form is x:x:x:x:x:x:x, where the x's are the hexadecimal values of the eight 16-bit pieces of the address.

```
FE80:0000:0000:0000:0001:0800:23e7:f5db
1080:0:0:0:8:800:200C:417A
```

It is not necessary to write the leading zeros in an individual field, but there must be at least one numeral in every field (except for the case described in the following bullet).

- Due to some methods of allocating certain styles of IPv6 addresses, it will be common for addresses to contain long strings of zero bits. In order to make writing addresses containing zero bits easier, a special syntax is available to compress the zeros. The use of :: indicates multiple groups of 16 bits of zeros. The :: can only appear once in an address. The :: can also be used to compress both leading and trailing zeros in an address.

The following is a preferred form address:

```
1080:0:0:0:8:800:200C:417A
FF01:0:0:0:0:0:0:101
0:0:0:0:0:0:0:1
0:0:0:0:0:0:0:0
```

The corresponding compressed forms are:


```
1080::8:800:200C:417A
FF01::101
::1
::
```

- An alternative form that is sometimes more convenient when dealing with a mixed environment of IPv4 and IPv6 nodes is x:x:x:x:x:d.d.d.d, where the x's are the hexadecimal values of the 6 high-order 16-bit pieces of the address, and the d's are the decimal values of the 4 low-order 8-bit pieces of the address (standard IPv4 representation). This form is used for IPv4-compatible IPv6 addresses and IPv4-mapped IPv6 addresses. These types of addresses are used to hold embedded IPv4 addresses in order to carry IPv6 packets over the IPv4 routing infrastructure.

```
0:0:0:0:0:0:13.1.68.3
0:0:0:0:0:FFFF:129.144.52.38
```

The same addresses in compressed form are:

```
::13.1.68.3
::FFFF:129.144.52.38
```

As important as the expanded address space is the use of hierarchical address formats. The IPv4 addressing hierarchy includes network and host components in an IPv4 address. IPv6, with its 128-bit addresses, provides globally unique and hierarchical addressing based on prefixes rather than address classes, which keeps routing tables small and backbone routing efficient.

The general format is as follows:

Table 5. IPv6 Address Format		
global routing prefix	subnet ID	interface ID
n bits	m bits	128-(n+m) bits

The global routing prefix is a value (typically hierarchically structured) assigned to a site; the subnet ID is an identifier of a link within the site; and the interface ID is a unique identifier for a network device on a given link (usually automatically assigned).

For more information on IPv6 addresses, prefixes and routing refer to the [z/VM: TCP/IP User's Guide](#).

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.

[Figure 14 on page 24](#) shows a simple network with a bridge and a router.

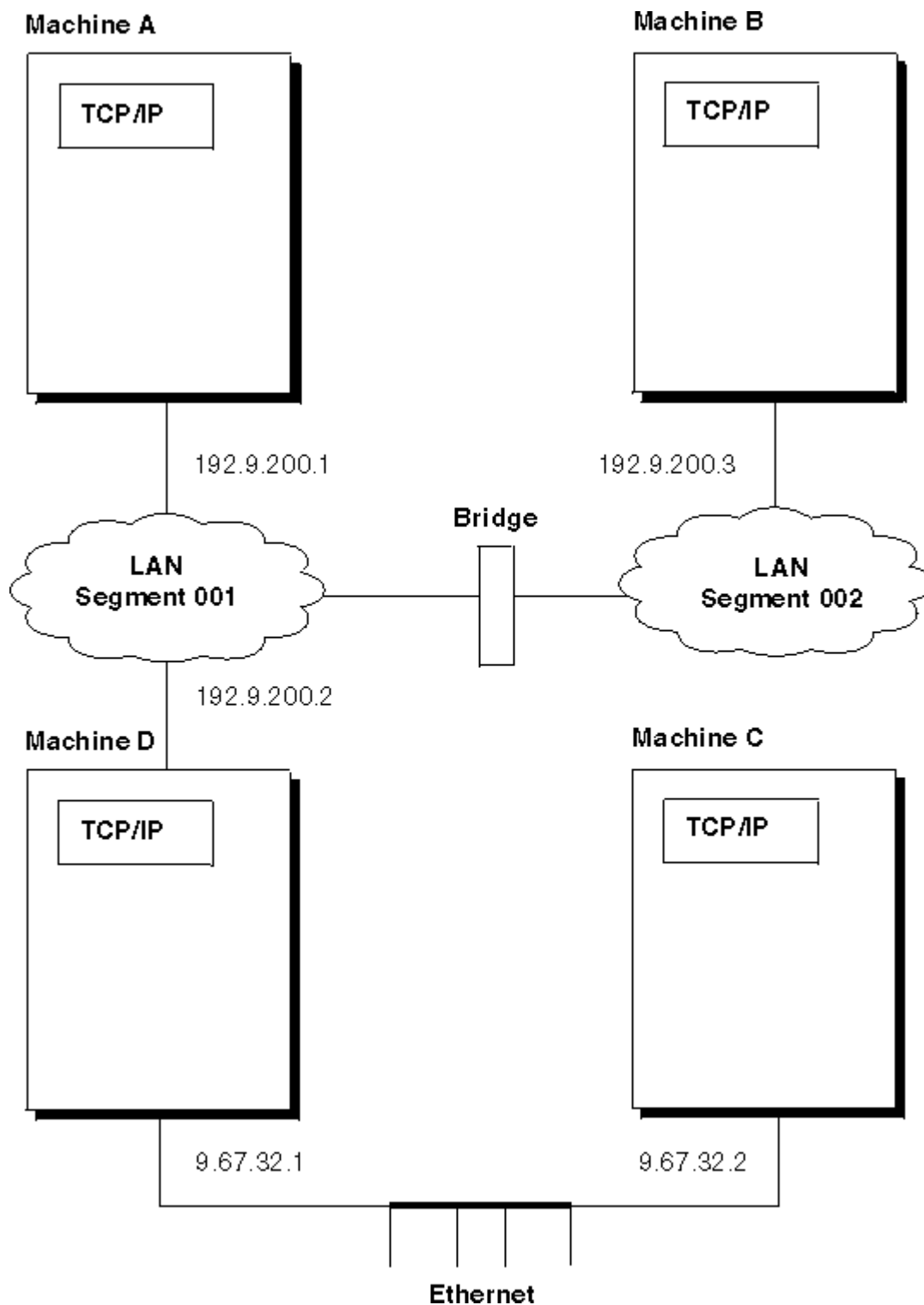


Figure 14. Routing and Bridging

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 on page 24](#), 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 24](#), 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 on page 25](#) is used.

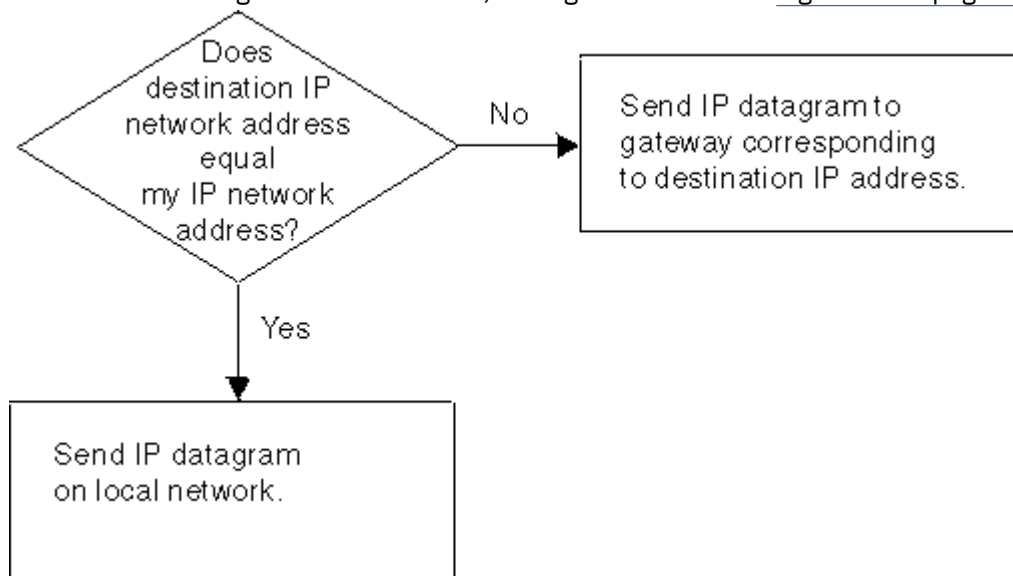


Figure 15. General IP Routing Algorithm

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 24](#).

```

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 Address	Host Address Range		
192	9	200	host				
11000000	00001001	11001000	xxxxxxxx	192.9.200	1 - 254		
With Subnetting:				Subnet Address	Host Address Range	Subnet Value	
192	9	200	64 host				
11000000	00001001	11001000	01xxxxxx	192.9.200.64	65 - 126	01	
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 on page 27 shows the routing algorithm used with subnets and Figure 17 on page 27 shows how a subnet route is resolved.

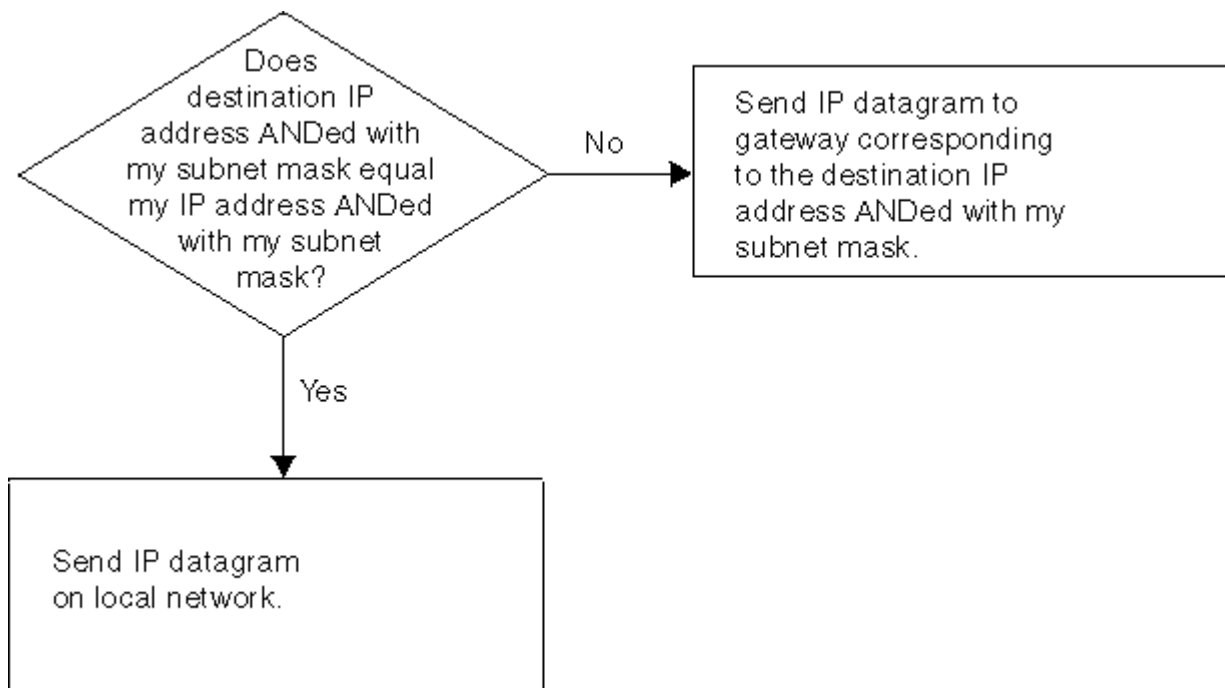


Figure 16. Routing Algorithm with Subnets

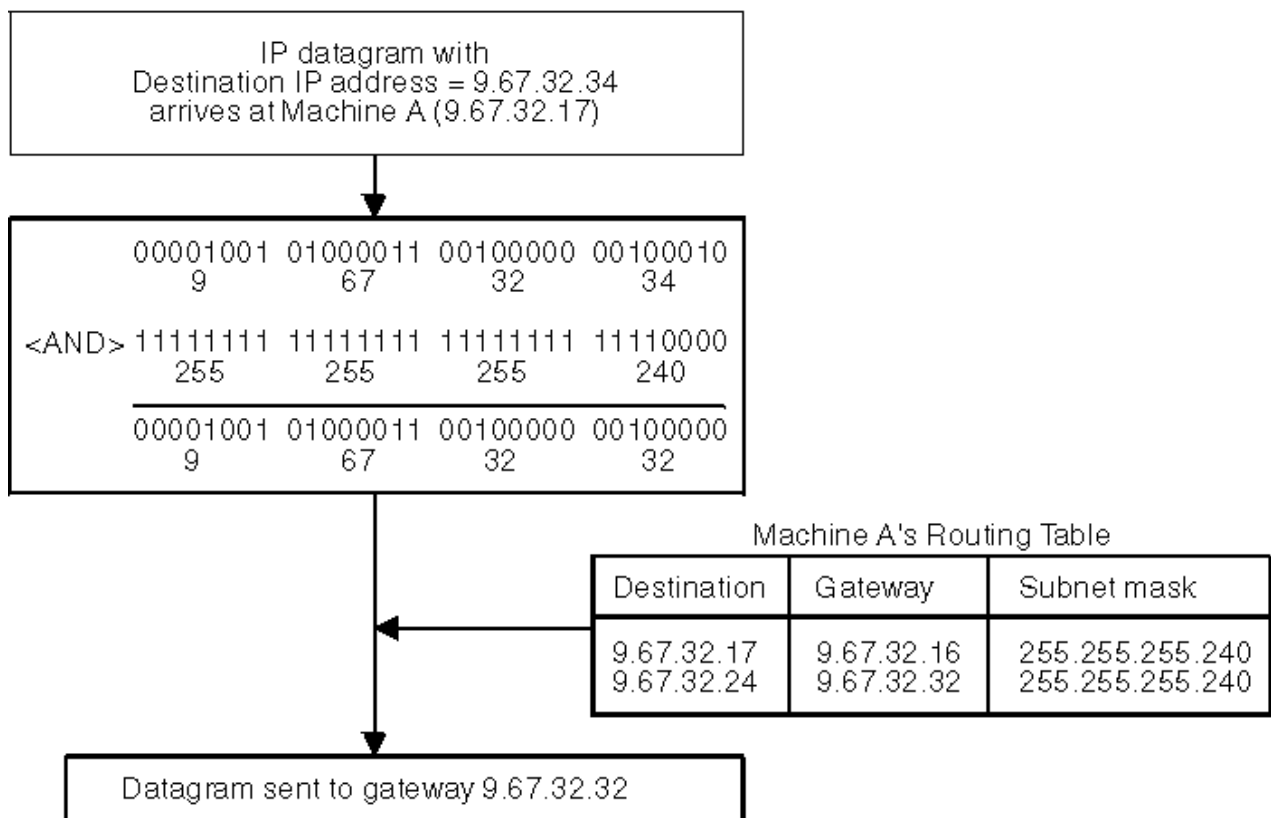


Figure 17. Example of Resolving a Subnet Route

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 [z/VM: 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 update dynamically the internal routing tables when changes to the network occur. TCP/IP uses the Open Shortest Path First (OSPF) and Routing Information Protocol (RIP) protocols and the MPROUTE virtual machine to monitor network changes. [z/VM: TCP/IP Planning and Customization](#) contains more details about MPRoute.

Dynamic Routing Tables

When TCP/IP is configured to use MPRoute, there are actually two routing tables: the MPRoute routing table, and the TCP/IP stack routing table. The MPRoute routing table is updated dynamically based on the RIP or OSPF protocol. The MPRoute server is responsible for managing the TCP/IP stack routing table (by sending routing updates to TCP/IP). The two routing tables will be similar, but **might not be identical**. The MPRoute server's job is limited to managing the TCP/IP stack routing table. MPRoute is not directly involved in the actual routing decisions made by the TCP/IP stack when routing a packet to its destination. When routing a packet, the TCP/IP stack will use the TCP/IP stack routing table to make its routing decisions.

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

Example of Network Connectivity

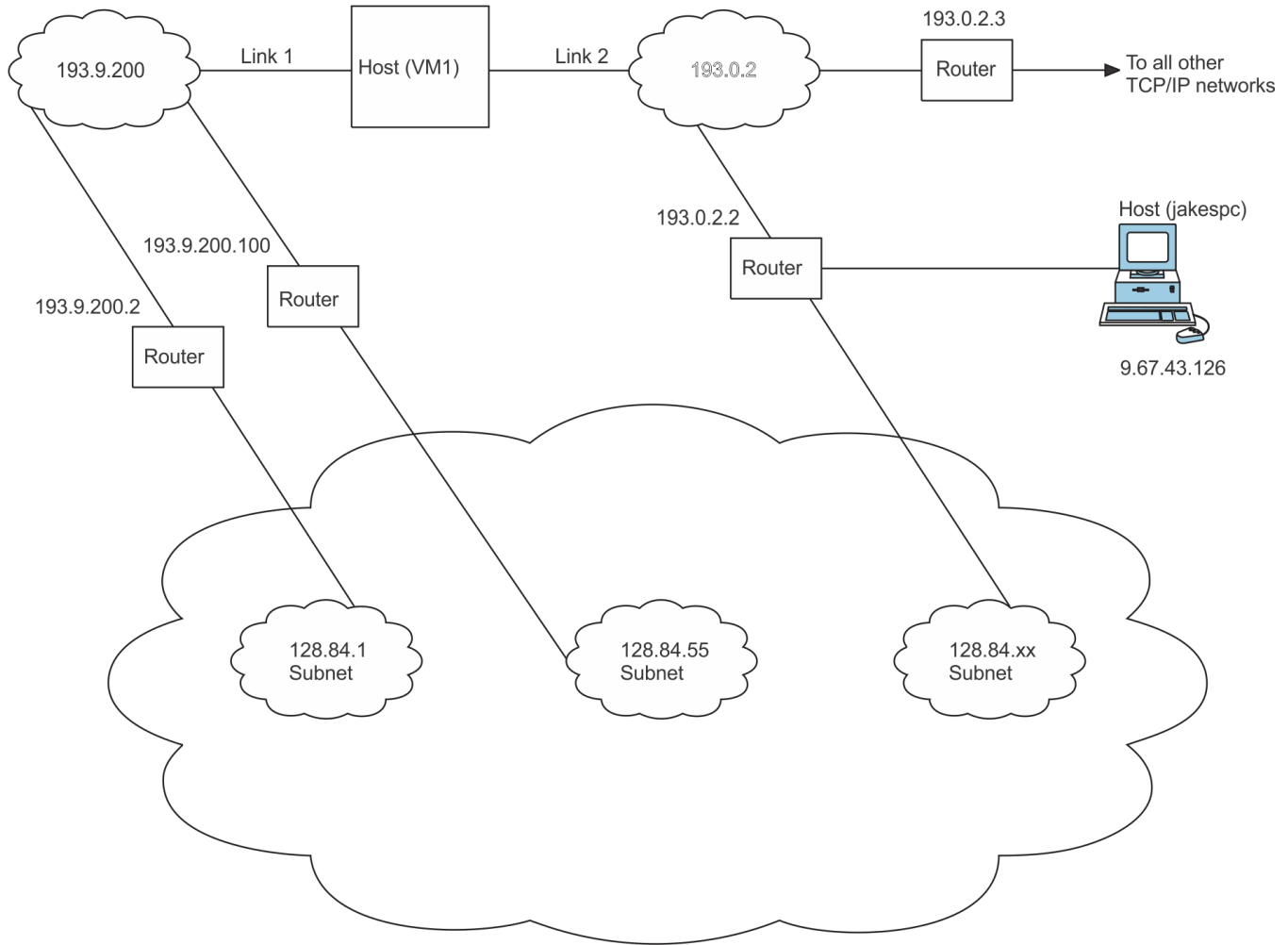


Figure 18. Example of Network Connectivity

Figure 18 on page 29 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.

Note:

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 *z/VM: 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 6 on page 33 describes the major internal Pascal procedures. These procedures are external declarations of processes invoked by the scheduler.

Table 6. 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 DEVICESdriverNAME.
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.
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.
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.

Table 6. TCP/IP Internal Procedures (continued)

Procedure	Description
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.
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.

Table 6. TCP/IP Internal Procedures (continued)

Procedure	Description
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.
ToIUCV	Sends the outgoing datagrams supplied by IpDown to PVM IUCV. See “IUCV Links” on page 39 for more information.
ToPCCA3	Sends the outgoing datagrams supplied by IpDown to PCCA3. PCCA is the name for LAN channel-attached units.
UdpRequest	Processes incoming UDP requests. Gives outgoing datagrams to IpDown.

Queues

Table 7 on page 35 describes the queues TCP/IP uses to control events that occur during run-time.

Table 7. TCPIP Queues

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

Table 7. TCPIP Queues (continued)

Queue	Description
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 8 on page 36 describes TCP/IP internal activities performed by TCP/IP processes. An example of called internal activities is shown in [Figure 48 on page 67](#). Activities, which are found in most TCP/IP internal traces, explain why the process has been called.

Table 8. 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 QueueOfCcbForTcpResources.

Table 8. TCP/IP Internal Activities (continued)

Activity	Description
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.
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.
HAVEcompletedIO	Sent by the I/O interrupt to a network driver when the most recent I/O operation has been completed.
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.

Table 8. TCP/IP Internal Activities (continued)

Activity	Description
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.
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 SNMPPDPI 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.

Table 8. TCP/IP Internal Activities (continued)

Activity	Description
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.
SENDwriteDIAG	Sent by any internal process before it tests the Series/1 write channel.
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.
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 (via timeout), 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 IUCV.

IUCV Links

At present, IUCV links support the IUCV communication: Passthrough Virtual Machine (PVM) IUCV.

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 19 on page 40](#)

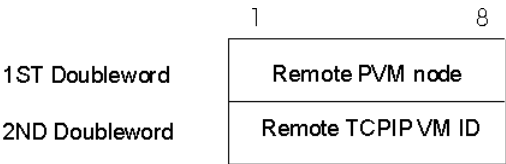


Figure 19. Format of the User Field for a CONNECT Request

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 20 on page 40](#)

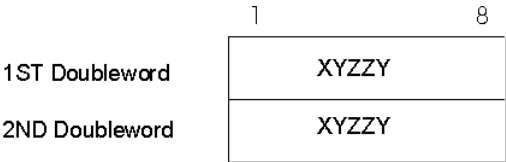


Figure 20. Format of the User Field for a Local IUCV CONNECT Request

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

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. 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 ETC HOSTS or HOSTS LOCAL file (if ETC HOSTS does not exist) 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 ETC HOSTS or HOSTS LOCAL file (if ETC HOSTS does not exist).
- **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 (MPRoute), verify that all routing daemons are operating.

- **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 [z/VM: 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.

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 [z/VM: 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:



The parameters of the TRACE command are:

Parameter

Description

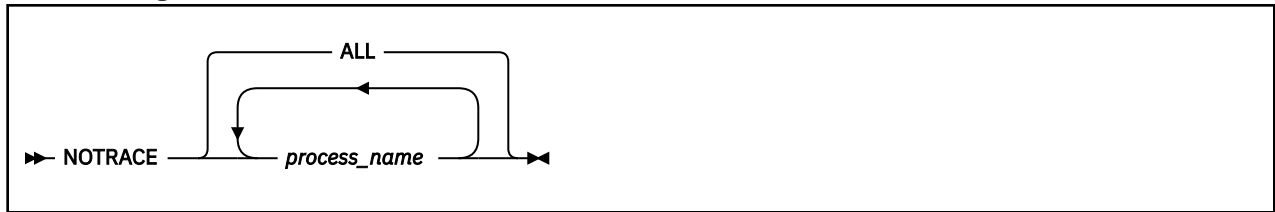
process_name

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:



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.

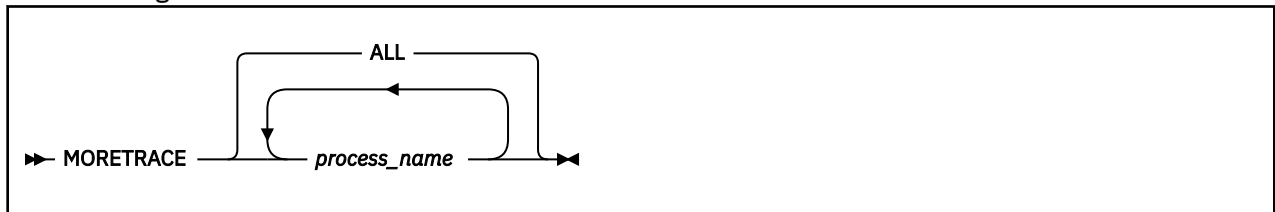
ALL

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:



The parameters of the MORETRACE command are:

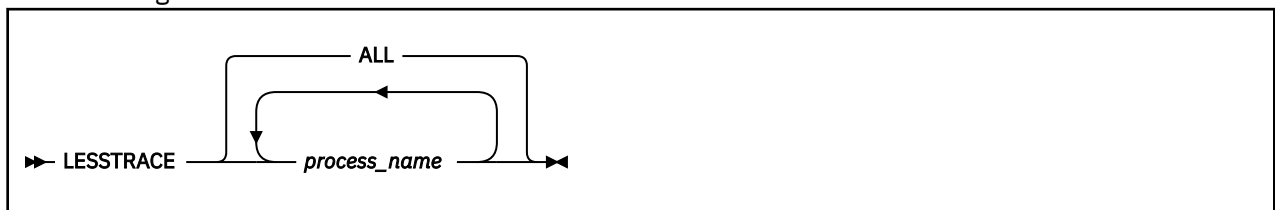
Parameter**Description*****process_name***

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:



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 44 on page 65 shows a sample trace using LESSTRACE.

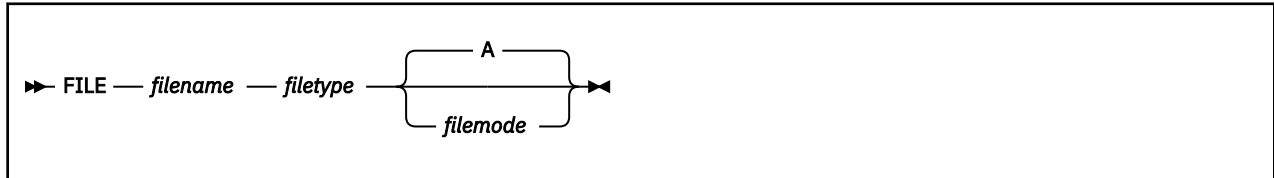
Directing Output

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 parameters of the FILE command are:

Parameter	Description
-----------	-------------

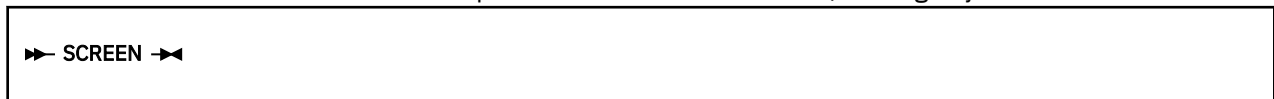
filename	The name of the file to which the output is written.
-----------------	--

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.



The SCREEN command has no parameters.

For more information about trace activation and output statements, see the [z/VM: TCP/IP Planning and Customization](#).

TCP/IP Packet Tracing

TCP/IP packet tracing provides a packet-by-packet view of inbound and outbound activity associated with a network interface. Two methods are available for collecting packet trace data on z/VM. The following sections describe each method with examples on their use:

- Native TCP/IP Stack Packet Trace
- TCP/IP Stack Packet Trace with TRSOURCE

Native TCP/IP Stack Packet Trace

This method of capturing packet data employs tracing facilities that are inherent to the TCP/IP server. Data is captured and stored in a designated trace file, or more commonly, the TCP/IP server console. Analysis of the collected data requires a thorough understanding of TCP/IP server processes, while the data itself is presented in a somewhat raw format. The collection of data in this manner, and its analysis is often performed in consultation with IBM Support personnel.

To collect a native TCP/IP stack packet trace, use the following steps:

1. To spool any initialization messages, first issue:

```
NETSTAT CP SPOOL CONS CLOSE
```

2. To start the packet trace issue:

```
NETSTAT OBEY SCREEN MORETRACE IPUP IPDOWN PACKET TRACEONLY a.b.c.d ENDTRACEONLY
```

- SCREEN specifies that trace data should be sent to the TCP/IP server console
- IPUP sets inbound IP tracing
- IPDOWN sets outbound IP tracing
- PACKET specifies the entire packet data should be included in the trace
- *a.b.c.d* is the IP address for which packet trace data should be collected

3. Recreate the problem to be traced

4. To stop tracing issue:

```
NETSTAT OBEY NOTRACE TRACEONLY ENDTRACEONLY
```

5. To spool the trace data issue:

```
NETSTAT CP SPOOL CONS CLOSE
```

6. The formatted packet trace data will be sent to the user specified on the :OWNER tag of the TCP/IP server DTCPARMS entry (for which the default is TCPMAINT)

TCP/IP Stack Packet Trace with TRSOURCE

This method of capturing packet data employs native CP tracing facilities. Data is captured and stored in a file for formatting and viewing by the IPFORMAT tool. The collection of data in this manner, and its analysis using IPFORMAT may prove to be an effective method of troubleshooting a variety of TCP/IP problems, especially for those familiar with network diagnostic tools on other platforms that provide formatted packet data for analytical use.

To collect a TCP/IP stack packet trace in concert with TRSOURCE tracing functions, use the steps that follow.

Note: A sample program (PKTTRACE SAMPEXEC) is provided with TCP/IP for z/VM that illustrates how the steps outlined below can be encapsulated to simplify the process of obtaining and processing a TRSOURCE-based packet trace. For more information, refer to the commentary and program instructions provided within the PKTTRACE SAMPEXEC file.

1. Issue the following commands to enable the trace:

```
NETSTAT OBEY PACKETTRACESIZE 64  
NETSTAT OBEY TRACEONLY ETH0 ENDTRACEONLY
```

- PACKETTRACESIZE 64 specifies that 64 bytes of inbound and outbound data will be collected for each packet
- TRACEONLY ETH0 ENDTRACEONLY specifies that packet trace data should be collected for device ETH0 (where ETH0 corresponds to the *device_name* on a DEVICE statement in the TCP/IP server configuration file)

2. Issue the following Control Program commands to start data collection:

```
TRSOURCE ID TCP TYPE GT BLOCK FOR USER tcip_userid  
TRSOURCE ENABLE ID TCP
```

- The first TRSOURCE command defines the trace desired (where *tcip_userid* is the user ID of the relevant TCP/IP server)

- The second TRSOURCE command enables the trace
3. Recreate the problem to be traced
 4. To stop tracing issue the following commands:

```
NETSTAT OBEY PACKETTRACESIZE 0
NETSTAT OBEY TRACEONLY ENDTRACEONLY
TRSOURCE DISABLE ID TCP
```

- The two NETSTAT commands disable the TCP/IP server packet trace
 - The TRSOURCE command disables the CP trace defined earlier
5. Use the following command to determine the spool file identifier for the TRF file:

```
QUERY TRFILES USERID tcip_userid
```

6. Use the following TRACERED command to format the trace data:

```
TRACERED nnnnn CMS TCP TRACE A ( ALL
```

where *nnnnn* is the spool file identifier returned from the previous QUERY command

7. Use IPFORMAT to view the formatted trace file (see [Chapter 8, “Using IPFORMAT Packet Trace Formatting Tool,”](#) on page 109).

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 33. 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.

The PING trace is valid only for those clients who use the VMCF PING interface. As of z/VM 5.1.0, the z/VM PING command no longer uses the VMCF interface. To trace the z/VM PING client, see the description of the DEBUG option on the PING command in the *z/VM: TCP/IP User's Guide*. A PING trace will show all inbound ICMP echo responses regardless of whether the initial request originated as a VMCF PING request. However, the trace only reflects the actions of the VMCF PING handler, so the trace may show a PING response being discarded while it is, in fact, being handled by a separate process.

ARP

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

Figure 21 on page 50 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
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 21. A Sample of an ARP Trace (Part 1 of 2)

```

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 22. A Sample of an ARP Trace (Part 2 of 2)

Figure 23 on page 51 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 9.67.58.234,
ScanTranslationVisitNode: NOT deleting entry for link TR2 address 9.67.58.193

```

Figure 23. A Sample of an ARP Trace Using MORETRACE

CCS

Figure 24 on page 52 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 24. A Sample of a CCS Trace

Congestion

Figure 25 on page 52 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. Thresh
esh now 1072, was 65535. MSS 536, SndWnd 0
TCPUTI032I Conn 1004: TcpSlowStart: CongestionWindow now 536, was 536. Thresh
h now 1072, was 1072. MSS 536, SndWnd 0
TCPUTI032I Conn 1004: TcpSlowStart: CongestionWindow now 536, was 536. Thresh
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
TCPUTI019I Bytes: 1 sent, 0 acked, 0 received
TCPUTI027I Max use: 1 in retransmit Q
TCPROU003I Conn 1004: Opening congestion win: CongestionWindow 65535, Thresho
ld 65535, MSS 536, increment 536
TCPUTI032I Conn 1004: TcpSlowStart: CongestionWindow now 536, was 65535. Thresh
esh now 4096, was 65535. MSS 536, SndWnd 8192
TCPROU003I Conn 1004: Opening congestion win: CongestionWindow 536, Thresho
ld 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 25. 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 26 on page 53 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
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 0
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
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 0
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 26. A Sample of a CONSISTENCYCHECKER Trace

DENIALOFSERVICE

The DENIALOFSERVICE trace provides data about the type of Denial-of-Service attack detected. The DENIALOFSERVICE trace supports two levels of tracing, TRACE and MORETRACE.

Note: Unlike TRACE DENIALOFSERVICE, which prints only one message for the first incoming TCP packet per IP address, MORETRACE DENIALOFSERVICE prints out a message for every packet in the attack. As attacks come in bulk, this could affect stack performance.

Figure 27 on page 54 shows a sample of a DENIALOFSERVICE trace. DENIALOFSERVICE was specified in the TRACE statement in the PROFILE TCPIP file.

```
08:55:15 DTCIPU087I A Smurf-OB denial-of-service attack has
        been detected from address 10.130.248.99

09:05:54 DTCIPU087I A POD denial-of-service attack has
        been detected from address 10.130.58.22

10:08:08 DTCIPU087I A Land denial-of-service attack has
        been detected from address 10.130.58.42

11:11:34 DTCIPU087I A Synflood denial-of-service attack has
        been detected from address 10.130.249.43
```

Figure 27. A Sample of a DENIALOFSERVICE in the TRACE Statement

Figure 28 on page 54 shows samples of DENIALOFSERVICE traces using MORETRACE. DENIALOFSERVICE was specified in the MORETRACE statement in the PROFILE TCPIP file.

```
12:49:55 DTCIPU087I A Smurf-OB denial-of-service attack has
        been detected from IP address 10.130.201.99
12:50:05 DTCIPU087I A Smurf-OB denial-of-service attack has
        been detected from IP address 10.130.201.99
12:50:15 DTCIPU087I A Smurf-OB denial-of-service attack has
        been detected from IP address 10.130.201.99
12:50:25 DTCIPU087I A Smurf-OB denial-of-service attack has
        been detected from IP address 10.130.201.99
12:50:35 DTCIPU087I A Smurf-OB denial-of-service attack has
        been detected from IP address 10.130.201.99
12:50:45 DTCIPU087I A Smurf-OB denial-of-service attack has
        been detected from IP address 10.130.201.99
12:50:55 DTCIPU087I A Smurf-OB denial-of-service attack has
        been detected from IP address 10.130.201.99

12:52:03 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
12:52:03 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
12:52:03 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
12:52:03 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
12:52:03 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
12:52:03 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
12:52:03 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
12:52:03 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
12:52:03 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
12:52:03 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
12:52:15 DTCIPU087I A Land denial-of-service attack has been
        detected from IP address 10.130.201.99
```

Figure 28. A Sample of a DENIALOFSERVICE in the MORETRACE Statement

DROPPED

The DROPPED trace provides information when the z/VM TCP/IP stack drops packets.

Figure 29 on page 55 shows a sample of a DROPPED trace. DROPPED was specified in the TRACE statement in the PROFILE TCPIP file.


```
11:11:15 DTCIPU050I IgmpHandle: dropping loopback IGMP datagram
11:11:17 DTCIPU050I IgmpHandle: dropping loopback IGMP datagram
```

Figure 29. A Sample of a DROPPED in the TRACE Statement

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 30 on page 55 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:
    APOLL0, dest: SA23, len: 256
```

Figure 30. A Sample of an ICMP Trace

IGMP

The IGMP trace provides information about the Internet Group Management Protocol (IGMP). This includes information about joining and leaving IGMP multicast groups. It also displays IGMP query and report messages received and the IGMP reports sent out. Figure 31 on page 56 shows a sample of a IGMP trace. IGMP was specified in the TRACE statement in the PROFILE TCPIP file.

```

DTCIPU055I IgmpAddGroup: Adding multicast group 224.0.0.9 on link TRING interface 9.130.48.70
DTCPD0088I SendIGMP : Sent IGMP report for multicast group 224.0.0.9
on link TRING interface 9.130.48.70
DTCIPU080I IGMPAddGroup: IGMP report message pending for group 224.0.0.9
DTCIPU057I IgmpHandle: received IGMP datagram
DTCPRC001I version: 4
DTCPRC002I Internet Header Length: 5 = 20 bytes
DTCPRC009I Type of Service:Precedence = Routine
DTCPRC010I Total Length: 28 bytes
DTCPRC011I Identification: 0
DTCPRC009I Flags: May Fragment, Last Fragment
DTCPRC009I Fragment Offset: 0
DTCPRC019I Time To Live: 1
DTCPRC020I Protocol: IGMP
DTCPRC021I Header CheckSum: 40975
DTCPRC022I Source Address: 09823046
DTCPRC023I Destination Address: E0000009
DTCIPU049I IP-up sees IGMP datagram, code: 18, source: 9.130.48.70,
dest: 224.0.0.9, group: 224.0.0.9, len: 8
DTCIPU050I IgmpHandle: dropping loopback IGMP datagram
DTCIPU055I IgmpAddGroup: Adding multicast group 224.0.0.9 on link FDNET interface 9.130.248.99
DTCPD0088I SendIGMP : Sent IGMP report for multicast group 224.0.0.9 on link FDNET interface
9.130.248.99
DTCIPU080I IGMPAddGroup: IGMP report message pending for group 224.0.0.9
DTCPD0088I SendIGMP : Sent IGMP report for multicast group 224.0.0.9 on link TRING interface
9.130.48.70
DTCPD0088I SendIGMP : Sent IGMP report for multicast group 224.0.0.9 on link FDNET interface
9.130.248.99
:
DTCIPU057I IgmpHandle: received IGMP datagram
DTCPRC001I version: 4
DTCPRC002I Internet Header Length: 5 = 20 bytes
DTCPRC009I Type of Service:Precedence = Routine
DTCPRC010I Total Length: 28 bytes
DTCPRC011I Identification: 36252
DTCPRC009I Flags: May Fragment, Last Fragment
DTCPRC009I Fragment Offset: 0
DTCPRC019I Time To Live: 1
DTCPRC020I Protocol: IGMP
DTCPRC021I Header CheckSum: 37567
DTCPRC022I Source Address: 0982B001
DTCPRC023I Destination Address: E0000001
DTCIPU049I IP-up sees IGMP datagram, code: 17, source: 9.130.176.1, dest: 224.0.0.1, group: *,
len: 8
DTCIPU059I IgmpHandle: processing IGMP query
DTCIPU078I IgmpHandle: IGMP report message pending for group 224.0.0.9
DTCIPU082I IgmpHandle: completed IGMP query processing

```

Figure 31. A Sample of an IGMP Trace

INITIALIZE

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

Figure 32 on page 57 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
.
.
.
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,

```

Figure 32. A Sample of an INITIALIZE Trace Using MORETRACE (Part 1 of 2)

```

Connection list: Queue size = 1
Ping response or timeout, MSG received
Last touched: 41
Login name: INTCLIEN
Notice list: empty
Reserved socket list: Queue size = 1
5104192:
PrevScb: 13601048
NextScb: 13601048
Client: INTCLIEN
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
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 socket
*.SMTP (25) *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. A Sample of an INITIALIZE Trace Using MORETRACE (Part 2 of 2)

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 on page 58 shows a sample of an IPDOWN trace.

```

Ipdwn: Link: Link Name: TR1, Link Type: IBMTR,
Dev Name: LCS1, Dev Type: LCS, QueueSize: 0
Ipdwn: 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 on page 59](#)

```
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 on page 59](#) 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 on page 59](#).

```
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 on page 60](#). To receive more information from the details provided by MORETRACE, use the MONITORquery function.

```

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!
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

```

Figure 38. A Sample of a MONITOR Trace Using MORETRACE (Part 1 of 2)

```

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
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 39. A Sample of a MONITOR Trace Using MORETRACE (Part 2 of 2)

MULTICAST

The MULTICAST trace provides information about the multicast options associated with sockets. This includes information about setting ttl, loopback, and outgoing interface. It also includes information about joining and leaving multicast groups. [Figure 40 on page 62](#) shows a sample of a MULTICAST trace. MULTICAST was specified in the TRACE statement in the PROFILE TCPIP file.

```

DTC SOC031I SetSockOptIp : Set IP_MULTICAST_TTL : 1
DTC IPU070I Multicast Socket Options
DTC IPU071I Output Interface address : *
DTC IPU072I Time to live (TTL) : 1
DTC IPU073I Loopback : Enabled
DTC IPU075I Number of Multicast groups : 0
DTC SOC032I SetSockOptIp : Set IP_ADD_MEMBERSHIP; multicast group: 224.0.0.9 interface: 9.130.48.70
DTC IPU076I IpMcastAdd: Adding multicast group 224.0.0.9 on link TRING interface 9.130.48.70
DTC IPU070I Multicast Socket Options
DTC IPU071I Output Interface address : *
DTC IPU072I Time to live (TTL) : 1
DTC IPU073I Loopback : Enabled
DTC IPU075I Number of Multicast groups : 1
DTC IPU063I Multicast Group Information
DTC IPU064I Multicast Group Address : 224.0.0.9
DTC IPU065I Interface Address : 9.130.48.70
DTC IPU084I Link Name : TRING
DTC IPU066I Reference Count : 1
DTC IPU067I Report pending : Yes
DTC IPU069I MAC address : C000000040000
DTC SOC032I SetSockOptIp : Set IP_ADD_MEMBERSHIP; multicast group: 224.0.0.9 interface: 9.130.176.198
DTC IPU076I IpMcastAdd: Adding multicast group 224.0.0.9 on link ETRING interface 9.130.176.198
DTC IPU070I Multicast Socket Options
DTC IPU071I Output Interface address : *
DTC IPU072I Time to live (TTL) : 1
DTC IPU073I Loopback : Enabled
DTC IPU075I Number of Multicast groups : 2
DTC IPU063I Multicast Group Information
DTC IPU064I Multicast Group Address : 224.0.0.9
DTC IPU065I Interface Address : 9.130.48.70
DTC IPU084I Link Name : TRING
DTC IPU066I Reference Count : 1
DTC IPU067I Report pending : Yes
DTC IPU069I MAC address : C000000040000
DTC IPU063I Multicast Group Information
DTC IPU064I Multicast Group Address : 224.0.0.9
DTC IPU065I Interface Address : 9.130.176.198
DTC IPU084I Link Name : ETRING
DTC IPU066I Reference Count : 1
DTC IPU067I Report pending : Yes
DTC IPU069I MAC address : 01005E0000009

```

Figure 40. A Sample of a MULTICAST Trace

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 41 on page 63](#) 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: SNMPD
  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
Notify called for ACB 13719104:
  Terminate notice -> Notify (from External interrupt handler)
  Last touched: 70090
  Client name: SNMPD
  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: SNMPD
  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 41. A Sample of a NOTIFY Trace

Figure 42 on page 64 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 42. A Sample of a NOTIFY Trace Using MORETRACE

OSD

The OSD trace provides information about control flows between TCP/IP for z/VM and an OSA Express device. [Figure 43 on page 65](#) shows a sample of an OSD trace.

```

DTCOSD080T OSD initializing
DTCPRI385I Device DEVOSD1:
DTCPRI386I   Type: OSD, Status: Not started
DTCPRI387I   Envelope queue size: 0
DTCPRI388I   Address: 1110
DTCOSD244I OSD device DEVOSD1: To0sd: ScheduleIO INITIALIZE ACB: 00000000 (nil) Type: Accept IP
request
DTCOSD088T To0sd: Acb Received:
DTCPRI048I   61141712:
DTCPRI058I   Have completed I/O -> OSD common routine (from OSD handler)
DTCPRI075I   IoDevice 1110
DTCPRI076I   Csw:
DTCPRI461I   Keys: 00, CcwAddress: 00000000
DTCPRI462I   Status bits: 00, SCFA: 1001
DTCPRI463I   Unit Status: 00, Channel Status: 00
DTCPRI464I   Byte Count: 0
DTCPRI470I   Subchannel Logout: 00000000
DTCPRI471I   Extended Report Word: 00000000
DTCPRI385I Device DEVOSD1:
DTCPRI386I   Type: OSD, Status: CSCH on Read Device
DTCPRI387I   Envelope queue size: 0
DTCPRI388I   Address: 1110

```

Figure 43. A Sample of an OSD Trace

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 44 on page 65 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 44 on page 65. For more information on TCP group processes, see “TCP” on page 98. TCPREQUEST is not listed in the DETAILEDTRACE statement in Figure 44 on page 65, 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 44. 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 45 on page 66 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
  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 45. A Sample of a PING Trace

QDIO

The QDIO trace provides information about data flows between TCP/IP for z/VM and an OSA Express device. [Figure 46 on page 67](#) shows a sample of a QDIO trace.

```
DTCQDI002T QDIO add buffer for device 0642 received return code 0
DTCQDI005I QDIO queue: 00B52B28 Buffer number: 0000001C
DTCQDI014I QDIO device 0642 OUTBOUND MULTICAST/BROADCAST data transfer of 0034 bytes
DTCQDI010I QDIO: SIGA issued to device 0642 FC: 0000 Mask1: 40000000 Mask2: 00000000 CC: 00
```

Figure 46. A Sample of a QDIO Trace

ROUNDTRIP or ROUND-TRIP

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

[Figure 47 on page 67](#) 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 47. 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 48 on page 67](#) 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 48. 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 49 on page 68](#) 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 Extnl interrupt hndlr)
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 hndlr)
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: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

```

Figure 49. A Sample of a SCHEDULER Trace Using MORETRACE (Part 1 of 2)

```

DTC SCH004I Scheduler: 2339528415 Terminate notice -> Notify
DTC PRI048I 32871704:
DTC PRI058I Terminate notice -> Notify (from External interrupt handler)
DTC PRI280I Timeout: 492.394 seconds
DTC PRI098I Client name: TCPMNTA
DTC PRI099I Message identifier:5
DTC SCH004I Scheduler: 2339529294 Accept TCP request -> TCP-request
DTC PRI048I 32871824:
DTC PRI058I Accept TCP request -> TCP-request (from External interrupt handler)
DTC PRI280I Timeout: 64.829 seconds
DTC PRI061I Client name: TCPMNTA
DTC PRI062I Message identifier:14
DTC PRI063I Client call: End TCP/IP service
DTC SCH004I Scheduler: 2339670616 Accept TCP request -> TCP-request
DTC PRI048I 32871824:
DTC PRI058I Accept TCP request -> TCP-request (from External interrupt handler)
DTC PRI280I Timeout: 64.829 seconds
DTC PRI061I Client name: TCPMNTA
DTC PRI062I Message identifier:6
DTC PRI063I Client call: Begin TCP/IP service
DTC SCH004I Scheduler: 2339671667 Accept monitor request -> Monitor
DTC PRI048I 32871824:
DTC PRI058I Accept monitor request -> Monitor (from External interrupt handler)
DTC PRI280I Timeout: 64.829 seconds
DTC PRI061I Client name: TCPMNTA
DTC PRI062I Message identifier:8
DTC PRI063I Client call: Monitor command
DASD 03EE LINKED R/O; R/W BY TCPMNTA
DMSACP723I Z (3EE) R/O
DASD 03EE DETACHED

```

Figure 50. A Sample of a SCHEDULER Trace Using MORETRACE (Part 2 of 2)

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 51 on page 70](#) 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 #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 51. 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 52 on page 70 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 52. 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 53 on page 71 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 53. A Sample of a SOCKET Trace

SSL

The SSL trace provides information about the SSL server's socket activities that are unique to the SSL server and information about secure connections.

Figure 54 on page 72 shows a sample of an SSL trace.

```

18:34:14 DTCSSL007I SkTcpSoc: Socket number 1 assigned by the stack.
18:34:14 DTCSSL009I SetIBMSockOpt: SO_PRIVSOCK issued for socket number 1.
18:34:16 DTCSSL007I SkTcpSoc: Socket number 5 assigned by the stack.
18:34:16 DTCSSL008I SetIBMSockOpt: Socket number 5 is now socket type SO_SSL.
18:34:16 DTCSSL027I Port 1024 being used by the SSL security server.
18:34:16 DTCSSL029I 3 concurrent connections can be handled by the SSL security
server.
18:34:16 DTCSSL024I SkSslAcc: Socket number 6 assigned by the stack for SSL
main accept processing.
18:34:26 DTCSSL025I SkTcpAcc: Socket number 7 assigned by the stack for SSLadmin
accept processing.
18:37:21 DTCSSL001I Connection destined for secure port 423.
18:37:21 DTCSSL003I Certificate label to be used: MEDCERT.
18:37:21 DTCSSL005I TCB 93975872 found for SSL security server.
18:37:21 DTCSSL014I Secure connection opened. Secure connections allowed
decreased to 2.
18:37:21 DTCSSL015I Maximum secure connections not reached. Passive open issued
for SSL security server port 1024.
18:37:21 DTCSSL028I SockAddrSsl: Family: 2,
From_address: 9.130.58.177, From_port:1167,
To_address: 9.130.249.34, To_port: 423, Labe
l: MEDCERT, Other Tcb: 93975872.
18:37:21 DTCSSL007I SkTcpSoc: Socket number 7 assigned by the stack.
18:37:21 DTCSSL024I SkSslAcc: Socket number 8 assigned by the stack for SSL
main accept processing.
18:37:21 DTCSSL008I SetIBMSockOpt: Socket number 7 is now socket type SO_SSL.
18:37:21 DTCSSL030I SSL security server issues a connect for the real server
at address: 9.130.249.34 port: 423.
18:37:21 DTCSSL032I 5 bytes received by SSL00001 from the secure client.
18:37:21 DTCSSL032I 37 bytes received by SSL00001 from the secure client.
18:37:22 DTCSSL031I 1615 bytes sent by SSL00001 to the secure client.
18:37:22 DTCSSL032I 5 bytes received by SSL00001 from the secure client.
18:37:22 DTCSSL032I 68 bytes received by SSL00001 from the secure client.

```

Figure 54. A Sample of an SSL 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 55 on page 72 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 55. 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 56 on page 73 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
  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 56. 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 57 on page 74 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 57. A Sample of a TCPUP Trace

Figure 58 on page 75 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
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

```

Figure 58. A Sample of a TCPUP Trace Using MORETRACE (Part 1 of 3)

```

SndNxt: 500650786
SndUna: 500650566
SndWl1: 568559378
SndWl2: 500650566
SndWnd: 15504
MaxSndWnd: 16384
State: Established
Pending TCP-receive buffer: 2048
WorkOn called:
  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

```

Figure 59. A Sample of a TCPUP Trace Using MORETRACE (Part 2 of 3)

```

    SndNxt: 500650786
    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 60. A Sample of a TCPUP Trace Using MORETRACE (Part 3 of 3)

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 [z/VM: 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 61 on page 78 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 61. 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 62 on page 79](#) 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
.
.
.
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 62. 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 TCP/IP 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 9 on page 80 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 9. 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 10 on page 80 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 10. 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

Table 10. Telnet Command Options from RFC 1060 (continued)

Option	Name
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 63 on page 82 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 9](#) on page 80

```

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
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.

```

Figure 63. A Sample of a TELNET Trace (Part 1 of 2)

```

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
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 64. A Sample of a TELNET Trace (Part 2 of 2)

Figure 66 on page 85 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
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

0
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.
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

```

Figure 65. A Sample of a TELNET Trace Using MORETRACE (Part 1 of 3)

```

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
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.

```

Figure 66. A Sample of a TELNET Trace Using MORETRACE (Part 2 of 3)

```

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: Ugrinfo: Mode is ; Number bytes is 0
Conn 1: StToCpGo returns TOcpDONE.

```

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

TIMER

The TIMER trace shows the processes with time-out marks. [Figure 68 on page 87](#) 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 68. 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 69 on page 88](#) 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
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 69. 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 70 on page 89](#) 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 70. 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 71 on page 90](#) 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
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 71. A Sample of a UDPREQUEST Trace Using MORETRACE

UDPUP

The UDPUP trace provides information about incoming UDP datagrams. Figure 72 on page 91 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 72. 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.

HANDLERS

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

IUCV

The IUCV group process combines IUCV handler and TOIUCV traces. It provides information about IUCV activities. [Figure 74 on page 94](#) 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 74 on page 94](#) 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
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

```

Figure 73. A Sample of an IUCV Trace (Part 1 of 2)

```

PVM IUCV LOCIUCV      : ToIucv PackWrites: Queuesize, SavedEnv: 1 0
PVM IUCV LOCIUCV      : PackWrites packing packet with length 276
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 74. A Sample of an IUCV Trace (Part 2 of 2)

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 75 on page 95 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 75 on page 95 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 195.


```

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
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

```

Figure 75. A Sample of a PCCA Trace (Part 1 of 2)

```

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
Atp 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 76. A Sample of a PCCA Trace (Part 2 of 2)

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 77 on page 97 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 21 on page 196](#)
- 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:
  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: FFFFFFFFFF
  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

```

Figure 77. A Sample of a PCCA Trace Using MORETRACE (Part 1 of 2)

```

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
  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 78. A Sample of a PCCA Trace Using MORETRACE (Part 2 of 2)

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 11. 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.
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.
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.
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.
<p>Note: If IP tracing is required, it is almost always worthwhile to trace IPUP and IPDOWN together.</p> <p>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.</p> <p>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.</p>		
IUCV	IUCV driver (PVMIUCV and IUCV devices) details, including path establishment	No additional tracing
IUCVSIGNON	IUCV driver, path establishment only	No additional tracing

Table 11. Commonly-used Trace Options (continued)

Option name	TRACE output	Addl MORETRACE output
NOTIFY	<p>Tracing related to sending of notifications to the internal client (Telnet server) and VMCF clients (Pascal interface and direct VMCF interface).</p> <p>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.</p>	<p>Additional details.</p> <p>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.</p>
PCCA	<p>LCS driver packet sizes, block headers, I/O interrupts.</p> <p>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.</p>	Packet headers, SIO return codes
PING	Traces ping requests and responses generated by the PingRequest Pascal call or PINGreq VMCF call.	No additional tracing
RAWIPREQUEST	<p>Traces requests using Raw IP through the Pascal interface or VMCF interface. Raw IP routines include</p> <ul style="list-style-type: none"> • RawIpOpen (OPENrawip) • RawIpClose (CLOSErawip) • RawIpSend (SENDrawip) • RawIpReceive (RECEIVERawip) 	<p>IP packet headers as supplied by application, before they are completed by the FillIpHeader routine.</p> <p>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.</p>
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

Table 11. Commonly-used Trace Options (continued)

Option name	TRACE output	Addl MORETRACE output
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 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.
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.

Table 11. Commonly-used Trace Options (continued)

Option name	TRACE output	Addl MORETRACE output
TCPREQUEST	<p>Information pertaining to execution of the following Pascal-interface and VMCF-interface requests:</p> <ul style="list-style-type: none"> • 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) <p>Also traces requests produced by the Version 1 socket interface module, CMSOCKET C, for stream sockets and initialization.</p>	<p>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.</p>
TCPUP	<p>Information related to processing of incoming TCP packets.</p>	<p>In two sample traces of the same traffic, MORETRACE TCPUP generated 14 times as many lines of output as TRACE TCPUP.</p> <p>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.</p>

Table 11. Commonly-used Trace Options (continued)

Option name	TRACE output	Addl MORETRACE output
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: <ul style="list-style-type: none"> • 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.
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 <i>high-order halfword</i> . 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". 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 12. TCP Connection States	
State name	Typical Situation
LISTEN	<p>Waiting for a connection request from the address and port listed in the Foreign Socket column of NETSTAT.</p> <ul style="list-style-type: none"> • "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. <p>If the application uses the Pascal interface or VMCF interface, it has done a TcpOpen (or TcpWaitOpen) with an initial pseudo-state of LISTENING.</p> <p>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.</p>
SYN-SENT	<p>The application has done an "active open" and is waiting for a response from the foreign server.</p> <p>If the application uses the Pascal interface or VMCF interface, it has done a TcpOpen (or TcpWaitOpen) with an initial pseudo-state of TRYINGtoOPEN.</p> <p>If the application uses the socket interface, from C or via IUCV, it has done a connect().</p>
SYN-RECEIVED	<p>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.</p>
ESTABLISHED	<p>Connection is completely established. Both sides can send and receive data. This is the normal state for the data transfer phase of a connection.</p>
FIN-WAIT-1	<p>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.</p>

Table 12. TCP Connection States (continued)

State name	Typical Situation
FIN-WAIT-2	<p>Application has issued a <code>TcpClose</code> or <code>close()</code>. FIN packet was sent and has been acknowledged. TCP is now waiting for the foreign host to send a FIN.</p> <p>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.</p> <p>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).</p> <p>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.</p>
CLOSE-WAIT	<p>The local host has received a FIN from the foreign host and has acknowledged it, but the application hasn't issued a <code>TcpClose</code> or <code>close()</code>.</p> <p>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.</p>
CLOSING	<p>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.</p>
LAST-ACK	<p>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.</p>
TIME-WAIT	<p>Both sides have closed the connection, and all packets have been acknowledged. The connection stays in this state for $2 * \text{MSL}$ (MSL = 60 seconds) as required by the protocol specification, to ensure that foreign host has received the acknowledgment of its FIN.</p> <p>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.</p>
CLOSED	<p>The connection is completely closed.</p> <p>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.</p>

Connection State As Known by Pascal or VMCF Applications

Pascal and direct VMCF applications do not see the actual TCP states described in [Table 12 on page 104](#). 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.

<i>Table 13. Connection Pseudo-states</i>		
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: <ul style="list-style-type: none"> • 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
RECEIVINGonly	Data can be received but not sent on this connection. This means that the client has done a one-way close.	Either: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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.

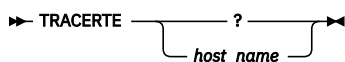
- A return code of -1 from `read()`, `recv()`, etc., with an *errno* value of `ECONNABORTED`, `ECONNRESET`, or `ETIMEDOUT`, indicates that the connection has been abruptly closed (reset) for the given reason.

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. 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 [z/VM: TCP/IP User's Guide](#) for a complete format of the TRACERTE command.



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. Using IPFORMAT Packet Trace Formatting Tool

This chapter describes how to use the IPFORMAT packet trace formatting tool to format and analyze network packet data that has been previously captured using the TRSOURCE and TRACERED commands.

The chapter is broken down into the following subjects:

- IPFORMAT Command Overview
- IPFORMAT Command
- Using IPFORMAT to View Packet Data
- IPFORMAT VIEW Function Keys
- IPFORMAT Subcommands

IPFORMAT Command Overview

Use IPFORMAT to format raw IP packet trace data that has been previously collected and processed using the TRSOURCE and TRACERED commands. Once the IPFORMAT has formatted the raw trace data, the data can be viewed in various summary and detailed forms. IPFORMAT is capable of formatting the following protocols:

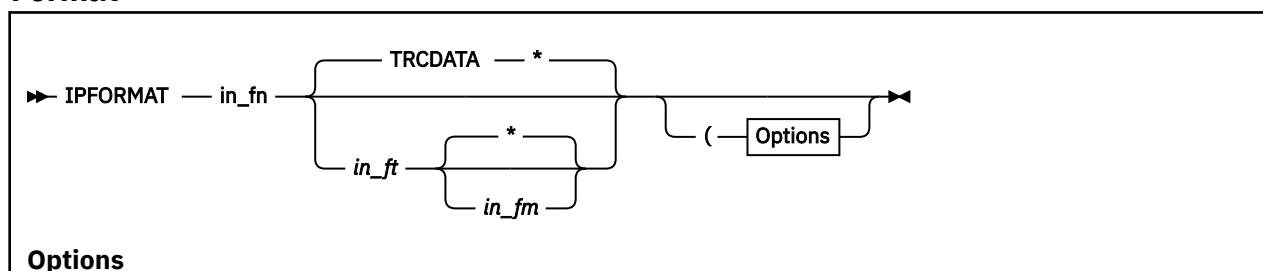
- QDIO and ETHERNET
- IP (IPv4 and IPv6)
- ICMP (IPv4 and IPv6)
- RPC, NFS, FTP, TELNET, SMTP, DNS, RIP, and ARP

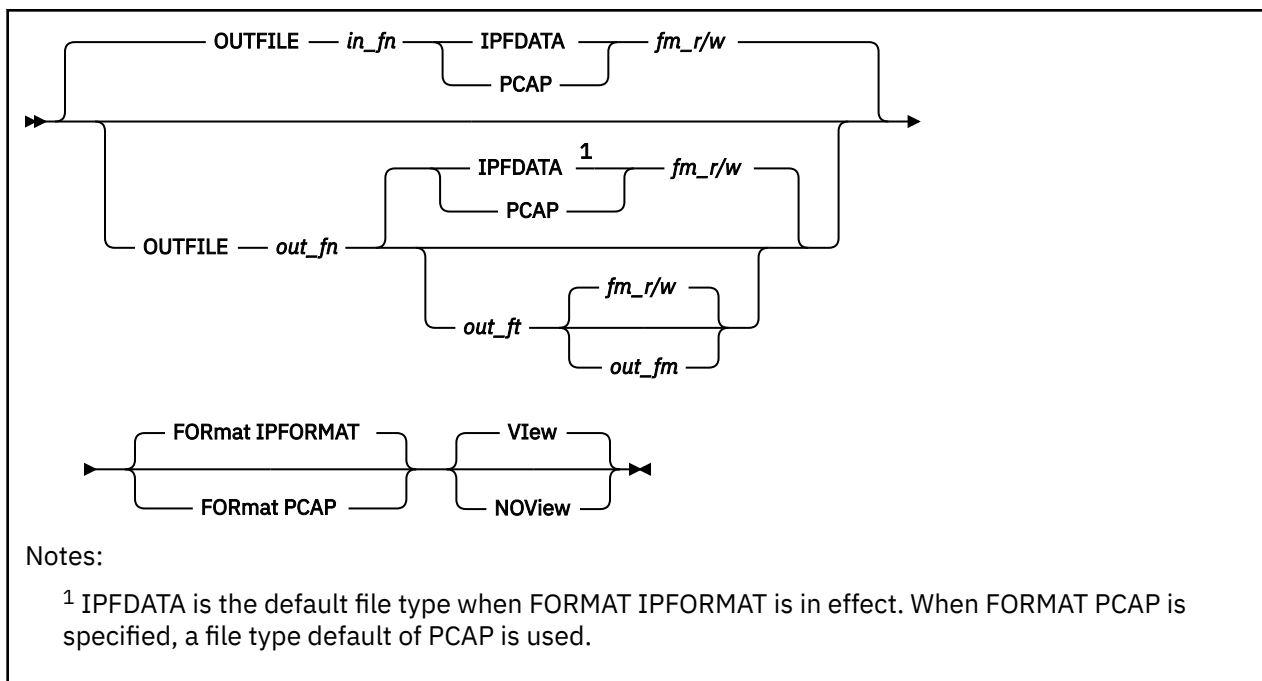
The raw trace data that is used by IPFORMAT is captured using the CP TRSOURCE command and the CP TRACERED command and is saved in a file that is used as input to IPFORMAT. There are two methods for capturing the raw trace data:

- TCP packet trace can be collected by using the PACKETTRACESIZE statement to set a non-zero PACKETTRACESIZE value and then running a TRSOURCE TYPE GT BLOCK trace. (More information about capturing TCP packet trace data using this method can be found in [“TCP/IP Packet Tracing”](#) on page 47).
- In a guest LAN environment, packet trace data can be captured using a TRSOURCE TYPE LAN trace. (More information about using a TRSOURCE TYPE LAN trace to capture packet data can be found in [Troubleshooting a Virtual Switch or Guest LAN](#) chapter in *z/VM: Connectivity*.)

IPFORMAT Command

Format





Purpose

Use IPFORMAT to format raw IP packet trace data that has been previously collected and processed using the TRSOURCE and TRACERED commands.

Operands

in_fn

The file name of a file that contains raw trace data to be processed, or the name of an already converted data file that is to be viewed.

in_ft

The file type of raw data file to be processed, or that of an already converted data file that is to be viewed.

When no file type is specified, a file type of TRCDATA is assumed and IPFORMAT attempts to format the data in such a file.

When a file type of IPFDATA is specified, IPFORMAT verifies the file contains already-converted packet information and then presents that information for review in an IPFORMAT-managed XEDIT session; format processing is not performed.

in_fm

The file mode of the file to be processed or viewed. The default is asterisk (*), which signifies that the first file in the search order that matches the specified name and type is to be used.

Options

OUTFILE out_fn out_ft out_fm

Identifies an output file into which converted IP packet data is to be written. By default, converted data is written to the file *in_fn* **IPFDATA** *out_fm* (where the file name *in_fn* is that of the given input file, and *out_fm* is the first available R/W file mode). If **FORMAT PCAP** is specified, the output is written to the file *in_fn* **PCAP** *out_fm*. The **OUTFILE** option and its operands can be omitted when the output defaults are used.

out_fn

The file name of the file that is to contain converted packet information.

out_ft

The file type of the file that is to contain converted packet information. The default file type is IPFDATA.

out_fm

The file mode of the file that is to contain converted packet information. The default file is to use the first available file mode that has R/W status.

Format

Specifies the data format to be used when formatted trace data is saved to a file. Supported values are:

IPFORMAT

Formatted data is saved in a mostly plain text format, in a packet data file (for which the file type default is IPFORMAT). Data saved in this format can be displayed in an IPFORMAT-managed XEDIT session. This is the default.

PCAP

Data is formatted in PCAP data format, to allow the data to be reviewed and evaluated using a GUI-based trace analysis tool. PCAP-format data is binary data, and must be transferred to the appropriate host system for review in binary form. When FORMAT PCAP is specified, the VIEW and NOVIEW options are ignored. Furthermore, the resulting data is not, nor can it be, displayed in an XEDIT session.

Note that the FORMAT option is not valid if specified for files that already contain converted data.

View

Indicates that formatted packet information should be displayed in an IPFORMAT-managed XEDIT session. Such information is initially presented in a summary format, from which specific packets or groups of packets can be selected for detailed inspection. By default, IPFORMAT presents packet information immediately after raw trace data has been converted. See the [“Using IPFORMAT to View Packet Data”](#) on page 112 for more information about these capabilities.

NOView

Indicates that trace data should be converted only, and not presented for evaluation.

IPFORMAT Configuration File

This section describes the statements used to configure the IPFORMAT program.

Configuration information for the IPFORMAT tool is contained in the IPFORMAT CONFIG file. A sample configuration file is shipped as IPFORMAT SCONFIG. This must be renamed or copied over to IPFORMAT CONFIG before using the IPFORMAT tool. The IPFORMAT CONFIG file defines color attributes and various descriptive substitution values that are to be used when formatted protocol headers and data are displayed by the IPFORMAT utility.

Within the configuration file blanks and <end-of-line> are used to delimit tokens. All characters to the right of, and including a semicolon are treated as a comment.

The format for each configuration entry is:

```

➤ :groupname.  ┌── statement ──┐ :END groupname. ➤
                │               │
                └──┬──────────┘

```

Where:

:groupname

Is a tag that names a group of configuration statements, and which signifies the beginning of each group. Group names that are recognized by the IPFORMAT program are RPCTYPES, SSESSIONCOLORS, TELNETOPTIONS, and TRANSLATE.

statement

Is a configuration statement associated with the previously names group. Details about the format of statements for a given group are documented below.

:END groupname.

Is a terminating tag for a names statement group.

Descriptions of configuration group names recognized by IPFORMAT:

:CMDOPTIONS.

The CMDOPTIONS group defines command option defaults that are to be applied to an IPFORMAT command. For example, the FORMAT PCAP option can be specified here, so that IPFORMAT always will create an output file that contains trace data in PCAP format.

Note:

1. Options specified as part of an IPFORMAT command override those specified using this group.
2. Defaults for the OUTFILE option and its associated values are not supported, and are not permitted within this group.

:RPCTYPES.

The RPCTYPES group defines RPC programs and NFS procedures that correspond to a given program or procedure number when displayed by the IPFORMAT program.

:SESSIONCOLORS.

The SESSIONCOLORS group defines the text colors to be used when data for a given header is displayed in the formatted data view.

:TELNETOPTIONS:

The TELNETOPTIONS group defines descriptive text to be displayed by the IPFORMAT utility for a given telnet option number.

:TRANSLATE.

The TRANSLATE group defines the TCP/IP translation table to be used for converting captured trace data between EBCDIC and ASCII. The specified translation table must have a file type of **TCPXLBIN**. The default is to use the standard translation table (STANDARD TCPXLBIN).

Note: For examples of the use of these configuration group names, refer to the sample IPFORMAT configuration file (shipped as IPFORMAT SCONFIG).

Using IPFORMAT to View Packet Data

This section describes using the IPFORMAT to view packet data.

The Packet Summary View

After raw trace data has been formatted by the IPFORMAT command, it can be viewed in an IPFORMAT-managed Xedit session for analytical purposes. When viewed in this manner, IP packet information is initially presented in summary form, as illustrated here:

Session A - [24 x 80]

File Edit View Communication Actions Window Help

PACKET SUMMARY Packets 1 - 16 of 715 packets

ID	Size	Time	Source	Destination	Proto	Appli
001	544	12:57:03	9.60.59.41	9.60.59.13	ICMP	Desti
002	544	12:57:03	9.60.59.41	9.60.59.13	ICMP	Desti
003	544	12:57:04	9.60.59.1..520	224.0.0.9..520	UDP	RIP
004	544	12:57:10	9.60.59.27..520	224.0.0.9..520	UDP	RIP
005	544	12:57:18	9.60.59.25..520	224.0.0.9..520	UDP	RIP
006	544	12:57:20	9.60.59.47..520	224.0.0.9..520	UDP	RIP
007	544	12:57:20	9.60.59.40..520	224.0.0.9..520	UDP	RIP
008	544	12:57:21	9.60.59.46..520	224.0.0.9..520	UDP	RIP
009	544	12:57:26	9.60.59.34..520	224.0.0.9..520	UDP	RIP
010	544	12:57:33	9.60.59.41	9.60.59.13	ICMP	Desti
011	544	12:57:33	9.60.59.41	9.60.59.13	ICMP	Desti
012	544	12:57:34	9.60.59.1..520	224.0.0.9..520	UDP	RIP
013	544	12:57:40	9.60.59.27..520	224.0.0.9..520	UDP	RIP
014	544	12:57:48	9.60.59.25..520	224.0.0.9..520	UDP	RIP
015	544	12:57:51	9.60.59.47..520	224.0.0.9..520	UDP	RIP
016	544	12:57:51	9.60.59.40..520	224.0.0.9..520	UDP	RIP

1= Help 2= Highlght 3= Quit 4= UndoFilt 5= 6= Right
7= Backward 8= Forward 9= Filter 10= Cursor 11= XEDIT 12= Retrieve

====> _

MA a 24/007

Connected to remote server/host gdlvm7.pok.ibm.com using port 23

Figure 79. Packet Summary of IPv4 Packets

This Packet Summary view presents a summary of IPv4 packets. Each packet from the trace data file has been formatted as a single data record, with certain, preselected attributes displayed in distinct columns. The attributes summarized for each packet are, in order:

- A numeric packet identifier
- The size of the packet
- An abbreviated timestamp
- The source socket
- The destination socket
- A protocol interpretation
- Application name

From this summary view, one or more packets can be selected for detailed inspection, and provides information about each header component of the packet, as well as any contained data. Packets can be selected for this detailed view on an individual basis or through the use of one or more IPFORMAT-provided *filters*. For more information about how packets can be selected for detailed inspection, see “IPFORMAT Subcommands” on page 119 and “IPFORMAT VIEW Function Keys” on page 117.

This next screen image is an example of a packet summary for IPv6 data:

Session A - [24 x 80]

File Edit View Communication Actions Window Help

PACKET SUMMARY Packets 61 - 76 of 126 packets

ID	Size	Time	Source	Destination	Proto	Appli
061	2080	13:00:44	FE80::209:5700:100:6	FE80::209:5700:100:5	ICMP	Neigh
062	2080	13:00:44	FE80::209:5700:100:5	FE80::209:5700:100:6	ICMP	Neigh
063	2080	13:00:52	9.60.59.13.520	224.0.0.9.520	UDP	RIP
064	2080	13:00:52	9.60.59.13.520	224.0.0.9.520	UDP	RIP
065	2080	13:00:52	9.60.59.13.520	224.0.0.9.520	UDP	RIP
066	2080	13:00:52	9.60.59.13.520	224.0.0.9.520	UDP	RIP
067	2080	13:00:52	9.60.59.13.520	224.0.0.9.520	UDP	RIP
068	2080	13:00:52	50C0:C2C1::9:60:59:13	50C0:C2C1::9:60:59:10	ICMP	Echo
069	2080	13:00:52	50C0:C2C1::9:60:59:13	50C0:C2C1::9:60:59:10	ICMP	Echo
070	2080	13:00:52	50C0:C2C1::9:60:59:10	50C0:C2C1::9:60:59:13	ICMP	Echo
071	2080	13:00:52	50C0:C2C1::9:60:59:10	50C0:C2C1::9:60:59:13	ICMP	Echo
072	2080	13:00:52	50C0:C2C1::9:60:59:13	50C0:C2C1::9:60:59:10	ICMP	Echo
073	2080	13:00:52	50C0:C2C1::9:60:59:13	50C0:C2C1::9:60:59:10	ICMP	Echo
074	2080	13:00:52	50C0:C2C1::9:60:59:10	50C0:C2C1::9:60:59:13	ICMP	Echo
075	2080	13:00:52	50C0:C2C1::9:60:59:13	50C0:C2C1::9:60:59:10	ICMP	Echo
076	2080	13:00:52	50C0:C2C1::9:60:59:13	50C0:C2C1::9:60:59:10	ICMP	Echo

1= Help 2= Highlght 3= Quit 4= UndoFilt 5= 6= Right
 7= Backward 8= Forward 9= Filter 10= Cursor 11= XEDIT 12= Retrieve

====> _

Ma **a** 24/007

Connected to remote server/host gdlvm7.pok.ibm.com using port 23

Figure 80. Packet Summary of a mix of IPv4 and IPv6 packets

The Packet Detail View

Once a packet has been selected for detailed inspection, the content of that packet is presented using the Packet Detail view. This view provides a formatted display of header information contained within a packet, as well as any data it contains. The data portion is presented in hexadecimal form, for which either an ASCII or EBCDIC interpretation can be selected.

Attributes for a given packet are also presented at the beginning (top) of the Packet Detail view. The attributes cited are:

- A packet ID
- Packet size information
- Packet arrival time and relative time information

When multiple packets have been selected for inspection, IPFORMAT provides the ability to traverse the chain of selected packets and view the details of each on an individual basis.

The screen image that follows shows a portion of the information presented for a packet using the Packet Detail view. The attributes section and several formatted headers can be seen:

```

Session A - [24 x 80]
File Edit View Communication Actions Window Help
PACKET ID: 10                                     Packets Available - Previous: 9 Next: 705

*****
* Packet ID : 10                                     *
* Arrival Time (Date/Time) .....: 2005-04-19 / 12:57:33.331870 *
* Time Since Previous Packet (sec) .....:          6.733387 *
* Time Relative to First Packet (sec) ...:          30.045236 *
* Packet Length (bytes) .....: 544 *
* Capture Length (bytes) .....: 512 *
*****
{LAN} ***** LAN Trace Header *****
{LAN} Owner ..... : SYSTEM
{LAN} LAN Name .....: SUBNTA
{LAN} Userid .....: TCPIP0D
{LAN} VDEV .....: 0x0D02
{LAN} VLAN ID .....: 0x0001
{LAN} Dropped Code .....: 0x0000 (Packet was delivered)
{LAN} OSA flag .....: 0x00
{LAN} IB/OB .....: 0xFF (Outbound)
{QDIO} ***** Queued Direct I/O Header *****

1= Help      2=          3= Quit      4=          5= PrevPkt   6= NextPkt
7= Backward  8= Forward  9= ASC/EBC 10= Cursor 11= XEDIT   12= Retrieve
====>
ma a
24/007
Connected to remote server/host gdlvm7.pok.ibm.com using port 23

```

Figure 81. Packet Detail of an ICMP Packet (Part 1 of 3)

```

Session A - [24 x 80]
File Edit View Communication Actions Window Help

PACKET ID: 10                      Packets Available - Previous: 9 Next: 705

{QDIO} ***** Queued Direct I/O Header *****
{QDIO} QDIO Header Format .....: 1
{QDIO} Flags Set .....: Uni-cast packet
{QDIO} Sequence Number .....: 0x0000
{QDIO} Token .....: 0x00000000
{QDIO} Datagram Length (bytes) : 56
{QDIO} VLAN Priority .....: 0x00
{QDIO} Extension Flags .....: (None)
{QDIO} VLAN Tag .....: 0x0001
{QDIO} Offset .....: 0
{QDIO} Dest Address .....: 9.60.59.13
{IP} ***** Internet Protocol Header *****
{IP} IP Version: 4
{IP} Type of Service: Routine, Normal Service
{IP} Flags .....: Last Fragment, Allow Fragmentation
{IP} IP Packet Length (bytes) : 56
{IP} IP Header Length (bytes) : 20
{IP} Identification Number ...: 51

1= Help      2=      3= Quit      4=      5= PrevPkt  6= NextPkt
7= Backward  8= Forward  9= ASC/ESC 10= Cursor 11= XEDIT  12= Retrieve
====> _

24/007
Connected to remote server host: gdm7-poc-4m.com, using port: 22

```

Figure 82. Packet Detail of an ICMP Packet (Part 2 of 3)

```

Session A - [24 x 80]
File Edit View Communication Actions Window Help
[Icons]
PACKET ID: 10 Packets Available - Previous: 9 Next: 705

{IP} Identification Number ...: 51
{IP} Fragment Offset ...: 0
{IP} Time to Live ...: 60
{IP} Checksum ...: 0xF5E4
{IP} Source IP Address ...: 9.60.59.41
{IP} Destination IP Address ...: 9.60.59.13
{IP} Protocol ...: ICMP
{IP} Options: (None)
{ICMP} ***** Internet Control Message Protocol Header *****
{ICMP} Type ...: Destination Unreachable; Port Unreachable
{ICMP} Checksum ...: 0xF8B8
Data Length (bytes) ...: 32
Captured Data (bytes) ...: 456 (ASCII)
0000(0038) 00 00 00 00 45 00 00 48 00 33 00 00 01 11 95 20 ....E..H.3..''''
0010(0048) 09 3C 3B 0D E0 00 00 09 02 08 02 08 00 34 00 00 ;<;'~..;'''''''.4..
0020(0058) 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0030(0068) 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0040(0078) 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

1= Help      2=          3= Quit      4=          5= PrevPkt  6= NextPkt
7= Backward  8= Forward  9= ASC/EBC  10= Cursor  11= XEDIT  12= Retrieve

====>
M8 a 24/007
Connected to remote server/host gdlvm7.pok.ibm.com using port 23

```

Figure 83. Packet Detail of an ICMP Packet (Part 3 of 3)

The previous screen image shows the remaining portion of information for this same packet. Packet data is presented last, after the Captured Data heading, and is viewed here in ASCII format.

Note that in the data portion of the preceding screen image, the values in the left-most column are not intrinsic to data within this packet, but are offset values determined by IPFORMAT. The first value indicates the offset of data from the end of the last formatted header (this example is from the end of the TCP (Transmission Control Protocol) header. The second, parenthetical value indicates the offset of the data from the beginning of the packet.

IPFORMAT VIEW Function Keys

This section describes the functions that are assigned to the PF keys when IPFORMAT is invoked. Two distinct function groups are provided, based on whether the Packet Summary view or the Packet Detail view is in effect. The PF key functions assigned for each view are explained in more detail here.

Packet Summary PF Keys

The table that follows shows the functions that are assigned to PF keys when the Packet Summary view is selected:

Table 14. Packet Summary PF Keys	
Key	Function
PF1	Displays a help menu.

<i>Table 14. Packet Summary PF Keys (continued)</i>	
Key	Function
PF2	Visually identifies (highlights) a packet for detailed inspection. Selected packets are then filtered as a group for further examination when <Enter> is pressed.
PF3	Ends the packet display session.
PF4	Returns to the previous summary menu, prior to having performed the most recent filter action (if any).
PF5	Scrolls the screen to the left.
PF6	Scrolls the screen to the right.
PF7	Scrolls backward one screen length.
PF8	Scrolls forward one screen length.
PF9	Filters packets on a column (and in some cases, a partial-column), basis. All packets having identical data for the selected column value are filtered for detailed examination.
PF10	Toggles the cursor between the command line and the packet record area.
PF11	Initiates an editing session of the currently displayed data, after having placed that data in a temporary file.
PF12	Retrieves the last command that was entered.

Packet Detail PF Keys

The table that follows shows the functions that are assigned to PF keys when the Packet Detail view is selected:

<i>Table 15. Packet Detail PF Keys</i>	
Key	Function
PF1	Displays the help menu.
PF2, PF4, and PF9	No function.
PF3	Returns to the current Packet Summary view.
PF5	Changes the detailed view to that for the previous available packet (if any) of the selected group
PF6	Changes the detailed view to that for the next available packet (if any) of the selected group
PF7	Scrolls backward one screen length.
PF8	Scrolls forward one screen length.
PF9	Toggles between the ASCII or EBCDIC data representation.

Table 15. Packet Detail PF Keys (continued)

Key	Function
PF10	Toggles the cursor between the command line and the packet record area.
PF11	Initiates an editing session of the currently displayed data, after having placed that data in a temporary file.
PF12	Retrieves the last command that was entered.

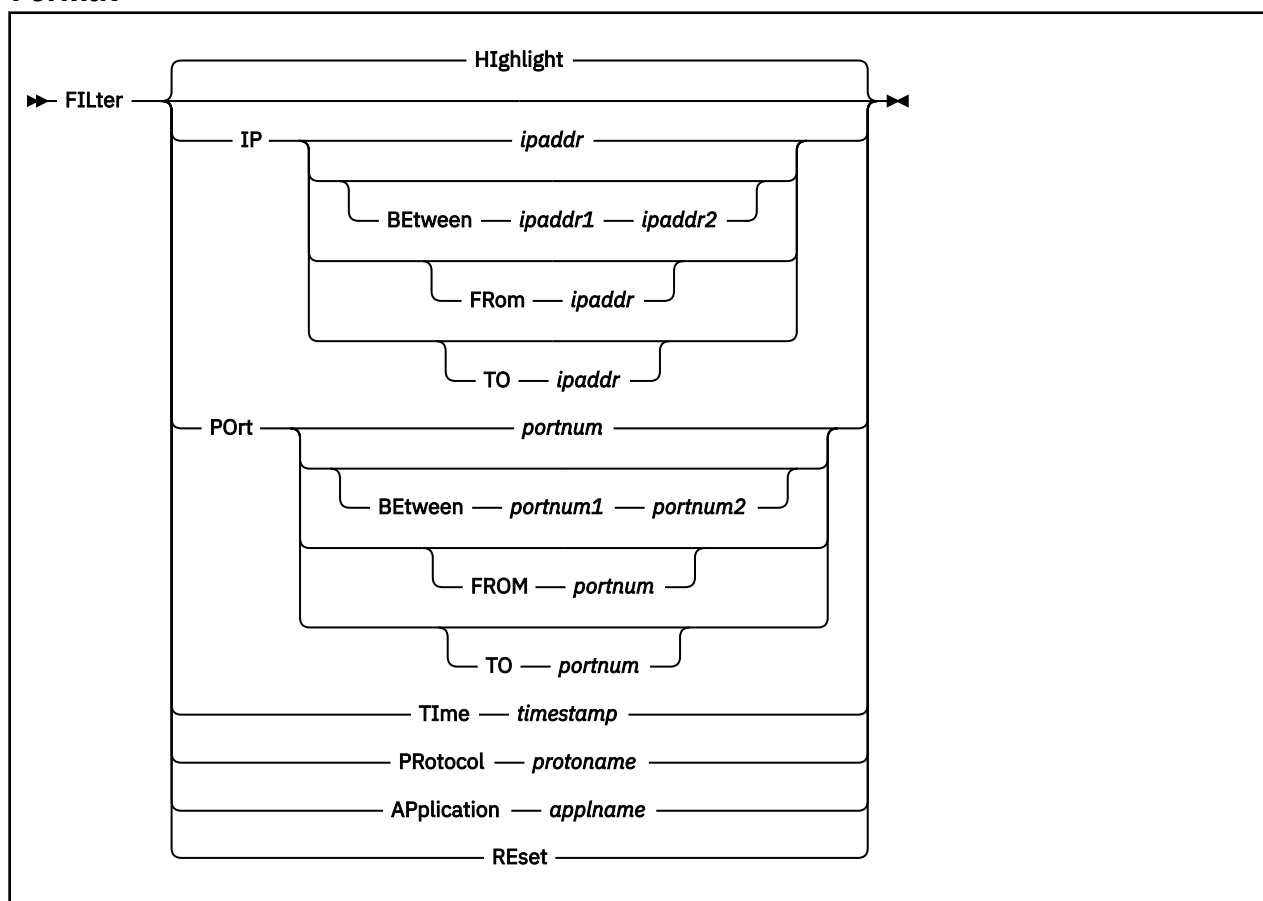
IPFORMAT Subcommands

IPFORMAT provides several different subcommands to assist with the analysis of formatted packet information, as well as to save specific portions of that information in readable form for later reference.

To invoke an IPFORMAT subcommand, simply type the command on the command line. Descriptions of available subcommands follow.

FILTER Subcommand

Format



Purpose

Use the FILTER subcommand to select one or more packets from the summary view for detailed display. Packet selection criteria is determined using one of the FILTER operands that follows. Sequential FILTER subcommands can be used as needed; isolate packets to those that share specific attributes.

Operands

Highlight

Selects packets that have been identified on an individual basis through use of the Highlight PF key. The HIGHLIGHT filter is the default.

IP

Selects packets based on a source IP address, destination IP address, or both such addresses.

BETween

Specifies that packet selection is to be based on the provided source and destination IP addresses. That is, only packets that have travelled between the designated hosts are selected.

FRom

Specifies that packet selection is to be based on the provided source IP address.

TO

ipaddr, ipaddr1, ipaddr2

A host IP address (or addresses) on which IP filtering is to be based.

PORT

Selects packets based on a source port number, destination port number, or both such numbers.

BETween

Specifies that packet selection is to be based on the provided source and destination port numbers. That is, only packets that have travelled between the designated hosts are selected.

FRom

Specifies that packet selection is to be based on the provided source port number.

TO

Specifies that packet selection is to be based on the provided destination port number.

portnum, portnum1, portnum2

A TCP or UDP host port number (or numbers) on which port filtering is to be based.

Time

Selects packets based on the time that they were received. Only packets whose timestamp matches the timestamp provided (or whose timestamp contain a match if only a partial timestamp is provided) are selected for presentation.

timestamp

The time (in the format *hh:mm:ss*) on which time filtering is to be based.

PROtocol

Selects packets based on a specific transport protocol. Only packets associated with this protocol are selected for presentation.

protoname

The name of the transport protocol to be used for filtering (for example: TCP, UDP, or ICMP).

APplication

Selects packets based on a specific application name. Only packets associated with this application are selected for presentation.

applname

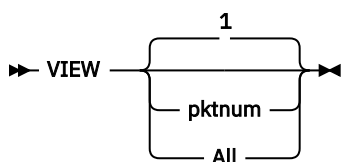
The name of an application to be used for filtering (for example: RPC, NFS, FTP, TELNET, SMTP, DNS, or RIP).

REset

Cancels all active filters, and restores an unfiltered packet summary. When the RESET subcommand is used, all filtered views are lost.

VIEW Subcommand

Format



Notes:

¹ When no operands are specified, packets that have been selected using the Highlight filter are displayed.

Purpose

Use the VIEW subcommand to display detailed information for selected packets.

Operands

pktnum

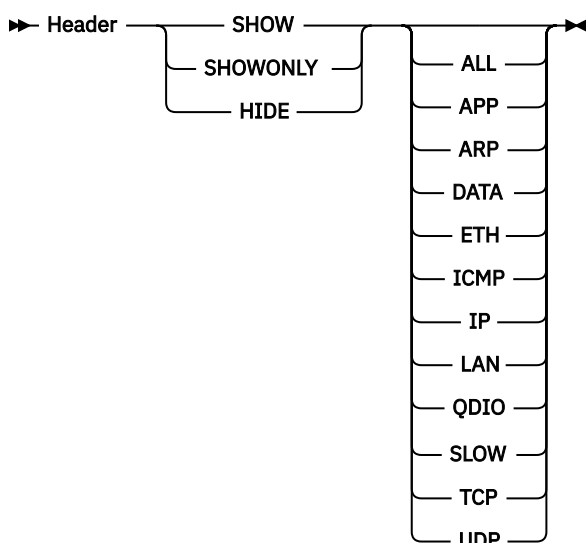
Defines a packet number. The formatted data for the packet with this number should be displayed.

ALL

Specifies that the formatted packet data for all packets should be displayed.

HEADER Subcommand

Format



Purpose

Use the HEADER subcommand to limit the display of formatted information to that associated with a specific type of packet header, or to restore the display of previously suppressed information for a specified type of header.

SAVE

Operands

SHOW

Specifies that information associated with the indicated header should be displayed, if it is not already displayed. If such information is already displayed, then no change is made to the current display of packet data.

SHOWONLY

Specifies that information associated with only the indicated header should be displayed. Information associated with any other type of header is suppressed.

HIDE

Specifies that information associated with the indicated header should be suppressed, if it is not already suppressed. If such information is already suppressed, then no change is made to the current display of packet data.

ALL

Specifies that all header information is to be shown or hidden.

APP

Specifies that application header information is to be shown or hidden.

ARP

Specifies that ARP information is to be shown or hidden.

DATA

Specifies that the data portion of a packet is to be shown or hidden.

ETH

Specifies that Ethernet header information is to be shown or hidden.

ICMP

Specifies that only ICMP header information is to be shown or hidden.

IP

Specifies that only IP header information is to be shown or hidden.

LAN

Specifies that only information from the LAN trace block header is to be shown or hidden.

QDIO

Specifies that QDIO header information is to be shown or hidden.

SLOW

Specifies that Slow Protocols (LACP or Marker) information is to be shown or hidden.

TCP

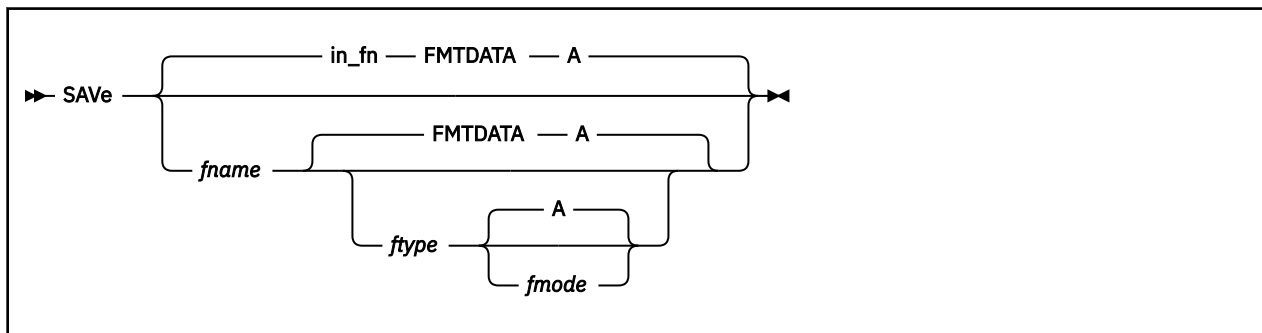
Specifies that TCP header information is to be shown or hidden.

UDP

Specifies that UDP header information is to be shown or hidden.

SAVE Subcommand

Format



Purpose

Use the SAVE subcommand to write currently displayed summary or formatted header data to a CMS file.

Operands

fname

The file name of the file in which data is to be saved. The default is to use the same file name as that of the original input file.

ftype

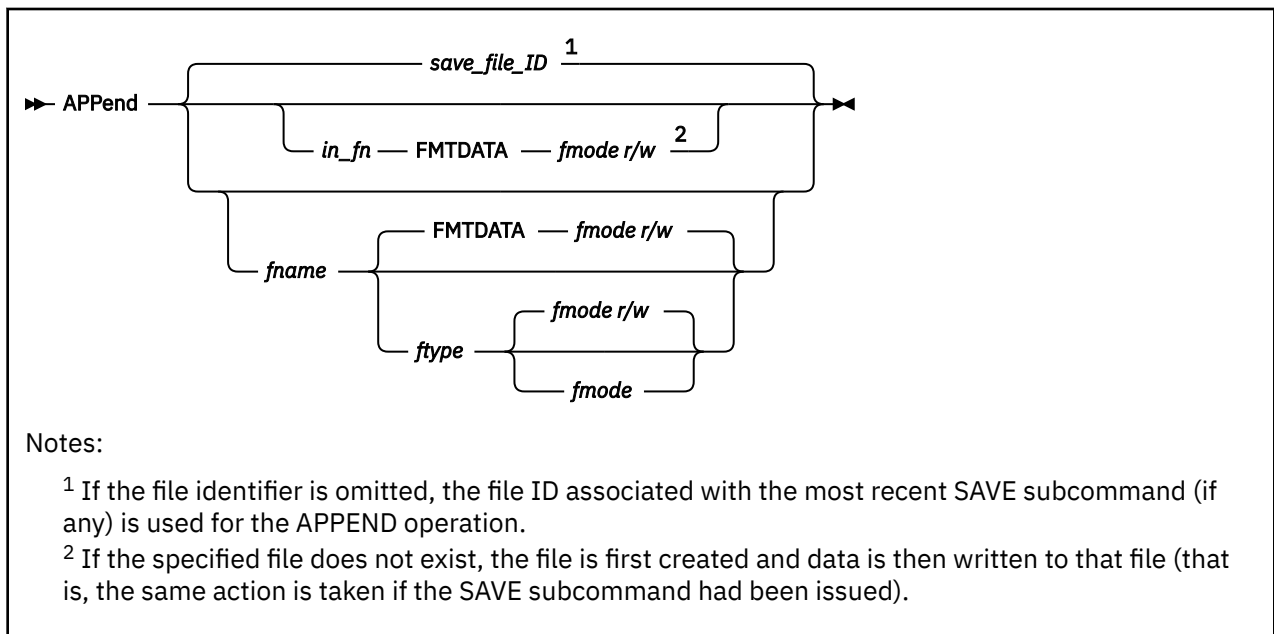
The file type of the file in which data is to be saved. The default file type is FMTDATA.

fmode

The file mode of the file in which data is to be saved. The default is to use the first available file mode that has R/W status.

APPEND Subcommand

Format



Purpose

Use the APPEND subcommand to write currently displayed summary or formatted header data to an existing CMS file.

Operands

fname

The file name of the file to which data is to be appended. If no file name is specified, the default is to use the file identifier that is associated with the most recent SAVE subcommand. If no such file ID exists, then the default is to use the same file name as that of the original input file.

ftype

The file type of the file to which data is to be appended. The default file type is FMTDATA.

fmode

The file mode of the file to which data is to be appended. The default is to use the first available file mode that has R/W status.

Usage Notes

1. If the file identifier is omitted, the file ID associated with the most recent SAVE subcommand (if any) is used for the APPEND operation.
2. If the specified file does not exist, the file is first created and data is then written to that file (that is, the same action is taken as if the SAVE subcommand had been issued).

Chapter 9. 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 84 on page 125](#) shows the relationship between user and server functions.

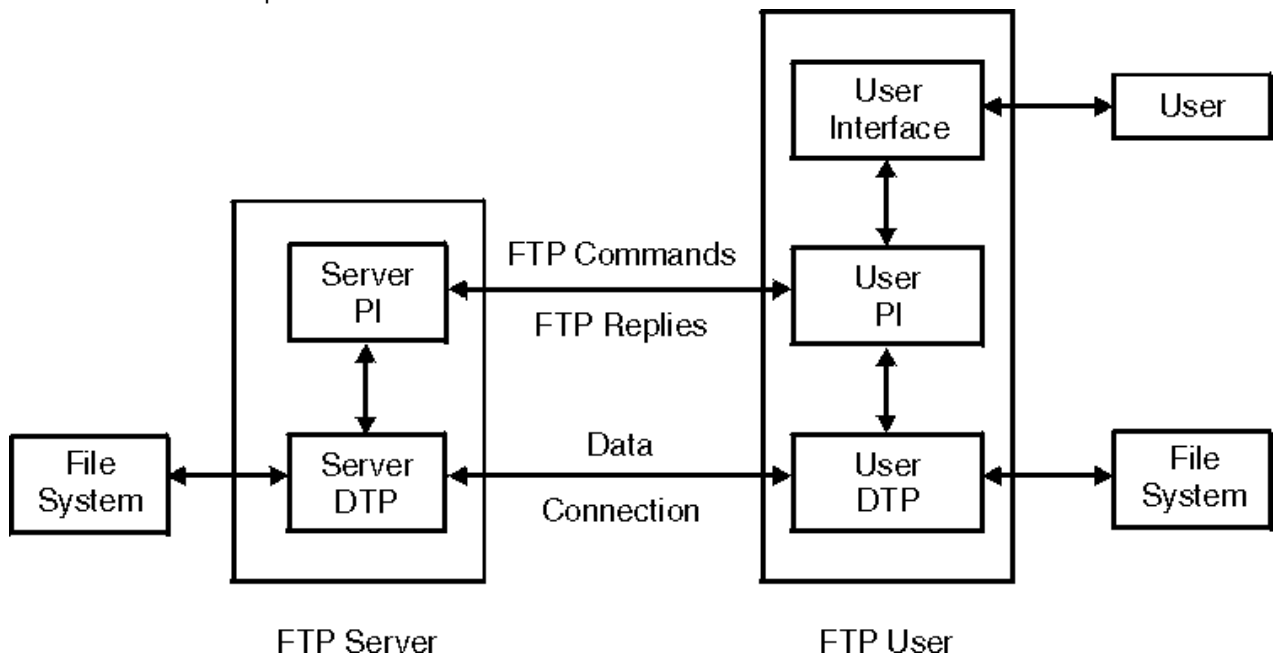


Figure 84. The FTP Model

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 operands 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 or EPRT command. The port is then assigned by TCPIP after an open request. The format of the EPRT and PORT command is:



1 and 2

Select the protocol to use - IPv4 and IPv6 respectively.

d1.d2.d3.d4

Is the 32-bit IPv4 address in decimal dotted quad notation like 9.67.58.226.

p1

Is the port number. It is 16-bit unsigned integer value like 1096.

x2

Is the 128-bit IPv6 address in hex-colon notation like 1080::8:800:200C:417A.

Prior to beginning a data transfer, the FTP client must indicate to the server whether the client or server should initiate the data connection. For IPv6 connections, the VM FTP client will always use the EPSV command to indicate that client will initiate the data connection. For IPv4 connections, EPSV is the default command but this may be changed via the ESPV4 and FWFRIENDLY configuration statements in the FTP DATA file (or their corresponding LOC SITE subcommands). The PASV command, like EPSV, indicates that the data connection will be initiated by the client while the PORT command indicates that the server will initiate the connection. The following table shows the relationship between the configuration statements and the FTP commands that will be used:

	IPv4 Connection	IPv6 Connection
EPSV4 True & FWFriendly True	Use EPSV, if that fails, use PASV	Use EPSV
EPSV4 True & FWFriendly False	Use EPSV, if that fails, use PORT	Use EPSV
EPSV4 False & FWFriendly True	Use PASV	Use EPSV
EPSV4 False & FWFriendly False	Use PORT	Use EPSV

For server-initiated connections, once the operands from the data connections have been transmitted, the user-DTP must be in listen mode. The server initiates the data connection using the data port indicated by the client. For VM FTP implementations, the client issues a PORT command. The port is then assigned by TCP/IP after an open request.

For client-initiated connections, once the operands from the data connections have been transmitted, the server-DTP must be in listen mode. The client initiates the data connection using the data port indicated by the server. For VM FTP implementations, the client issues a EPSV/PASV command.

The format of the PORT, EPSV and PASV command is:

➤ PORT — <i>h1.h2.h3.h4.p1.p2</i> ➤

The only operand for the PORT command is:

Operand
Description

h1.h2.h3.h4.p1.p2

Is the address space for the default data port. The port specification is a conventional IP address to which a 16 bit TCP port address is concatenated, where each byte of the port address value is represented using separate decimal numbers (*p1.p2*). For example, the port specification 9.67.58.226.4.72 represents (decimal port) 1096 on the host with IP address 9.67.58.226.

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

➤ EPSV — <i>net_pro</i> ➤

The only operand for the EPSV command is:

Operand
Description

net_pro

Select the protocol to use - IPv4 and IPv6 respectively, the valid value is 1 and 2.

```
➤ PASV ➤
```

There is no operand for the PASV command.

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** operand in addition to the usual processing operands 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 command using the TRACE option:

```
➤ FTP — foreignhost — portnumber ( TRACe ) ➤
```

For information on all of the possible operands of the FTP command, see the [z/VM: TCP/IP User's Guide](#).

The operands for the FTP command are:

Operands

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 operand is usually used for system testing only.

TRACe

Starts the generation of tracing output. TRACE is used to assist in debugging.

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 operands.

Trace output is directed to the virtual machine console.

For more information about the FTP command and DEBUG subcommand, see the [z/VM: 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 85 on page 129](#) 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 IBMHOST.IBM.COM (TRACE
VM TCP/IP FTP Level 640
Translate Table: STANDARD
about to call BeginTcpIp
DBG: SECURECONTROL NO CERTFULLCHECK
DBG: SECUREDATA NO CERTFULLCHECK
Connecting to IBMHOST.IBM.COM 192.0.2.25, port 21
SysAct 0 21 192.0.2.25 CC -1
==> Active open to host 192.0.2.25 port 21 from host 0 port 65535
In SysActiveOpen: ConnState Open for Fd 1
In SysRead, calling TcpFReceive with args: 0 0065AC40 8192
In SysRead: Note received: => TcpId 0 Data delivered 145 bytes Push
In SysRead, TcpFReceive returned: 145
220-FTPSERVE IBM VM Level 640 at IBMHOST.IBM.COM, 10:00:09 EDT MONDAY 2017-05-08
220 Connection will close if idle for more than 5 minutes.
GetReply returns 220
entering ReadNETRCfile
NETRC DATA file not found, Rc = 3
leaving ReadNETRCfile
USER (identify yourself to the host):
===testuser
>>>USER testuser
In SysSendFlush, calling TcpWaitSend with args: 0 00658948 14
In SysSendFlush, TcpWaitSend returned: OK
In SysRead, calling TcpFReceive with args: 0 0065AC40 8192
In SysRead: Note received: => TcpId 0 Data delivered 27 bytes Push
In SysRead, TcpFReceive returned: 27
331 Send password please.
GetReply returns 331
Password:
===----- (non-display entry)
>>>PASS *****
In SysSendFlush, calling TcpWaitSend with args: 0 00658948 15
In SysSendFlush, TcpWaitSend returned: OK
In SysRead, calling TcpFReceive with args: 0 0065AC40 8192
In SysRead: Note received: => TcpId 0 Data delivered 56 bytes Push
In SysRead, TcpFReceive returned: 56
230 TESTUSER logged in; working directory = TESTUSER 191
GetReplCodeText returns 230 230 TESTUSER logged in; working directory = TESTUSER 191
leaving dologin

```

Figure 85. A Sample of an FTP Client Trace (Part 1 of 2)

```

Command:
===GET TEST.FILE
DBG: SendEpsvCommand: Local=: Foreign=192.0.2.25 Proto=1 ModePort= TRUE
>>>EPSV 1
In SysSendFlush, calling TcpWaitSend with args: 0 00658948 8
In SysSendFlush, TcpWaitSend returned: OK
In SysRead, calling TcpFReceive with args: 0 0065AC40 8192
In SysRead: Note received: => TcpId 0 Data delivered 49 bytes Push
In SysRead, TcpFReceive returned: 49
229 Entering Extended Passive Mode. (|||15676|)
GetReplCodeText returns 229 229 Entering Extended Passive Mode. (|||15676|)
DBG: SendEpsvCommand: Code=229 Reply="229 Entering Extended Passive Mode. (|||15676|)"
DBG: SendEpsvCommand:229 DataHost=192.0.2.25 DataPort=15676
==> Active open to host 192.0.2.25 port 15676 from host 0 port 65535
In SysActiveOpen: ConnState Open for Fd 2
DBG: DoTrFile: ModePort= TRUE Local Address=192.0.2.20 DataPort=50981 DataConn=2 Result=0
DBG: * PrintConn(1):
DBG: : Local=192.0.2.20; Remote=192.0.2.25.
>>>RETR TEST.FILE
In SysSendFlush, calling TcpWaitSend with args: 0 00658948 15
In SysSendFlush, TcpWaitSend returned: OK
In SysRead, calling TcpFReceive with args: 0 0065AC40 8192
In SysRead: Note received: => TcpId 0 Data delivered 51 bytes Push
In SysRead, TcpFReceive returned: 51
125 Sending file 'TEST.FILE' FIXrecfm 80
GetReplCodeText returns 125 125 Sending file 'TEST.FILE' FIXrecfm 80
Filename: "TEST.FILE.A"
In Openfscb: OUTFILE mode, using ESTATEW
In OpenFscb, DMSQFMOD rc=0, rsc=0, fmode=A
In OpenFscb, sfopen rc = 0
In DoTrFile: GETFILE processing. Openfscb rc = 0
Transferring in AsciiToRecord
In GetFromTcp, calling TcpFReceive with args: 1 1FAD4FF8 32768
In GetFromTcp: Note received: => TcpId 1 Data delivered 7626 bytes
GetFromTcp: 7626 bytes in buffer
In GetFromTcp, TcpFReceive returned: 0
In GetFromTcp, calling TcpFReceive with args: 1 1FAD4FF8 32768
In GetFromTcp: Note received: => TcpId 1 Connection state changed Sending only
In GetFromTcp: NewState Sending only for Fd 2
In GetFromTcp, TcpFReceive returned: 0
Sysclose called with Fd = 2
In SysClose: Note received: => TcpId 1 Connection state changed Connection closing
In SysClose: NewState Connection closing for Fd 2
Exiting from SysClose: Fd = 2, TcpId = 1
In SysRead, calling TcpFReceive with args: 0 0065AC40 8192
In SysRead: Note received: => TcpId 1 Connection state changed Nonexistent
In SysRead: NewState Nonexistent for TcpId 1
In SysRead: Note received: => TcpId 0 Data delivered 38 bytes Push
In SysRead, TcpFReceive returned: 38
250 Transfer completed successfully.
GetReply returns 250
7626 bytes transferred in 0.008 seconds. Transfer rate 953.25 Kbytes/sec.
Command:
===quit
>>>QUIT
In SysSendFlush, calling TcpWaitSend with args: 0 00658948 6
In SysSendFlush, TcpWaitSend returned: OK
In SysRead, calling TcpFReceive with args: 0 0065AC40 8192
In SysRead: Note received: => TcpId 0 Data delivered 37 bytes Push
In SysRead, TcpFReceive returned: 37
221 Quit command received. Goodbye.
GetReply returns 221
Sysclose called with Fd = 1
In SysClose: Note received: => TcpId 0 Connection state changed Sending only
In SysClose: NewState Sending only for Fd 1
In SysClose: Note received: => TcpId 0 Connection state changed Connection closing
In SysClose: NewState Connection closing for Fd 1
Exiting from SysClose: Fd = 1, TcpId = 0
SysHalt has been Called

```

Figure 86. A Sample of an FTP Client Trace (Part 2 of 2)

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

192.0.2.25

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.

TcpFReceive

Name of a TCPIP client procedure.

0

ID of the connection between the TCPIP and FTP client.

0065AC40

Buffer address that contains the data or text to be sent by the remote host.

8192

Buffer size authorized by the client.

145

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.

00658948

Buffer address of the data or command sent to the remote host.

14

Length of the command sent plus CR/LF.

4. FTP issues an EPSV command to the FTP server, asking the server to perform a passive open for the data connection. The FTP server responds with the port number to be used for the active open the FTP client will be performing.

Trace Item**Description****EPSV 1**

The request from the FTP client for the FTP server to perform a passive open

15676

The port the FTP server issued the passive open on. The FTP client issue an active open on this port and the FTP server will connect to that port.

In SysActiveOpen

Name of the FTP client procedure.

Fd 2

Internal connection slot number in the FTP client program; the Fd for the first control connection is 1.

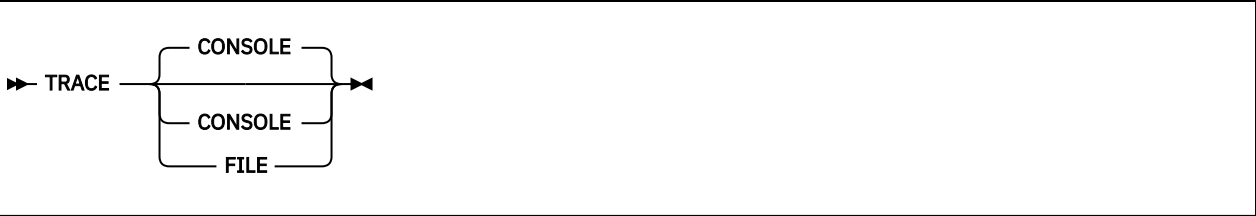
- 5. FTP transfers the requested file by calling TcpFReceive to retrieve the data sent by the server. Once the file transfer is completed, the data connection is closed.
- 6. When the QUIT command is issued, the control connection is closed and the FTP client exits.

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 at FTP server initialization time by specifying the TRACE statement in the FTP server configuration file (SRVRFTP CONFIG), or dynamically by using the FTP server SMSG interface to issue an SMSG TRACE command. The following is the format for the FTP server configuration file TRACE statement:



The operands for the TRACE statement are:

Operands

Description

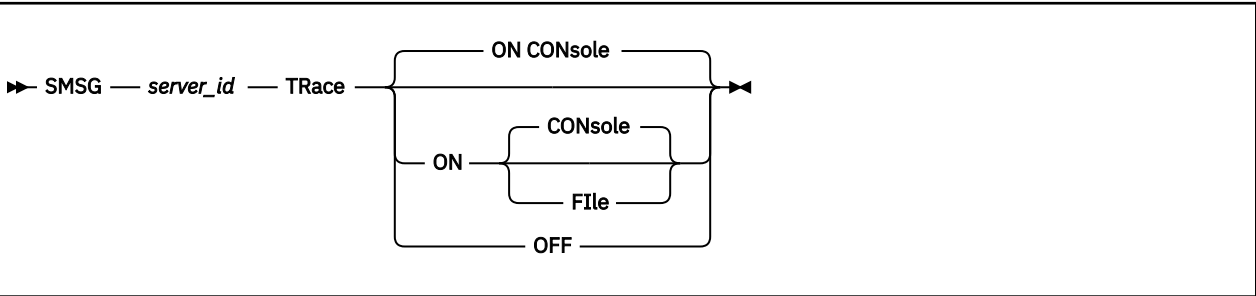
CONSOLE

Specifies that trace information should be directed to the FTP server console.

FILE

Specifies that trace information should be directed to the FILE DEBUGTRA file on the FTP server 191 minidisk.

To enable or disable FTP server tracing interactively, use the FTP server SMSG TRACE command. The following is the format of the FTP server SMSG TRACE command:



The operands for the SMSG TRACE command are:

Operands

Description

server_id

Specifies the user ID of the FTP server virtual machine.

OFF

Disables server tracing.

ON CONSOLE

Enables server tracing and directs trace information to the FTP server console.

ON FILE

Enables server tracing and directs trace information to the FILE DEBUGTRA file on the FTP server 191 minidisk. If the trace file already exists, its previous contents are deleted.

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, FTP server operations, and the status of the data and control connections are also documented. Since the FTP server TRACE function records all FTP server activity and writes the trace information to either the FTP server console or FILE DEBUGTRA file, using the TRACE function significantly degrades FTP server performance and should only be used for debug purposes.

Figure 87 on page 133 shows a sample of an FTP Server Trace.

```

DTCFTS0359I Filemode 'A' will be used for reader file support
DTCFTS7008I Server-FTP: CHKIPADRfound = TRUE
DTCFTS8507I AUDITexitINuse=FALSE, COMMANDexitINuse=FALSE, CDexitINuse=FALSE
DTCFTS0371I Default list format is UNIX
DTCFTS0373I Default automatic translation is turned ON
DTCFTS1248I z/VM Version 4 Release 3.0, service level 0000 (32-bit)
DTCFTS1248I CMS Level 19, Service Level 000
DTCFTS8099I SystemInitialize: Diagnose 88, class B check, DMSLINK rc=4 rsc=0
DTCFTS7003I Diagnose 88 authorization and Class B privilege confirmed
DTCFTS8112I In SystemInitialize, OpenVMF: Function code 3, Rval 0, rc 0, rsc 0.
DTCFTS2619I No VMFILETYPEDEFAULT statement in TCPIP DATA file
DTCFTS2620I No VMFILETYPE statement in TCPIP DATA file
DTCFTS4024I OpenConnection(00000000,21,00000000,65535,2147483647,FALSE
DTCFTS4013I AdvertizeService gets connection #0
DTCFTS8603I -----
DTCFTS0023I Got note Connection state changed for #0, Trying to open
DTCFTS4024I OpenConnection(00000000,21,00000000,65535,2147483647,FALSE
DTCFTS4013I AdvertizeService gets connection #1
DTCFTS8603I -----
DTCFTS0023I Got note Connection state changed for #0, Open
DTCFTS2502I Allocating buffer of 8192 bytes
DTCFTS4050I Send reply '220-FTPSERVE IBM VM Level 430 at TCPIPDEV.ENDICOTT.IBM.COM, ...'
DTCFTS4050I Send reply '220 Connection will close if idle for more than 5 minutes.'
DTCFTS4026I ReinitConn(0)
DTCFTS4020I GetData(0)
DTCFTS4022I In GetData, TcpFReceive: Where = 1
DTCFTS8603I -----
DTCFTS0026I Got note Data delivered for #0, 15 bytes
DTCFTS2560I 15 bytes arrived on conn #0
DTCFTS7022I Command Received on conn #0: USER TCPUSER1
DTCFTS8196I IP address checking returns 0.
DTCFTS2581I In VMIPAdrChk, list format changed to VM
DTCFTS2591I In VMIPAdrChk, automatic translation turned ON
DTCFTS4050I Send reply '331 Send password please.'
DTCFTS4020I GetData(0)
DTCFTS4022I In GetData, TcpFReceive: Where = 1
DTCFTS8603I -----

```

Figure 87. A Sample of an FTP Server Trace (Part 1 of 4)

```

DTCFTS0026I Got note Data delivered for #0, 11 bytes
DTCFTS2560I 11 bytes arrived on conn #0
DTCFTS7022I Command Received on conn #0: PASS ftp4you
DTCFTS8096I CheckPassword: DMSPWCHK for User:'TCPUSER1', ByUser:'', rc=0
DTCFTS8060I LogData: 'Diag 0x88/0 Agent=(TCPUSER1,****) Target=TCPUSER1 RC=0'
DTCFTS8005I MinidiskLink(0) for TCPUSER1 191
DTCFTS8006I NewVirtual = 351
DTCFTS8001I DMSLINK rc=0 rsc=0 owner="TCPUSER1" agent="TCPUSER1".
DTCFTS8002I DMSLINK mdiskaddr="0191" vaddr="0351" Pass=" " ESMtoken=0.
DTCFTS8060I LogData: 'Diag 0x88/4 Agent=TCPUSER1 Target=(TCPUSER1.191,351,X,) RC=(0,0)'
DTCFTS8259I User TCPUSER1 working directory changed to TCPUSER1.191
DTCFTS8007I MinidiskLink Result = 0, VirtAddr = 351, Writable = TRUE
DTCFTS8008I Owner TCPUSER1, Addr 191, NewOwner TCPUSER1, NewAddr 191
DTCFTS4050I Send reply '230 TCPUSER1 logged in; working directory = TCPUSER1 191'
DTCFTS4020I GetData(0)
DTCFTS4022I In GetData, TcpFReceive: Where = 1
DTCFTS8603I -----
DTCFTS0026I Got note Data delivered for #0, 26 bytes
DTCFTS2560I 26 bytes arrived on conn #0
DTCFTS7022I Command Received on conn #0: PORT 9,117,222,18,70,189
DTCFTS4050I Send reply '200 Port request OK.'
DTCFTS4020I GetData(0)
DTCFTS4022I In GetData, TcpFReceive: Where = 1
DTCFTS8603I -----
DTCFTS0026I Got note Data delivered for #0, 6 bytes
DTCFTS2560I 6 bytes arrived on conn #0
DTCFTS7022I Command Received on conn #0: LIST
DTCFTS8124I In FindMode, "CMS ACCESS 351 B", rc 0.
DTCFTS8094I FindMode minidisk TCPUSER1.191 accessed as 351 B
DTCFTS8125I In DoSFMinidiskList, "CMS LISTFILE * * B ( EXEC LABEL NOHEADER ALLFILE", rc 0.
DTCFTS4024I OpenConnection(0982F92E,20,0975DE12,18109,30,TRUE
DTCFTS4050I Send reply '125 List started OK'
DTCFTS8014I DoList: Sopenfscb of clean file
DTCFTS7009I SOpenfscb: name is: CONN-2.FTPLIST.A
DTCFTS7010I SOpenFscb: ESTATE returns: 0
DTCFTS6005I SOpenFscb: recfm: V lrecl: 79
DTCFTS4020I GetData(0)
DTCFTS4022I In GetData, TcpFReceive: Where = 1
DTCFTS8603I -----
DTCFTS0023I Got note Connection state changed for #2, Open
DTCFTS2502I Allocating buffer of 131072 bytes
DTCFTS2503I Allocating RdFromDiskBuf of 8192 bytes
DTCFTS2508I Data connection 2 open for sending
DTCFTS4052I ReinitDataConn(2)
DTCFTS4053I FtpFormat: A FtpMode: S FtpOptFormat: 0
DTCFTS4054I RecordFormat: V RecordLength: 65535
DTCFTS4058I AutomaticTranslation: ON
DTCFTS4027I StartTransfer for 2:
DTCFTS4031I Xfread: totalread = 8190 Result = 0 FByte = 1 LByte = 0
DTCFTS4028I 8190 bytes sent on connection 2
DTCFTS8603I -----

```

Figure 88. A Sample of an FTP Server Trace (Part 2 of 4)


```

DTCFTS0028I Got note FSend response for #2, SendTurnCode = 0
DTCFTS4031I Xfread: totalread = 1692 Result = 0 FByte = 8191 LByte = 8190
DTCFTS4028I 1692 bytes sent on connection 2
DTCFTS8603I -----
DTCFTS0028I Got note FSend response for #2, SendTurnCode = 0
DTCFTS4031I Xfread: totalread = 0 Result = -12 FByte = 1693 LByte = 1692
DTCFTS7013I Calling CMS(ERASE CONN-2 FTPLIST A)
DTCFTS7012E TidyFile: FINIS returns 6
DTCFTS4017I Closing connection #2
DTCFTS4019I Completed CloseConnection
DTCFTS8603I -----
DTCFTS0023I Got note Connection state changed for #2, Receiving only
DTCFTS8603I -----
DTCFTS0023I Got note Connection state changed for #2, Nonexistent
DTCFTS2511I CloseCompleted on #2: OK
DTCFTS7013I Calling CMS(ERASE CONN-2 FTPLIST A)
DTCFTS4050I Send reply '250 List completed successfully.'
DTCFTS4056I DataReply: Setting CmdInProgress to CUNKNOWN on conn #2, was LIST
DTCFTS8603I -----
DTCFTS0026I Got note Data delivered for #0, 26 bytes
DTCFTS2560I 26 bytes arrived on conn #0
DTCFTS7022I Command Received on conn #0: PORT 9,117,222,18,70,202
DTCFTS4050I Send reply '200 Port request OK.'
DTCFTS4020I GetData(0)
DTCFTS4022I In GetData, TcpFReceive: Where = 1
DTCFTS8603I -----
DTCFTS0026I Got note Data delivered for #0, 17 bytes
DTCFTS2560I 17 bytes arrived on conn #0
DTCFTS7022I Command Received on conn #0: RETR TEST1.DATA
DTCFTS8095I FindMode minidisk TCPUSER1.191 re-accessed as 351 B
DTCFTS4024I OpenConnection(0982F92E,20,0975DE12,18122,30,TRUE
DTCFTS4052I ReinitDataConn(2)
DTCFTS4053I FtpFormat: A FtpMode: S FtpOptFormat: 0
DTCFTS4054I RecordFormat: V RecordLength: 65535
DTCFTS4058I AutomaticTranslation: ON
DTCFTS7009I SOpenfscb: name is: TEST1.DATA.B
DTCFTS7010I SOpenFscb: ESTATE returns: 0
DTCFTS6005I SOpenFscb: recfm: F lrecl: 80
DTCFTS4050I Send reply '150 Sending file 'TEST1.DATA' FIXrecfm 80'
DTCFTS4020I GetData(0)
DTCFTS4022I In GetData, TcpFReceive: Where = 1

```

Figure 89. A Sample of an FTP Server Trace (Part 3 of 4)

```

DTCFTS8603I -----
DTCFTS0023I Got note Connection state changed for #2, Open
DTCFTS2502I Allocating buffer of 131072 bytes
DTCFTS2503I Allocating RdFromDiskBuf of 8192 bytes
DTCFTS2508I Data connection 2 open for sending
DTCFTS4052I ReinitDataConn(2)
DTCFTS4053I FtpFormat: A FtpMode: S FtpOptFormat: 0
DTCFTS4054I RecordFormat: V RecordLength: 65535
DTCFTS4058I AutomaticTranslation: ON
DTCFTS4027I StartTransfer for 2:
DTCFTS4031I Xfread: totalread = 656 Result = 0 FByte = 1 LByte = 0
DTCFTS4028I 656 bytes sent on connection 2
DTCFTS8603I -----
DTCFTS0028I Got note FSend response for #2, SendTurnCode = 0
DTCFTS4031I Xfread: totalread = 0 Result = -12 FByte = 657 LByte = 656
DTCFTS7012E TidyFile: FINIS returns 0
DTCFTS4017I Closing connection #2
DTCFTS4019I Completed CloseConnection
DTCFTS8603I -----
DTCFTS0023I Got note Connection state changed for #2, Receiving only
DTCFTS8603I -----
DTCFTS0023I Got note Connection state changed for #2, Nonexistent
DTCFTS2511I CloseCompleted on #2: OK
DTCFTS4050I Send reply '250 Transfer completed successfully.'
DTCFTS4056I DataReply: Setting CmdInProgress to CUNKNOWN on conn #2, was RETR
DTCFTS8603I -----
DTCFTS0026I Got note Data delivered for #0, 6 bytes
DTCFTS2560I 6 bytes arrived on conn #0
DTCFTS7022I Command Received on conn #0: QUIT
DTCFTS4050I Send reply '221 Quit command received. Goodbye.'
DTCFTS4017I Closing connection #0
DTCFTS4019I Completed CloseConnection
DTCFTS4020I GetData(0)
DTCFTS4022I In GetData, TcpFReceive: Where = 1
DTCFTS8603I -----
DTCFTS0023I Got note Connection state changed for #0, Receiving only
DTCFTS8603I -----
DTCFTS0023I Got note Connection state changed for #0, Nonexistent
DTCFTS2511I CloseCompleted on #0: OK
DTCFTS8184I FreeMode B.

```

Figure 90. A Sample of an FTP Server Trace (Part 4 of 4)

Chapter 10. 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 z/VM 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 SMTPQUEUE 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 SMTPQUEUE command. [Figure 91 on page 137](#) 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 91. 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 16 on page 138 describes the SMTP commands that are helpful when interpreting SMTP trace output.

Table 16. 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.
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.

Table 16. SMTP Commands (continued)

Name	Command	Description
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.
STARTTLS	STARTTLS	This command causes the SMTP server to negotiate the use of a secure connection with the SMTP client that issued the command. If the SMTP server is configured for Secure SSL (using the TLS and TLSLABEL configuration statements), and if the negotiation between the client and server succeeds, all subsequent data will be encrypted and flow over a secure connection.
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 92 on page 140 shows the SMTP reply codes. The information shown in this figure 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 recognized	
Postel	{Page 35}

Figure 92. SMTP Reply Codes

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 93 on page 141 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 93 on page 141, 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 93. 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 94 on page 141 shows sample LOG information matching the sample trace shown in Figure 93 on page 141. 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 94. 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 95 on page 142 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 * * * * *
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 95. 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 96 on page 143 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 96. 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 97 on page 143](#) 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 97. A Sample of a Connection Activity Trace

Chapter 11. 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 98 on page 146](#) 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 17 on page 146)
X'20'+Cred-L	Authentication verifier data length
Data field	Data specific to the procedure called.

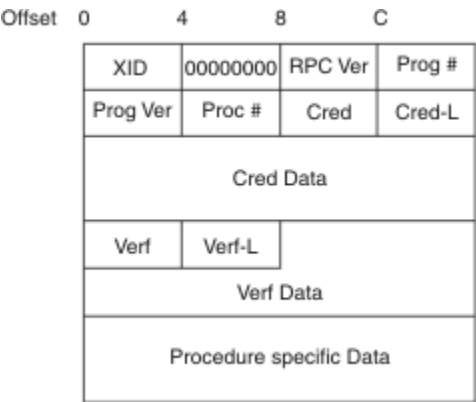


Figure 98. RPC Call Message Structure

Table 17 on page 146 describes the RPC credentials found in the Cred data field, shown in [Figure 98 on page 146](#).

Table 17. 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 99 on page 147](#) 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 17 on page 146)
X'10'	Authentication verifier data byte length
X'14'	Accept_stat

X'18'

Acc_stat dependent data.

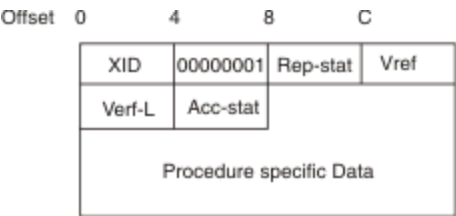


Figure 99. Structure of an RPC Accepted Reply Message

Acc_stat is a one word return code for NFS procedures that has a value described in Table 18 on page 147. 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 18 on page 147, data is not returned. For more information about acc_stat values, see RFC 1057.

Table 18. 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.
GARBAGE_ARGS	4	Procedure cannot decode parameters.

Rejected Reply Messages

Figure 100 on page 147 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.

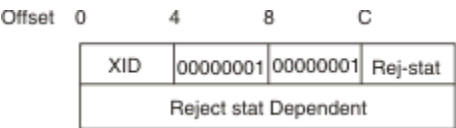


Figure 100. Structure of an RPC Rejected Reply Message

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 100 on page 147, 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 100 on page 147](#), contains a one word auth_stat value. [Table 19 on page 148](#) describes the auth_stat values. For more information about auth_stat values, see RFC 1057.

<i>Table 19. RPC Auth_stat Values</i>		
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 20 on page 148](#) describes Portmapper procedures.

<i>Table 20. Portmapper Procedures</i>		
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 12. Diagnosing MPRoute Problems

This chapter provides information and guidance to help you diagnose MPRoute problems.

For IPv4, MPRoute implements the OSPF protocol described in RFC 1583 (OSPF Version 2) and the RIP protocols described in RFC 1058 (RIP Version 1) and in RFC 1723 (RIP Version 2). For IPv6, MPRoute implements the IPv6 OSPF protocol described in RFC 2740 (OSPF for IPv6) and the IPv6 RIP protocol described in RFC 2080 (RIPng for IPv6).

MPRoute provides an alternative to the static TCP/IP GATEWAY definitions. When configured properly, the z/VM host running MPRoute becomes an active OSPF or RIP (or both) router in a TCP/IP network. The routing protocols are used to maintain the host routing table dynamically. For example, MPRoute can determine that a new route has been created, that a route is temporarily unavailable, or that a more efficient route exists.

MPRoute works best without static routes, and the use of static routes (defined through the GATEWAY TCP/IP configuration statement) is not recommended. Static routes might interfere with MPRoute's ability to discover a better route to the destination or to switch to another route if the destination becomes unreachable. For example, if you define a static host route through one interface and that interface becomes unreachable, MPRoute does not define a route to that same host through an alternative interface.

If you must define static routes, all static routes are considered to be of equal cost and will not be replaced by OSPF or RIP routes. Use extreme care when working with static routes and MPRoute. Set `IMPORT_STATIC_ROUTES = YES` on the `AS_Boundary_Routing` or `IPv6_AS_Boundary_Routing` configuration statement, or both. Alternatively, set `SEND_STATIC_ROUTES = YES` on the `RIP_Interface` or `IPv6_RIP_Interface` configuration statement, or both. These settings allow the static routes to be advertised to other routers.

Unlike static routes added through the GATEWAY statement, generated static routes can be replaced by dynamic routes learned by MPRoute. When a subnet mask is specified for IPv4 home addresses, the TCP/IP server automatically generates a direct static route to the subnet described by the IP address and mask. For IPv6 addresses, the TCP/IP server automatically generates a direct static route to the network described by the first 64 bits of the address.

MPROUTE must be defined correctly to TCP/IP. For detailed information about TCP/IP definitions, refer to the chapter on configuring MPRoute in [z/VM: TCP/IP Planning and Customization](#).

Categorizing MPRoute Problems

Problems with MPRoute are generally reported under one of the following categories:

- Abends
- MPRoute connection problems
- Routing failures.

These categories are described in the following sections.

Abends

An abend during MPRoute processing should result in messages and error-related information being sent to the MPRoute virtual machine's console. A dump of the error is needed unless the symptoms match a known problem.

MPRoute Connection Problems

MPRoute connection problems are reported when MPRoute is unable to connect to TCP/IP or to one of the ports required for OSPF or RIP communication. Generally, an inability to connect to TCP/IP is caused

by an error in the configuration or definitions in TCP/IP. An inability to connect to one of the required ports is generally caused by an error in the configuration or definitions in TCP/IP or by attempting to start MPRoute when MPRoute is already connected to the specified stack.

If MPRoute cannot communicate with the stack or is unable to initialize its required ports, it issues an error message describing the problem and terminates.

Routing Failures

Routing problems are usually the result of outages in a network and a lack of alternative routing paths available for recovery. Routing problems can also be the result of incorrect configurations in the channel-attached and network-attached routers as well as incorrect ARP entries. PING and TRACERTE commands to and from a z/VM host are useful diagnosis aids for problem determination. If a PING or TRACERTE command fails on a system where MPRoute is being used, a client is unable to get a positive response to a PING or TRACERTE command. Before doing any other problem determination, issue the NETSTAT GATE and SMSG *server_id* RTTABLE or SMSG *server_id* RT6TABLE commands on the local and remote hosts to get the routing table information for both the TCP/IP stack and MPRoute.

From the NETSTAT GATE outputs, determine which route is used to reach the destination and determine the route-active state. For IPv4, a routing table is searched in the following order, starting with the most specific to the least specific:

1. Host Routes
2. Subnet Routes
3. Network Routes
4. Supernet Routes
5. Default Routes

For IPv6, a routing table is searched in the following order, starting with the most specific to the least specific:

1. Host Routes
2. Prefix Routes
3. Default Routes

If there are no active routes available to reach the destination or if there are improperly configured channel-attached or network-attached routers along the routing path, the PING and TRACERTE commands will fail. To function correctly, PING requires active routes in both directions between the PING origin and the PING destination. If the routes are shown to be active at the local and remote hosts, the problem is most likely caused by a router along the routing path. Use the output from the TRACERTE command to locate the suspect router.

Documenting Routing Failures

The following documentation should be available for initial diagnosis of routing failures:

- The MPRoute virtual machine's console.
- Output from NETSTAT GATE.
- The file containing MPRoute's trace and debug information. For details, see [“MPRoute Traces and Debug Information” on page 151](#).
- Output from appropriate MPRoute SMSG commands as described in [“Using Privileged MPRoute SMSG Commands” on page 151](#).

Guidelines for Analyzing Routing Failures

When analyzing routing failures, follow these guidelines:

- Make sure that the address used in attempting to contact the remote host is a valid IP address.

- If the output from the NETSTAT GATE command does not show the expected results relative to the desired destination, do one or more of the following:
 - Make sure that the router(s) involved in providing information relative to this destination are operational and participating in the correct routing protocol.
 - Make sure that the physical connections involved in reaching the destination are active.
 - Use the MPRoute SMSG commands to determine whether anything in the configuration or current state of MPRoute has caused a route to the destination to be absent. See [“Using Privileged MPRoute SMSG Commands”](#) on page 151.
- Make sure routing is possible to and from the z/VM host. For most TCP/IP communications, two-way routing is required: the origin must have routes to reach the destination, and the destination must have routes to reach the origin. So even if the NETSTAT GATE command you issue at the origin shows correct routing, you must also issue the NETSTAT GATE command at the destination to verify that it can send replies back to the origin.

Using Privileged MPRoute SMSG Commands

The z/VM Special Message Facility (SMSG) command provides an interactive interface to the MPRoute virtual machine to perform privileged system administration tasks.

Privileged users are specified in the OBEY list of the TCP/IP server configuration file.

Note: Command responses are returned to the originator of the command through CP MSG commands.

For information about the MPRoute SMSG commands, see [Dynamic Server Operation in z/VM: TCP/IP Planning and Customization](#).

MPRoute Traces and Debug Information

MPRoute internal tracing and debugging can be started when MPRoute is started. Also, the SMSG command can be used to start, stop, or alter MPRoute's tracing and debugging after MPRoute has been started.

This section describes each of these methods.

Starting MPRoute Tracing and Debugging from the z/VM Console

If MPRoute is started from the command line (using the MPROUTE command), you can specify parameters to indicate the level of tracing or debugging you want:

–*tn* and –*6tn* (where *n* is a supported trace level)

These options specify the MPRoute external tracing levels, with –*tn* covering both MPRoute initialization and IPv4 routing protocols and –*6tn* covering IPv6 routing protocols. These options provide information about the operation of the routing application and can be used for many purposes, such as debugging a configuration, education on the operation of the routing application, verification of test cases, and so on. The following trace levels are supported:

- 1 = Informational messages
- 2 = Formatted packet trace

–*dn* and –*6dn* (where *n* is a supported debug level)

These options specify the MPRoute internal debugging levels, with –*dn* covering both MPRoute initialization and IPv4 routing protocols and –*6dn* covering IPv6 routing protocols. These options provide internal debugging information needed for debugging problems. The following levels are supported:

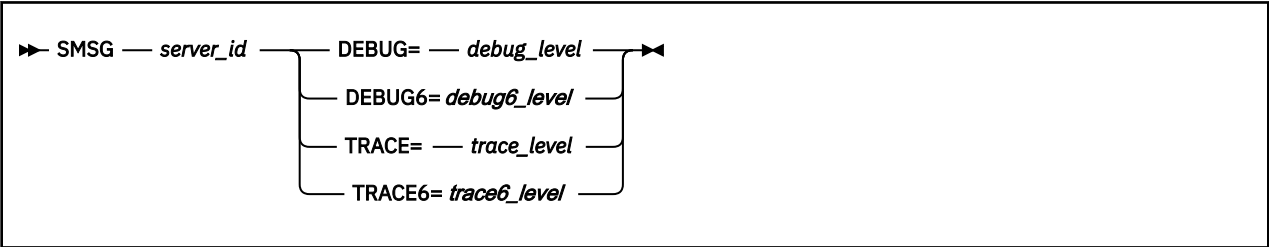
- 1 = Internal debugging messages
- 2 = Unformatted hexadecimal packet trace
- 3 = Function entry or exit trace

- 4 = Task add or run

Note:

1. The `-tn`, `-6tn`, `-dn`, and `-6dn` options affect MPRoute performance. As a result, you might have to increase the dead router interval on OSPF and IPv6 OSPF interfaces to prevent neighbor adjacencies from collapsing.
2. The trace and debug levels are cumulative: each level includes all lower levels. For example, `-t2` provides formatted packet trace and informational messages. You can enter more than one parameter by inserting a space after each parameter; for example, `mproute -t1 -d2`, which is the trace level most often requested by support.
3. You can specify parameters in mixed case.

Starting MPRoute Tracing and Debugging using the SMSG Command



Operands

server_id

Specifies the user ID of the MPRoute server virtual machine.

DEBUG=debug_level

Sets or changes the debug level for MPRoute initialization as well as IPv4 routing protocols. The following debug levels are available:

debug_level

Description

- 0** Turns debug messages off.
- 1** Provides internal debugging messages.
- 2** Provides unformatted hex packet tracing.
- 3** Provides function entry/exit trace.
- 4** Provides task add/run.

DEBUG6=debug6_level

Sets or changes the debug level for IPv6 routing protocols. The following debug levels are available:

debug6_level

Description

- 0** Turns debug messages off.
- 1** Provides internal debugging messages.
- 2** Provides unformatted hex packet tracing.

- 3 Provides function entry/exit trace.
- 4 Provides task add/run.

TRACE=trace_level

Sets or changes the trace level for MPRoute initialization as well as IPv4 routing protocols. The following trace levels are available:

trace_level**Description**

- 0 Turns MPRoute tracing off.
- 1 Provides all informational messages.
- 2 Provides formatted packet tracing.

TRACE6=trace6_level

Sets or changes the trace level for IPv6 routing protocols. The following trace levels are available:

trace6_level**Description**

- 0 Turns MPRoute tracing off.
- 1 Provides all informational messages.
- 2 Provides formatted packet tracing.

Usage Notes

- Use of MPRoute debugging and tracing affect MPRoute performance. As a result, you may have to increase the dead router interval on OSPF and IPv6 OSPF interfaces to prevent neighbor adjacencies from collapsing.
- The trace and debug levels are cumulative; each level includes all lower levels.

Examples

1. The following SMSG command passes a trace operand to an MPRoute server running in the MPROUTE1 virtual machine.

```
msg mproute1 trace=0
Ready;
07:02:30 * MSG FROM MPROUTE1 : MPRROUTE SMSG command accepted
```

Destination of MPRoute Trace and Debug Output

Output from MPRoute's tracing and debugging is written to the z/VM console.

Sample MPRoute Trace Output

The following is a sample MPRoute initialization and IPv4 routing protocol trace. Numbers in reverse type match the explanations that follow the sample.

```
DTCRUN1022I Console log will be sent to default owner ID:
TCPMAINT
DTCRUN1022I Console log will be sent to redefined owner ID:
TCPMNTM0
DTCRUN1096I STORAGE =
```

Diagnosing MPRoute Problems

```
32M
DTCRUN1027I Server will use TcpiUserid
TCIPM0
DTCRUN1011I Server started at 07:46:01 on 18 Nov 2010
(Thursday)
DTCRUN1011I Running server command:
MPROUTE
DTCRUN1011I Parameters in
use:
DTCRUN1011I -T1
1 MPROUTM0
starting
EZZ7845I Established affinity with
TCIPM0
EZZ7817I Using defined OSPF protocol
89
EZZ7817I Using defined OSPF protocol 89
EZZ7838I Using configuration file: MPROUTE CONFIG D1 dated 01/19/09 at
15:37
2 Processing interface from stack, address 10.0.4.1, name M0TOGLAN4, index 1, flags
463
EZZ7883I Processing interface from stack, address 10.0.0.13, name M0TOM1, index 2, flags
451
EZZ7883I Processing interface from stack, address 10.0.0.9, name M0TOM2, index 3, flags
451
EZZ7883I Processing interface from stack, address 10.0.0.5, name M0TOM3, index 4, flags
451
EZZ7883I Processing interface from stack, address 10.0.0.1, name M0TOM4, index 5, flags
451
EZZ7882I Processing static route from stack, destination 10.0.0.0, Mask 255.255.255.252, gateway
0.0.0.0
3 11/18 07:46:01 EZZ8059I Added network 10.0.0.0 with route via 10.0.0.1 on net 5 interface
M0TOM4
EZZ7882I Processing static route from stack, destination 10.0.0.4, Mask 255.255.255.252, gateway
0.0.0.0
11/18 07:46:01 EZZ8059I Added network 10.0.0.4 with route via 10.0.0.5 on net 4 interface
M0TOM3
EZZ7882I Processing static route from stack, destination 10.0.0.8, Mask 255.255.255.252, gateway
0.0.0.0
11/18 07:46:01 EZZ8059I Added network 10.0.0.8 with route via 10.0.0.9 on net 3 interface
M0TOM2
EZZ7882I Processing static route from stack, destination 10.0.0.12, Mask 255.255.255.252, gateway
0.0.0.0
11/18 07:46:01 EZZ8059I Added network 10.0.0.12 with route via 10.0.0.13 on net 2 interface
M0TOM1
EZZ7882I Processing static route from stack, destination 10.0.4.0, Mask 255.255.255.0, gateway
0.0.0.0
11/18 07:46:01 EZZ8059I Added network 10.0.4.0 with route via 10.0.4.1 on net 1 interface
M0TOGLAN4
EZZ7882I Processing static route from stack, destination e1:4::, prefixlen 64,
gateway ::
EZZ8023I The RIP routing protocol is
Enabled
EZZ8036I The IPv6 RIP routing protocol is
Enabled
EZZ7938I SPF Interface 10.0.1.1 (M0TOGLAN1) is not an IP interface, interface not
installed
EZZ8171I IPv4 OSPF is using assigned router ID 10.0.0.4 from M0TOM3
interface
EZZ7937I The IPv4 OSPF routing protocol is
Enabled
EZZ7937I The IPv6 OSPF routing protocol is
Disabled
11/18 07:46:01 EZZ8057I Added network 10.0.0.0 to interface 10.0.0.1 on net 5 interface
M0TOM4
11/18 07:46:01 EZZ7827I Adding stack route to 10.0.0.0, Mask 255.255.255.252 via 0.0.0.0, link M0TOM4, metric 1,
type 1
11/18 07:46:01 EZZ7879I Joining multicast group 224.0.0.9 on interface
10.0.0.1
11/18 07:46:01 EZZ8057I Added network 10.0.0.4 to interface 10.0.0.5 on net 4 interface
M0TOM3
11/18 07:46:01 EZZ7827I Adding stack route to 10.0.0.4, Mask 255.255.255.252 via 0.0.0.0, link M0TOM3, metric 1,
type 1
4 11/18 07:46:01 EZZ7910I Sending multicast, type 1, destination 224.0.0.5 net 4 interface
M0TOM3
5 11/18 07:46:01 EZZ7913I State change, interface 10.0.0.5, new state 16, event
1
.
.
.
EZZ7875I No IPv4 default route
installed
11/18 07:46:01 EZZ8067I Network 0 interface M0TOGLAN4 is
inactive
EZZ7875I No IPv6 default route
installed
EZZ7898I MPROUTM0 Initialization
Complete
11/18 07:46:01 EZZ8050I Updating BSD Route Params for link M0TOM4, MTU 32760, metric 1, subnet 255.255.255.252,
destination 0.0.0.0
11/18 07:46:01 EZZ8050I Updating BSD Route Params for link M0TOM3, MTU 32760, metric 1, subnet 255.255.255.252,
destination 0.0.0.0
11/18 07:46:01 EZZ8050I Updating BSD Route Params for link M0TOM2, MTU 32760, metric 1, subnet 255.255.255.252,
```

```

destination 0.0.0.0
11/18 07:46:01 EZZ8050I Updating BSD Route Parms for link M0TOM1, MTU 32760, metric 1, subnet 255.255.255.252,
destination 0.0.0.0
11/18 07:46:01 EZZ8050I Updating BSD Route Parms for link M0TOGLAN4, MTU 8192, metric 1, subnet 255.255.255.0,
destination 0.0.0.0
11/18 07:46:01 EZZ7934I Originating LS advertisement: typ 1 id 10.0.0.5 org
10.0.0.5
6 11/18 07:46:01 EZZ7908I Received packet type 1 from
10.0.0.6
11/18 07:46:01 EZZ7908I Received packet type 1 from
10.0.0.10
11/18 07:46:01 EZZ7908I Received packet type 1 from
10.0.0.14
11/18 07:46:01 EZZ7910I Sending multicast, type 1, destination 224.0.0.5 net 4 interface
M0TOM3
7 11/18 07:46:01 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.6, new state 4, event
1
8 11/18 07:46:01 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.6, new state 8, event
3
11/18 07:46:01 EZZ7910I Sending multicast, type 1, destination 224.0.0.5 net 3 interface
M0TOM2
11/18 07:46:01 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.10, new state 4, event
1
11/18 07:46:01 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.10, new state 8, event
3
11/18 07:46:01 EZZ7910I Sending multicast, type 1, destination 224.0.0.5 net 2 interface
M0TOM1
11/18 07:46:01 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.14, new state 4, event
1
11/18 07:46:01 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.14, new state 8, event
3
9 11/18 07:46:01 EZZ8011I send request to address
224.0.0.9
11/18 07:46:01 EZZ8015I sending packet to
224.0.0.9
11/18 07:46:02 EZZ8015I sending packet to
224.0.0.9
11/18 07:46:02 EZZ8021I sending RIP2 response to address 224.0.0.9 from 10.0.0.1 in 1 packets with 4
routes
11/18 07:46:02 EZZ8004I response received from host
10.0.0.2
11/18 07:46:02 EZZ8010I update route to net 10.0.0.2 at metric 1 hops via router
10.0.0.2
11/18 07:46:02 EZZ7827I Adding stack route to 10.0.0.2, Mask 255.255.255.255 via 0.0.0.0, link M0TOM4, metric 1,
type 129
11/18 07:46:02 EZZ8010I update route to net 10.0.3.0 at metric 2 hops via router
10.0.0.2
11/18 07:46:02 EZZ7827I Adding stack route to 10.0.3.0, Mask 255.255.255.0 via 10.0.0.2, link M0TOM4, metric 2,
type 130
EZZ7882I Processing static route from stack, destination e1:4::, prefixlen 64,
gateway ::
11/18 07:46:02 EZZ7908I Received packet type 1 from
10.0.0.14
11/18 07:46:02 EZZ7908I Received packet type 2 from
10.0.0.6
10 11/18 07:46:02 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.6, new state 16, event
14
11 11/18 07:46:02 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.6, new state 32, event 5
12 11/18 07:46:02 EZZ7910I Sending multicast, type 2, destination 224.0.0.5 net 4 interface M0TOM3
11/18 07:46:02 EZZ7908I Received packet type 2 from
10.0.0.10
11/18 07:46:02 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.10, new state 16, event
14
11/18 07:46:02 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.10, new state 32, event
5
11/18 07:46:02 EZZ7910I Sending multicast, type 2, destination 224.0.0.5 net 3 interface
M0TOM2
11/18 07:46:02 EZZ7908I Received packet type 2 from
10.0.0.14
11/18 07:46:02 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.14, new state 16, event
14
11/18 07:46:02 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.14, new state 32, event
5
11/18 07:46:02 EZZ7910I Sending multicast, type 2, destination 224.0.0.5 net 2 interface
M0TOM1
13 11/18 07:46:02 EZZ7908I Received packet type 2 from
10.0.0.6
14 11/18 07:46:02 EZZ7910I Sending multicast, type 3, destination 224.0.0.5 net 4 interface
M0TOM3
11/18 07:46:02 EZZ7908I Received packet type 2 from
10.0.0.10
11/18 07:46:02 EZZ7910I Sending multicast, type 3, destination 224.0.0.5 net 3 interface
M0TOM2
11/18 07:46:02 EZZ7908I Received packet type 2 from
10.0.0.14
11/18 07:46:02 EZZ7910I Sending multicast, type 3, destination 224.0.0.5 net 2 interface
M0TOM1
15 11/18 07:46:02 EZZ7908I Received packet type 4 from
10.0.0.6
16 11/18 07:46:02 EZZ7928I from 10.0.0.6, new LS advertisement: typ 1 id 10.0.0.5 org
10.0.0.5
11/18 07:46:02 EZZ7927I from 10.0.0.5, self update: typ 1 id 10.0.0.5 org
10.0.0.5

```

Diagnosing MPRoute Problems

```
11/18 07:46:02 EZZ7928I from 10.0.0.6, new LS advertisement: typ 1 id 10.0.0.17 org
10.0.0.17
11/18 07:46:02 EZZ7928I from 10.0.0.6, new LS advertisement: typ 1 id 10.0.0.18 org
10.0.0.18
11/18 07:46:02 EZZ7928I from 10.0.0.6, new LS advertisement: typ 1 id 10.0.0.22 org
10.0.0.22
11/18 07:46:02 EZZ7928I from 10.0.0.6, new LS advertisement: typ 2 id 10.0.2.1 org
10.0.0.22
11/18 07:46:02 EZZ7928I from 10.0.0.6, new LS advertisement: typ 5 id 10.0.0.2 org
10.0.0.5
11/18 07:46:02 EZZ7927I from 10.0.0.5, self update: typ 5 id 10.0.0.2 org
10.0.0.5
11/18 07:46:02 EZZ7928I from 10.0.0.6, new LS advertisement: typ 5 id 10.0.3.0 org
10.0.0.5
11/18 07:46:02 EZZ7927I from 10.0.0.5, self update: typ 5 id 10.0.3.0 org
10.0.0.5
11/18 07:46:02 EZZ7934I Originating LS advertisement: typ 1 id 10.0.0.5 org
10.0.0.5
11/18 07:46:02 EZZ7934I Originating LS advertisement: typ 5 id 10.0.0.2 org
10.0.0.5
11/18 07:46:02 EZZ7934I Originating LS advertisement: typ 5 id 10.0.3.0 org
10.0.0.5
17 11/18 07:46:02 EZZ7910I Sending multicast, type 4, destination 224.0.0.5 net 4 interface
MOTOM3
11/18 07:46:02 EZZ7910I Sending multicast, type 4, destination 224.0.0.5 net 3 interface
MOTOM2
11/18 07:46:02 EZZ7910I Sending multicast, type 4, destination 224.0.0.5 net 2 interface
MOTOM1
11/18 07:46:02 EZZ7908I Received packet type 4 from
10.0.0.10
18 11/18 07:46:02 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.6, new state 128, event
6
11/18 07:46:02 EZZ7910I Sending multicast, type 2, destination 224.0.0.5 net 4 interface
MOTOM3
11/18 07:46:02 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.10, new state 128, event
6
11/18 07:46:02 EZZ7910I Sending multicast, type 2, destination 224.0.0.5 net 3 interface
MOTOM2
11/18 07:46:02 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.14, new state 128, event
6
11/18 07:46:02 EZZ7910I Sending multicast, type 2, destination 224.0.0.5 net 2 interface
MOTOM1
11/18 07:46:02 EZZ7908I Received packet type 4 from
10.0.0.14
11/18 07:46:02 EZZ7926I from 10.0.0.10, old LS advertisement: typ 1 id 10.0.0.5 org
10.0.0.5
11/18 07:46:02 EZZ7910I Sending multicast, type 4, destination 224.0.0.5 net 3 interface
MOTOM2
11/18 07:46:02 EZZ7926I from 10.0.0.10, old LS advertisement: typ 5 id 10.0.0.2 org
10.0.0.5
11/18 07:46:02 EZZ7910I Sending multicast, type 4, destination 224.0.0.5 net 3 interface
MOTOM2
11/18 07:46:02 EZZ7926I from 10.0.0.10, old LS advertisement: typ 5 id 10.0.3.0 org
10.0.0.5
11/18 07:46:02 EZZ7910I Sending multicast, type 4, destination 224.0.0.5 net 3 interface
MOTOM2
11/18 07:46:02 EZZ7932I IPv4 OSPF LS acknowledgement sent directly to neighbor
10.0.0.10
11/18 07:46:02 EZZ7910I Sending multicast, type 5, destination 224.0.0.5 net 3 interface
MOTOM2
11/18 07:46:02 EZZ7926I from 10.0.0.14, old LS advertisement: typ 1 id 10.0.0.5 org
10.0.0.5
11/18 07:46:02 EZZ7910I Sending multicast, type 4, destination 224.0.0.5 net 2 interface
MOTOM1
11/18 07:46:02 EZZ7926I from 10.0.0.14, old LS advertisement: typ 5 id 10.0.0.2 org
10.0.0.5
11/18 07:46:02 EZZ7910I Sending multicast, type 4, destination 224.0.0.5 net 2 interface
MOTOM1
11/18 07:46:02 EZZ7926I from 10.0.0.14, old LS advertisement: typ 5 id 10.0.3.0 org
10.0.0.5
11/18 07:46:02 EZZ7910I Sending multicast, type 4, destination 224.0.0.5 net 2 interface
MOTOM1
11/18 07:46:02 EZZ7932I IPv4 OSPF LS acknowledgement sent directly to neighbor
10.0.0.14
11/18 07:46:02 EZZ7910I Sending multicast, type 5, destination 224.0.0.5 net 2 interface
MOTOM1
11/18 07:46:02 EZZ7908I Received packet type 4 from
10.0.0.14
19 11/18 07:46:02 EZZ7908I Received packet type 5 from
10.0.0.6
11/18 07:46:02 EZZ7908I Received packet type 5 from
10.0.0.10
11/18 07:46:02 EZZ7908I Received packet type 5 from
10.0.0.10
11/18 07:46:02 EZZ7908I Received packet type 5 from
10.0.0.14
11/18 07:46:02 EZZ7939I Duplicate LS acknowledgement received from IPv4 neighbor
10.0.0.14
11/18 07:46:02 EZZ7908I Received packet type 5 from
10.0.0.14
11/18 07:46:02 EZZ7939I Duplicate LS acknowledgement received from IPv4 neighbor
10.0.0.14
11/18 07:46:02 EZZ7908I Received packet type 5 from
```

```

10.0.0.14
11/18 07:46:02 EZZ7939I Duplicate LS acknowledgement received from IPv4 neighbor
10.0.0.14
20 11/18 07:46:02 EZZ7910I Sending multicast, type 5, destination 224.0.0.5 net 4 interface
MOTOM3
11/18 07:46:02 EZZ7910I Sending multicast, type 1, destination 224.0.0.5 net 3 interface
MOTOM2
21 11/18 07:46:02 EZZ7949I Dijkstra calculation performed, on 1 IPv4
area(s)
11/18 07:46:02 EZZ7935I New MPROUTM0 route to destination Net 10.0.0.4, type Dir cost
1
11/18 07:46:02 EZZ7935I New MPROUTM0 route to destination Net 10.0.0.8, type Dir cost
1
11/18 07:46:02 EZZ7935I New MPROUTM0 route to destination Net 10.0.0.12, type Dir cost
1
22 11/18 07:46:02 EZZ7827I Adding stack route to 10.0.4.0, Mask 255.255.255.0 via 0.0.0.0, link M0TOGLAN4, metric 1,
type 1
11/18 07:46:02 EZZ7801I Deleting stack route to 10.0.4.0, Mask 255.255.255.0 via 0.0.0.0, link M0TOGLAN4, metric 1,
type 1
.
.
23 11/18 07:46:57 DTCMPR7895I Processing SMSG command from TCPMNTM0 - OSPF LIST
INTERFACES
11/18 07:46:57 EZZ7895I Processing console command -
OSPF,LIST,INTERFACES
11/18 07:46:57 EZZ7809I EZZ7833I INTERFACE
CONFIGURATION

11/18 07:46:57 EZZ7809I IP ADDRESS          AREA          COST RTRNS TRDLY PRI HELLO  DEAD
DB_EX

11/18 07:46:57 EZZ7809I 10.0.0.5          1.1.1.1          1    5    1    1    10    40
40
11/18 07:46:57 EZZ7809I 10.0.0.9          1.1.1.1          1    5    1    1    10    40
40
11/18 07:46:57 EZZ7809I 10.0.0.13         1.1.1.1          1    5    1    1    10    40
40
11/18 07:46:57 EZZ7809I 10.0.4.1          1.1.1.1          1    5    1    1    10    40
40
11/18 07:46:57 EZZ7809I 10.0.1.1          1.1.1.1          1    5    1    1    10    40
40
11/18 07:46:57
EZZ7809I
11/18 07:46:57 EZZ7809I Demand circuit
parameters

11/18 07:46:57 EZZ7809I IP address      DoNotAge      Hello Suppression      Poll
Interval

11/18 07:46:57 EZZ7809I 10.0.0.5      Off           Allow
60
11/18 07:46:57 EZZ7809I 10.0.0.9      Off           Allow
60
11/18 07:46:57 EZZ7809I 10.0.0.13     Off           Allow
60
11/18 07:46:57 EZZ7809I 10.0.4.1      Off           N/A
N/A
11/18 07:46:57 EZZ7809I 10.0.1.1      Off           Allow
60
11/18 07:46:58 EZZ7910I Sending multicast, type 1, destination 224.0.0.5 net 2 interface
MOTOM1
11/18 07:46:58 EZZ7908I Received packet type 1 from
10.0.0.10
.
.
24 11/18 07:47:18 DTCMPR7895I Processing SMSG command from TCPMNTM0 -
TRACE=2
11/18 07:47:18 EZZ7895I Processing console command -
TRACE=2
.
.
25 11/18 07:47:37 DTCMPR7895I Processing SMSG command from TCPMNTM0 -
TRACE6=2
11/18 07:47:38 EZZ7895I Processing console command -
TRACE6=2
11/18 07:47:38 EZZ7910I Sending multicast, type 1, destination 224.0.0.5 net 2 interface
MOTOM1
26 11/18 07:47:38 EZZ7876I -- OSPF Packet Sent ----- Type:
Hello
11/18 07:47:38 EZZ7878I OSPF Version: 2          Packet Length:
48
11/18 07:47:38 EZZ7878I Router ID: 10.0.0.5      Area:
1.1.1.1
11/18 07:47:38 EZZ7878I Checksum: e586          Auth Type:
0
11/18 07:47:38 EZZ7878I Hello_Interval: 10      Network mask:

```

Diagnosing MPRoute Problems

```
255.255.255.252
11/18 07:47:38 EZZ7878I Options:
E
11/18 07:47:38 EZZ7878I Router_Priority: 1          Dead_Router_Interval:
40
11/18 07:47:38 EZZ7878I Backup DR: 0.0.0.0          Designated Router:
0.0.0.0
11/18 07:47:38 EZZ7878I Neighbor:
10.0.0.17
11/18 07:47:38 EZZ7877I -- OSPF Packet Received -- Type:
Hello
11/18 07:47:38 EZZ7878I OSPF Version: 2            Packet Length:
48
11/18 07:47:38 EZZ7878I Router ID: 10.0.0.22        Area:
1.1.1.1
11/18 07:47:38 EZZ7878I Checksum: e581              Auth Type:
0
11/18 07:47:38 EZZ7878I Hello_Interval: 10          Network mask:
255.255.255.252
11/18 07:47:38 EZZ7878I Options:
E
11/18 07:47:38 EZZ7878I Router_Priority: 1          Dead_Router_Interval:
40
11/18 07:47:38 EZZ7878I Backup DR: 0.0.0.0          Designated Router:
0.0.0.0
11/18 07:47:38 EZZ7878I Neighbor:
10.0.0.5
11/18 07:47:38 EZZ7908I Received packet type 1 from
10.0.0.10
:
11/18 07:47:47 -- RIP Packet Sent ----- Type: Response
(V2)
11/18 07:47:47 Destination_Addr: 10.0.0.4          metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.252        Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.0.8          metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.252        Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.0.12         metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.252        Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.4.0          metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.0          Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.0.2          metric:
16
11/18 07:47:47 Subnet Mask: 255.255.255.255        Next Hop:
10.0.0.2
11/18 07:47:47 Destination_Addr: 10.0.3.0          metric:
16
11/18 07:47:47 Subnet Mask: 255.255.255.0          Next Hop:
10.0.0.2
11/18 07:47:47 Destination_Addr: 10.0.0.6          metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.255        Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.0.10         metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.255        Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.0.14         metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.255        Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.0.18         metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.255        Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.0.22         metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.255        Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.0.17         metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.255        Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.0.21         metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.255        Next Hop:
0.0.0.0
11/18 07:47:47 Destination_Addr: 10.0.2.0          metric:
1
11/18 07:47:47 Subnet Mask: 255.255.255.0          Next Hop:
0.0.0.0
11/18 07:47:47 EZZ8021I sending RIP2 response to address 224.0.0.9 from 10.0.0.1 in 1 packets with 14
routes
27 11/18 07:47:47 -- RIP Packet Received -- Type: Response
```



```
(V2)
11/18 07:47:47 Destination_Addr: 10.0.0.4      metric:
16
11/18 07:47:47 Subnet Mask: 255.255.255.252    Next Hop:
10.0.0.1
11/18 07:47:47 Destination_Addr: 10.0.0.8      metric:
16
11/18 07:47:47 Subnet Mask: 255.255.255.252    Next Hop:
10.0.0.1
11/18 07:47:47 Destination_Addr: 10.0.0.12     metric:
16
11/18 07:47:47 Subnet Mask: 255.255.255.252    Next Hop:
10.0.0.1
11/18 07:47:47 Destination_Addr: 10.0.4.0      metric:
16
11/18 07:47:47 Subnet Mask: 255.255.255.0      Next Hop:
10.0.0.1
11/18 07:47:47 Destination_Addr: 10.0.2.0      metric:
16
11/18 07:47:47 Subnet Mask: 255.255.255.0      Next Hop:
10.0.0.1
11/18 07:47:47 EZZ8004I response received from host
10.0.0.2
11/18 07:47:47 EZZ8010I update route to net 10.0.0.2 at metric 1 hops via router
10.0.0.2
.
.
28 11/18 07:48:34 DTCMPR7895I Processing MSG command from TCPMNTM0 -
TRACE=1
11/18 07:48:34 EZZ7895I Processing console command -
TRACE=1
.
.
29 11/18 07:50:11 EZZ7862I Received update interface M0TOM3
30 11/18 07:50:11 EZZ8061I Deleted net 10.0.0.4 route via 10.0.0.5 net 3 interface M0TOM3
11/18 07:50:11 EZZ7864I Deleting all stack routes to 10.0.0.4, Mask 255.255.255.252
31 11/18 07:50:11 EZZ7919I State change, IPv4 OSPF neighbor 10.0.0.6, new state 1, event 11
EZZ7921I OSPF adjacency failure, neighbor 10.0.0.6, old state 128, new state 1, event 11
11/18 07:50:11 EZZ7879I Leaving multicast group 224.0.0.5 on interface 10.0.0.5
32 11/18 07:50:11 EZZ7913I State change, interface 10.0.0.5, new state 1, event 7
11/18 07:50:11 EZZ7934I Originating LS advertisement: typ 1 id 10.0.0.5 org 10.0.0.5
11/18 07:50:12 EZZ7949I Dijkstra calculation performed, on 1 IPv4 area(s)
11/18 07:50:12 EZZ7943I Destination Net 10.0.0.6 now unreachable
11/18 07:50:12 EZZ7864I Deleting all stack routes to 10.0.0.6, Mask 255.255.255.255
11/18 07:50:12 EZZ7943I Destination Net 10.0.0.18 now unreachable
11/18 07:50:12 EZZ7864I Deleting all stack routes to 10.0.0.18, Mask 255.255.255.255
11/18 07:50:12 EZZ7943I Destination Net 10.0.0.22 now unreachable
11/18 07:50:12 EZZ7864I Deleting all stack routes to 10.0.0.22, Mask 255.255.255.255
11/18 07:50:12 EZZ7943I Destination Net 10.0.0.17 now unreachable
11/18 07:50:12 EZZ7864I Deleting all stack routes to 10.0.0.17, Mask 255.255.255.255
11/18 07:50:12 EZZ7943I Destination Net 10.0.0.5 now unreachable
11/18 07:50:12 EZZ7943I Destination Net 10.0.0.21 now unreachable
11/18 07:50:12 EZZ7864I Deleting all stack routes to 10.0.0.21, Mask 255.255.255.255
11/18 07:50:12 EZZ7943I Destination Net 10.0.2.0 now unreachable
11 /18 07:50:12 EZZ7864I Deleting all stack routes to 10.0.2.0, Mask 255.255.255.0
```

The explanations are:

1. MPRoute initializing (trace level 1 was specified at startup).
2. MPROUTE learns of TCP/IP stack IPv4 interfaces.
3. Direct routes are added for each TCP/IP stack IPv4 interface.
4. OSPF Hello packet sent out OSPF interface.
5. OSPF interface transitions to state "point-to-point."
6. OSPF Hello packet received from OSPF neighbor.
7. OSPF neighbor transitions to state "Init."
8. OSPF neighbor transitions to state "2-Way."
9. RIP requests and responses begin being sent out on the RIP interface.
10. OSPF neighbor transitions to state "ExStart."
11. OSPF neighbor transitions to state "Exchange."
12. OSPF Database Description packet sent out on the OSPF interface.
13. OSPF Database Description received from an OSPF neighbor.
14. OSPF Link State Request packet sent out on the OSPF interface.

15. OSPF Link State Update packet received from an OSPF neighbor.
16. Link State Advertisements from received Update packet are processed.
17. OSPF Link State Update packet sent out on the OSPF interface.
18. OSPF neighbor transitions to state "Full."
19. OSPF Link State Acknowledgment packet received from OSPF neighbor.
20. OSPF Link State Acknowledgment packet sent out on the OSPF interface.
21. OSPF Dijkstra calculation is performed.
22. Learned route is added to TCP/IP stack IPv4 route table.
23. Request received to display OSPF Interface configuration information.
24. Request received to change IPv4 tracing level to 2 (adds formatted packets).
25. Request received to change IPv6 tracing level to 2 (adds formatted packets).
26. Formatted OSPF packet.
27. Formatted RIP packet.
28. Request received to change tracing level back to 1.
29. MPRROUTE learns of stopped TCP/IP IPv4 interface.
30. Routes over stopped interface are deleted.
31. Neighbor over stopped interface transitions to state "Down."
32. Stopped interface transitions to state "Down."

The following is a sample MPRRoute initialization and IPv6 routing protocol trace. Numbers in reverse type match the explanations that follow the sample.

```
DTCRUN1022I Console log will be sent to redefined owner ID:
TCPMNTM6
DTCRUN1096I STORAGE =
32M
DTCRUN1027I Server will use TcpipUserid
TCPIPM6
DTCRUN1011I Server started at 10:47:51 on 18 Nov 2010
(Thursday)
DTCRUN1011I Running server command:
MPROUTE
DTCRUN1011I Parameters in
use:
DTCRUN1011I
-6T1
EZZ7800I MPROUTM6
starting
EZZ7845I Established affinity with
TCPIPM6
EZZ7817I Using defined OSPF protocol
89
EZZ7817I Using defined OSPF protocol
89
EZZ7838I Using configuration file: MPRROUTE CONFIG D1 dated 03/04/08 at
14:48
EZZ7882I Processing static route from stack, destination e1:4::, prefixlen 64,
gateway ::
EZZ7882I Processing static route from stack, destination e1:5::, prefixlen 64,
gateway ::
EZZ8023I The RIP routing protocol is
Disabled
EZZ8036I The IPv6 RIP routing protocol is
Enabled
EZZ7937I The IPv4 OSPF routing protocol is
Disabled
EZZ7937I The IPv6 OSPF routing protocol is
Disabled
EZZ7875I No IPv6 default route
installed
EZZ7898I MPROUTM6 Initialization
Complete
11/18 10:47:52 EZZ7863I Received add route to
e1:4::
EZZ7882I Processing static route from stack, destination e1:4::, prefixlen 64,
gateway ::
11/18 10:47:52 EZZ7863I Received add route to
e1:5::
EZZ7882I Processing static route from stack, destination e1:5::, prefixlen 64,
gateway ::
11/18 10:47:52 EZZ7862I Received add interface
```

```

M6TOGLAN4
2 11/18 10:47:52 EZZ8057I Added network e1:4:: to interface fe80::209:5700:160:d3 on net 1 interface
M6TOGLAN4
11/18 10:47:52 EZZ7827I Adding stack route to e1:4::, prefixlen 64 via ::, link M6TOGLAN4, metric 1, type
1
11/18 10:47:52 EZZ7879I Joining multicast group ff02::9 on interface
M6TOGLAN4
11/18 10:47:52 EZZ7862I Received update interface
M6TOGLAN4
3 11/18 10:47:52 EZZ8011I send request to address
ff02::9
11/18 10:47:52 EZZ8015I sending packet to
ff02::9
4 11/18 10:47:52 EZZ7863I Received add route to
e1:4::
11/18 10:47:52 EZZ8077I Ignoring replaceable static route to e1:4::, prefixlen , using :: - Dynamic routes already
active
11/18 10:47:52 EZZ7862I Received add interface
M6TOGLAN5
11/18 10:47:52 EZZ8057I Added network e1:5:: to interface fe80::209:5700:160:d4 on net 2 interface
M6TOGLAN5
11/18 10:47:52 EZZ7827I Adding stack route to e1:5::, prefixlen 64 via ::, link M6TOGLAN5, metric 1, type
1
11/18 10:47:52 EZZ7879I Joining multicast group ff02::9 on interface
M6TOGLAN5
11/18 10:47:52 EZZ7862I Received update interface
M6TOGLAN5
11/18 10:47:52 EZZ7863I Received add route to
e1:5::
11/18 10:47:52 EZZ8077I Ignoring replaceable static route to e1:5::, prefixlen , using :: - Dynamic routes already
active
11/18 10:47:52 EZZ8004I response received from host
fe80::209:5700:160:44
11/18 10:47:52 EZZ8004I response received from host
fe80::209:5700:160:58
11/18 10:47:52 EZZ8011I send request to address
ff02::9
11/18 10:47:52 EZZ8015I sending packet to
ff02::9
11/18 10:47:52 EZZ8015I sending packet to
ff02::9
11/18 10:47:52 EZZ8021I sending IPv6RIP response to address ff02::9 from fe80::209:5700:160:d4 in 1 packets with 1
routes
11/18 10:47:52 EZZ8015I sending packet to
ff02::9
11/18 10:47:52 EZZ8021I sending IPv6RIP response to address ff02::9 from fe80::209:5700:160:d3 in 1 packets with 1
routes
11/18 10:47:52 EZZ8004I response received from host
fe80::209:5700:160:4f
11/18 10:47:52 EZZ8004I response received from host
fe80::209:5700:160:48
11/18 10:47:53 EZZ7981I Received add address e1:4::4:6 to interface
M6TOGLAN4
11/18 10:47:53 EZZ8057I Added network e1:4::4:6 to interface fe80::209:5700:160:d3 on net 1 interface
M6TOGLAN4
11/18 10:47:53 EZZ7981I Received add address e1:5::5:6 to interface
M6TOGLAN5
11/18 10:47:53 EZZ8057I Added network e1:5::5:6 to interface fe80::209:5700:160:d4 on net 2 interface
M6TOGLAN5
:
:
11/18 10:48:57 EZZ8021I sending IPv6RIP response to address ff02::9 from fe80::209:5700:160:d4 in 1 packets with 4
routes
5 11/18 10:48:57 DTCMPR7895I Processing SMSG command from TCPMNTM6 -
TRACE6=2
6 11/18 10:48:58 -- IPv6 RIP Packet Received (M6TOGLAN5) -- Type:
Response
11/18 10:48:58 Destination_Addr:
e1:4::
11/18 10:48:58 Prefix Length: 64 metric:
1
11/18 10:48:58 Destination_Addr:
e1:6::
11/18 10:48:58 Prefix Length: 64 metric:
2
11/18 10:48:58 Destination_Addr:
e1:8::
11/18 10:48:58 Prefix Length: 64 metric:
2
11/18 10:48:58 Destination_Addr:
e1:7::
11/18 10:48:58 Prefix Length: 64 metric:
2
11/18 10:48:58 EZZ8004I response received from host
fe80::209:5700:160:48
11/18 10:49:06 -- IPv6 RIP Packet Received (M6TOGLAN4) -- Type:
Response
11/18 10:49:06 Destination_Addr:
e1:5::
11/18 10:49:06 Prefix Length: 64 metric:
16
11/18 10:49:06 Destination_Addr:

```

Diagnosing MPRoute Problems

```
e1:6::
11/18 10:49:06      Prefix Length: 64  metric:
16
11/18 10:49:06      Destination_Addr:
e1:8::
11/18 10:49:06      Prefix Length: 64  metric:
16
11/18 10:49:06      Destination_Addr:
e1:7::
11/18 10:49:06      Prefix Length: 64  metric:
16
.
.
.
11/18 10:49:57 EZZ8015I sending packet to
ff02::9
11/18 10:49:57 -- IPv6 RIP Packet Sent (M6TOGLAN5) -- Type:
Response
11/18 10:49:57      Destination_Addr:
e1:4::
11/18 10:49:57      Prefix Length: 64  metric:
1
11/18 10:49:57      Destination_Addr:
e1:6::
11/18 10:49:57      Prefix Length: 64  metric:
2
11/18 10:49:57      Destination_Addr:
e1:8::
11/18 10:49:57      Prefix Length: 64  metric:
2
11/18 10:49:57      Destination_Addr:
e1:7::
11/18 10:49:57      Prefix Length: 64  metric:
2
11/18 10:49:57 EZZ8021I sending IPv6RIP response to address ff02::9 from fe80::209:5700:160:d4 in 1 packets with 4
routes
```

The explanations are:

1. MPROUTE learns of TCP/IP stack IPv6 interface addresses. Note that each home address on an IPv6 interface is described separately; MPROUTE uses the interface name to assign addresses to a specific interface.
2. Direct routes are added for each non-link-local TCP/IP stack IPv6 home address. When an interface's home address is needed in a message, its link-local address is used.
3. IPv6 RIP requests and responses begin being sent out IPv6 RIP interface. Note the use of link-local address when the interface is being identified by address only.
4. IPv6 RIP Response received and associated routes added to IPv6 route table. Note that the source address is always link-local.
5. Request received to change IPv6 tracing level to 2 (adds formatted packets). The operator command to set the tracing level appears in the IPv4 trace, because modify commands run on the IPv4 thread.
6. Formatted IPv6 RIP packet.

Chapter 13. SSL Server Diagnosis

This chapter describes some of the debugging facilities for the Secure Socket Layer (SSL) server. SSL trace facilities exist that can be useful in identifying the cause of SSL server problems. This section discusses these trace requests and how they can be started and stopped. Included are descriptions of traces as well as examples of how these traces are invoked.

The Secure Socket Layer (SSL) server provides the processing that allows secure (encrypted) communication between a remote client and a VM TCP/IP server (in this context known as the application server). The application server must be listening on a port identified as secure by the installation, and the remote client must support the SSL protocol. Transport Layer Security (TLS) is the Internet Standards protocol based on SSL and is described in RFC 2246.

[Figure 101 on page 163](#) expresses the viewpoint of the client and the server that there is a connection between them.

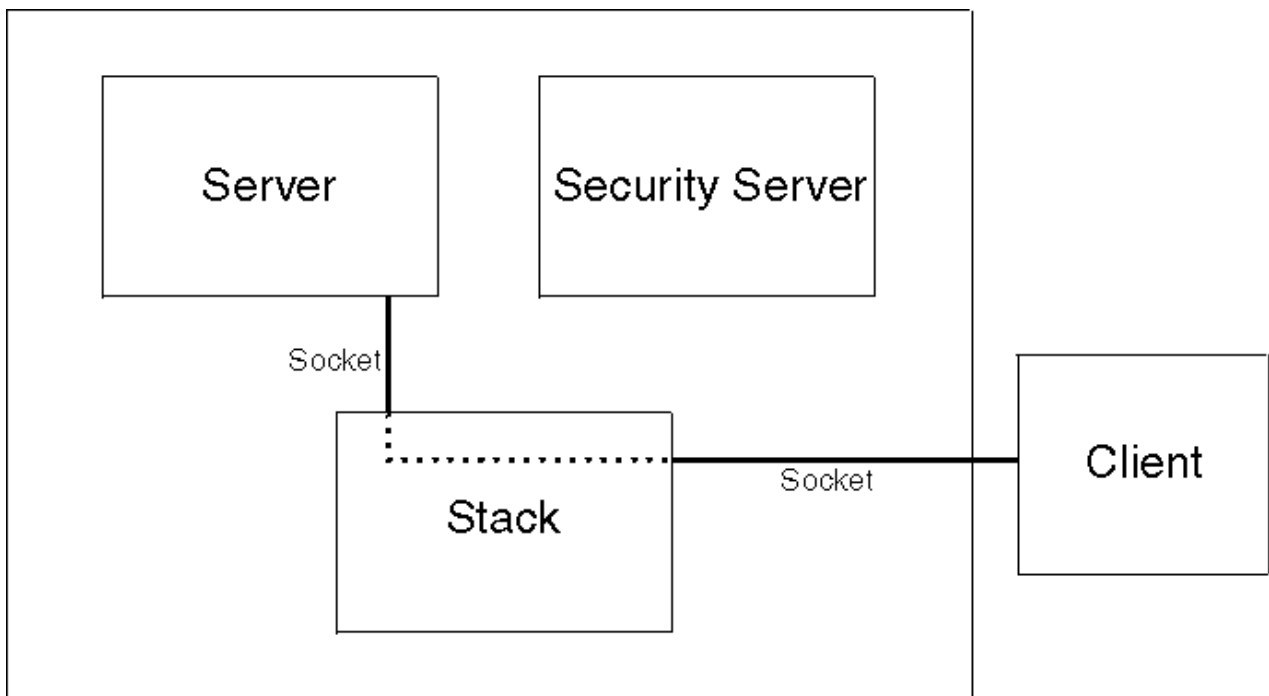


Figure 101. SSL Client and Server Environment

The reality is that the client has a connection with the SSL server and the SSL server has a connection with the server as illustrated in [Figure 102 on page 164](#):

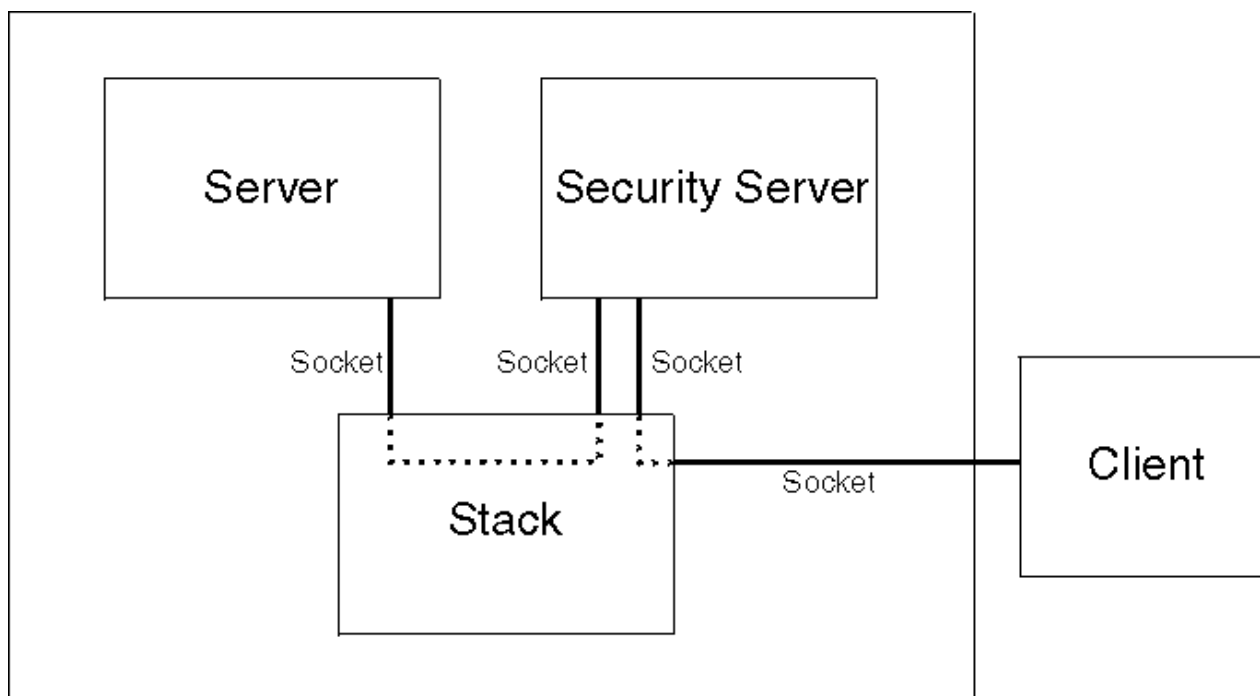


Figure 102. TCP/IP Stack View of connection

SSL component Flow

The following diagram illustrates how the SSL server and stack work together to provide SSL processing on behalf of a secure server:

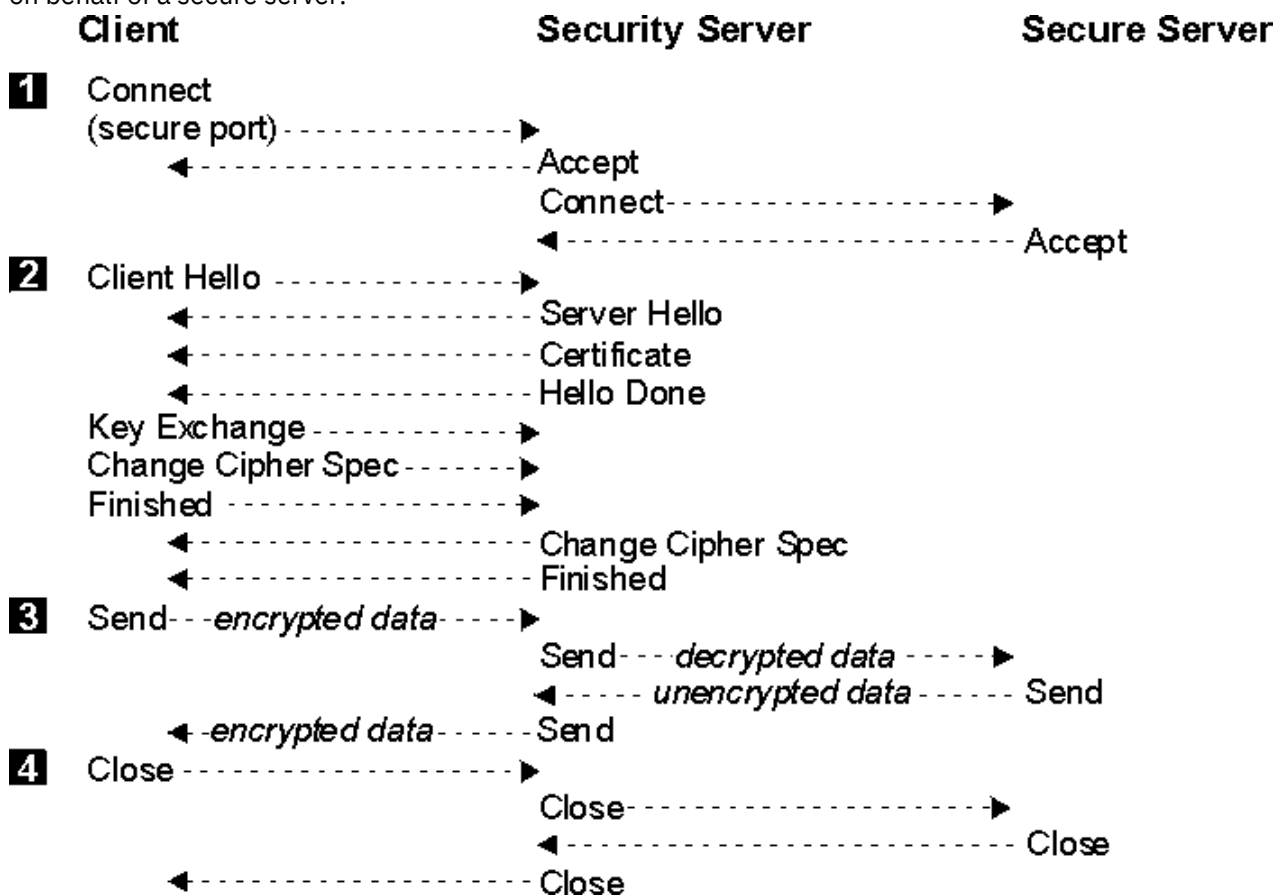


Figure 103. SSL processing flow

An SSL session consists of the following general processing steps:

1 Connect

The SSL session is maintained as two separate connections: the connection from the remote client to the SSL server, and the connection from the SSL server to the application server. The intervention of the SSL server is transparent to the client and the application server; to them, it seems that they are communicating directly with each other.

2 Client Hello

After its connect request is accepted, the client initiates a handshake protocol to produce the cryptographic parameters for the session. The SSL server (representing the application server) responds to the handshake and sends the application server's certificate to the client. The client and the SSL server agree on a protocol version, select cryptographic algorithms (known as cipher suites), and use asymmetric (public-key) encryption techniques to generate shared secrets. From the shared secrets, the SSL server and the client generate the symmetric (private) keys to be used for the encryption and decryption of data sent on the connection.

3 Send

When the handshake completes, the client sends encrypted data over the network. The SSL server receives the encrypted data from the client, decrypts it, and sends it to the application server. The application server responds by sending unencrypted data to the SSL server. The SSL server receives the unencrypted data from the application server, encrypts it, and sends it to the client.

4 Close

When a close is received from either the client or the application server, the SSL server sends a close to the other party and cleans up the connection.

SSL Server Traces

The type of activity that can be traced using SSL server tracing facilities consists of the following:

- NORMAL activity - provides basic information when a connection is successfully established
- CONNECTION activity - provides connection state information and handshake results
- FLOW processes - provides program control and system activity information
- DEBUG processes - provides detailed operational information

Note: In most cases, trace activity can be refined and limited based on a connection number, IP address or port.

SSL server tracing facilities can be activated using two mechanisms. One is to initiate SSL server tracing actions when the server starts, the other is to start or stop tracing actions, after the server initialized and is running.

- To begin tracing activity when the SSL server starts, specify the appropriate TRACE operand for the VMSSL command. This operand can be specified as a **:parms.** tag parameter of an SSL server DTCPARMS entry, or as part of a VMSSL command entered at the SSL server console.

When the SSL server is started, the initialization program searches the DTCPARMS files for configuration definitions that apply to the server. If the SSL entry is unaltered, then default operands for the command (including default tracing) are used. If you want to override the VMSSL command operand defaults, create or modify a SYSTEM DTCPARMS file entry for the SSL server that includes pertinent operands as **:parms.** tag parameters.

- To initiate or halt tracing of the SSL server dynamically, one can issue an SSLADMIN TRACE or NOTRACE command. For SSLADMIN TRACE, specify the appropriate operands with this command to begin immediate tracing. For SSLADMIN NOTRACE, additional operands are not required; all tracing will cease when the command is processed by the SSL server.

For more information on VMSSL, SSLADMIN and the DTCPARMS entry for the SSL Server, see [Configuring the SSL Server in z/VM: TCP/IP Planning and Customization](#).

Diagnosing Problems

The following provides information about problems that you might encounter with the SSL server and suggestions for diagnosing the problem.

Symptom - The SSL Server Does Not Initialize

Documentation

The following documentation should be available for initial diagnosis:

- TCPIP DATA information
- DTCPARMS information
- TCP/IP server configuration file (PROFILE TCPIP, or its equivalent)
- SSL, TCP/IP and SSL DCSS Management Agent server console messages
- GSKADMIN console information
- Trace output (as applicable)

Analysis

1. If the server does not initialize and run the SSLSERV module:
 - a. Check the SSL server console for messages that indicate a problem. Refer to *z/VM: TCP/IP Messages and Codes* and follow the directions for the system programmer response for a particular message.
 - b. Verify that the TCPIPUSERID statement of the TCP/IP data file (TCPIP DATA, by default) referenced by the SSL server cites the correct TCP/IP server virtual machine. If a DTCPARMS **:tcpdatafile.** tag has been used to designate an alternate TCP/IP data file for use, verify that the designated file is accessible by the SSL server (and, that its content is correct). In addition, confirm that the correct TCP/IP and DCSS Management Agent servers have been identified by a DTCPARMS **:stack.** tag defined for the subject server.
 - c. For an SSL pool server, confirm that a DTCPARMS **:vmlink.** tag that identifies the a Shared File System (SFS) directory for access is being properly referenced, and that the cited directory name is correct. Conversely, for a single-instance SSL server, confirm that no such tag is being referenced.
 - d. For an SSL pool server, confirm that the server has been enrolled in the appropriate SFS file pool, and that an alias to the (common-use) PROFILE EXEC is in place.
 - e. For an SSL pool server (and, the case of having attempted a restart of the subject server) confirm that DTCPARMS configuration has not been changed, while one or more other pool servers remain in operation.
2. If the server cannot use the key database:
 - a. Verify that the Byte File System (BFS) parameters for the DTCPARMS **:mount.** tag are correct, and that the necessary file permissions have been established.
 - b. Confirm that the file pool server for the BFS user space (**VMYSU**, by default) is operational.
 - c. Use the GSKKYMAN utility to confirm that the key database has been properly created, and that the correct database has been identified via the VMSSL command KEYFILE operand.
3. If the server cannot use the session cache:
 - a. Verify that the SSL DCSS Management Agent defined for use with the subject SSL server is operational, and that it has been initialized prior to the SSL server. The CP QUERY commands listed here (issued by a user with appropriate command privileges) can help confirm that the server has successfully defined and initialized the session cache DCSS:
 - CP QUERY NSS NAME *nss_name* MAP
 - CP QUERY NSS USERS *nss_name*

- b. Confirm that the necessary NAMESAVE statements are present in the CP directories for the SSL server and its DCSS Management Agent.
4. If the server cannot connect to the TCP/IP virtual machine:
 - a. Verify that the TCPIPUSERID statement of the TCPIP DATA file referenced by the SSL server cites the correct TCP/IP server virtual machine. Also, confirm that the correct TCP/IP server is identified by a DTCPARMS **:stack.** tag defined for the subject SSL server.
 - b. Verify that the TCP/IP server is started.
 - c. Check the TCP/IP server console for messages that indicate a problem. Refer to *z/VM: TCP/IP Messages and Codes* and follow the directions for the system programmer response for a particular message.
 - d. Use the FLOW or DEBUG traces to gather additional information. Update the DTCPARMS **:parms.** tag for the SSL server to include the TRACE FLOW or TRACE DEBUG operand, then restart the server. This will provide debug information during the server start up.

Symptom - Parameters Are Not Correctly Passed to the SSL Server

Documentation

The following documentation should be available for initial diagnosis:

- DTCPARMS information
- SSL server console messages
- SSLADMIN QUERY STATUS output

Analysis

1. Use the SSLADMIN QUERY STATUS command to determine which options are in effect.
2. Check that all parameters are correctly specified on the **:parms.** tag of the DTCPARMS entry for the SSL server entry.
3. Check the server startup console for message DTCRUN1011I; verify the parameters cited are correct and match those defined by the DTCPARMS entry for the SSL server.

Symptom - Protected Application Server Shuts Down at Startup

Documentation

The following documentation should be available for initial diagnosis:

- Application server configuration files
- TCP/IP server configuration file (PROFILE TCPIP, or its equivalent)
- Application and TCP/IP server console messages
- NETSTAT CONN output
- GSKKYMANN output

Analysis

1. Use the NETSTAT CONFIG SSL command to confirm that at least one SSL server is listed with **Active** status. Also, use the NETSTAT CONN command to verify that a listen has been posted by this server (as noted in the example output in [“Displaying Local Host Information”](#) on page 174). Start the server, or server pool, as warranted.
2. Verify that the SSLSERVERID statement in the TCP/IP server configuration file reflects the correct SSL server configuration (identifies a single-instance server, or includes an asterisk (*) for the use of a server pool).

3. Check the application server console for indications of problems. Refer to *z/VM: TCP/IP Messages and Codes* and follow the directions for the system programmer response for a particular message.
4. Verify the TLSLABEL statement and the correct value have been specified in the application server configuration file:
 - PROFILE TCPIP (or its equivalent) for TELNET
 - SMTP CONFIG (or its equivalent) for SMTP
 - SRVRFPT CONFIG (or its equivalent) for FTP

An incorrect or misspelled TLSLABEL value in an application server configuration file can prevent such a server from initializing successfully.
5. Using the GSKKYMANT utility to verify that the TLSLABEL for the certificate specified is present in the certificate database, and that the label name conforms to the requirements cited in *Configuring the SSL Server of z/VM: TCP/IP Planning and Customization*.

Symptom - Connection to a Protected Application Server Cannot be Established

Documentation

The following documentation should be available for initial diagnosis:

- SSL server console messages
- SSLADMIN QUERY STATUS
- NETSTAT CONFIG SSL output
- NETSTAT CONN output
- SSL server TRACE CONNECTIONS output (as applicable)
- TCP/IP server trace output

Analysis

1. Use the SSLADMIN QUERY STATUS or NETSTAT CONFIG SSL command to confirm that an SSL server is active. Also, use the NETSTAT CONN command to verify that a listen has been posted by both the SSL server(s) and the subject application server. Start the servers, as necessary.
2. Use the SSLADMIN QUERY STATUS or NETSTAT CONFIG SSL commands to determine the number of active sessions and the maximum number of sessions that can be accommodated by the SSL server(s) and the system. When the maximum is reached, the TCP/IP server rejects further connections that are destined for the SSL server, until the number of active sessions is less than the system maximum. The maximum number of sessions is determined by the SLLIMITS statement, defined within the TCP/IP server configuration file.
3. Check the SSL server console for messages that indicate a problem. Refer to *z/VM: TCP/IP Messages and Codes* and follow the directions for the system programmer response for a particular message.
4. Issue the SSLADMIN TRACE CONNECTIONS command to activate connection tracing, then try the connection again. The information produced by this trace might explain why the connection cannot be established. When using the CONNECTIONS trace, consider limiting the trace to a specific client IP address or local port.
5. Activate SSL, TCPUP and TCPDOWN process tracing within the TCP/IP server, then attempt another connection, to gather more debug information.

Symptom - Connections Close Due to Errors

Documentation

The following documentation should be available for initial diagnosis:

- TCP/IP server configuration file (PROFILE TCP/IP, or its equivalent)
- SSL server console messages
- SSL server trace output
- GSKKYMANT output
- SSL server TRACE CONNECTIONS output (as applicable)

Analysis

1. Verify that certificate label (specified with a PORT statement, or as part of an application server TLSLABEL statement) is correct. Use the GSKKYMANT utility to ensure that a certificate with the subject label exists in the key database, and that the certificate is valid (for example, has not expired). Note that certificates, added to the key database while an SSL server is in operation, cannot be utilized until after an SSLADMIN REFRESH command has been issued.
2. Check the SSL server console for messages that indicate a problem. Refer to [z/VM: TCP/IP Messages and Codes](#) and follow the directions for the system programmer response for a particular message.
3. Issue the SSLADMIN TRACE CONNECTIONS command to activate connection tracing, then try the connection again. The information produced by this trace might explain why the connection cannot be established. When using the CONNECTIONS trace, consider limiting the trace to a specific client IP address or local port.

Symptom - Incorrect Input or Output

Documentation

The following documentation should be available for initial diagnosis:

- SSLADMIN QUERY SESSIONS output
- SSL server console messages
- SSL server TRACE CONNECTIONS DATA output (as applicable)

Analysis

1. Use the SSLADMIN QUERY SESSIONS command to confirm the subject connection has been established.
2. Check the SSL server console for messages that indicate a problem. Refer to [z/VM: TCP/IP Messages and Codes](#) and follow the directions for the system programmer response for a particular message.
3. Verify that the data is flowing correctly through the SSL server. Issue the SSLADMIN TRACE CONNECTIONS DATA command to activate connection tracing, then try the connection again. The information produced by this trace might explain why the connection cannot be established. When using the CONNECTIONS trace, consider limiting the trace to a specific client IP address or local port.

Trace Output

The trace examples that follow show the output produced when the SSLADMIN command is used to activate **normal**, **connections**, **data** and **flow** traces, respectively. It might be beneficial to refer to the processing flow in [Figure 103 on page 164](#), when these trace examples are reviewed.

Trace Normal

Administrative Console

```
ssladmin trace (ssl ssl00001
DTCSSL2404I Sending command to server(s): SSL00001
SSL00001: Setting trace complete
SSL00001: DTCSSL133I SSLADMIN request processed; RC: 0
```

SSL Server Console

```
14:16:39 DTCSSL003I SSLADMIN request received from TCPMAINT: TRACE    NORMAL ALL
1 14:16:59 Client 9.60.67.164:1544 Port 992 Label TESTCERT
      Cipher RC4_128_SHA Connection established
```

Explanation

1

Identifies a client that has connected to the system. This trace entry includes the client **IP address** and **port**, as well as the **local port** used for the connection. In addition, the certificate **label** and the **cipher** used for authentication and securing the connection are presented. This trace entry is displayed after a session handshake is performed.

Trace Connections NODATA

Administrative Console

```
ssladmin trace connections nodata (ssl ssl00001
DTCSSL2404I Sending command to server(s): SSL00001
SSL00001: Setting trace complete
SSL00001: DTCSSL133I SSLADMIN request processed; RC: 0
```

SSL Server Console

```
15:25:16 DTCSSL003I SSLADMIN request received from TCPMAINT: TRACE CONNECTIONS NODATA ALL
1 14:06:55 DTCSSL019I ( 1005) Connection received from client;      Session ID: ( 8 / -1) Label: TESTCERT
14:06:55 Client Socket:      9.60.67.137..3575
14:06:55 Server Socket:      9.60.28.52..992
2 14:06:55 DTCSSL020I ( 1005) Connection accepted by server;      Session ID: ( 8 / 9)
14:06:55 Client Socket:      9.60.67.137..3575
14:06:55 Server Socket:      9.60.28.52..992
3 14:06:55 DTCSSL021I ( 1005) Handshake complete with remote client; Session ID: ( 8 / 9) Cipher: RC4_128_SHA
14:06:55 Client Socket:      9.60.67.137..3575
14:06:55 Server Socket:      9.60.28.52..992
4 14:06:55 DTCSSL057I ( 1005) CNAME of the validated peer certificate: TCPTEST T1 Client Cert Session ID: ( 8 / 9)
14:06:55 Client Socket:      9.60.67.137..3575
14:06:55 Server Socket:      9.60.28.52..992
5 14:07:01 DTCSSL021I ( 1005) Connection closed; Session ID: ( 8 / 9)
14:07:01 Client Socket:      9.60.67.137..3575
14:07:01 Server Socket:      9.60.28.52..992
14:07:04 DTCSSL003I SSLADMIN request received from TCPMAINT: CLOSECON
```

Explanation

The CONNECTION DATA trace entries supply the following information for a connection:

- **Session ID** for this connection (a pair of file descriptors)
- TCP/IP connection number (**Conn**)
- remote and local host sockets (**Client Socket** and **Server Socket**, respectively), which identify the **IP address** and **port** used by each host.

Additional information is provided based on the nature of the trace entry.

1

Identifies a client that has connected to the system (specifically, the SSL server). The **label** of the certificate used for authentication is included in this trace entry. A connection to a local application has not yet been created.

2

Indicates that a connection has been accepted by the local host application server (here, the Telnet server), with a connection established between it and the SSL server. An association between this connection and that with the remote client has not yet been established.

3

Handshake operations between the SSL server and the client have completed, with the agreed upon cipher suite cited by this entry. Association of the two connections, to form the primary client-to-application connection, has completed. If instead the handshake had failed, this would have been indicated by this trace entry.

4

The client's certificate has been presented and validated by the SSL server. This always follows the validation of the server certificate and would not appear if handshaking had failed. If the client certificate had been rejected, that would have been indicated by this trace entry. The certificate is identified by the Common Name, per X.509 format standards.

5

The primary client-to-application connection (and, its supporting connections) has closed.

Trace Connections DATA

Administrative Console

```
ssladmin trace connections data (ssl ssl00001
DTCSSL2404I Sending command to server(s): SSL00001
SSL00001: Setting trace complete
SSL00001: DTCSSL133I SSLADMIN request processed; RC: 0
```

SSL Server Console

```
17:06:34 DTCSSL003I SSLADMIN request received from TCPMAINT: TRACE CONNECTIONS DATA ALL 20
17:07:05 DTCSSL019I ( 1008) Connection received from client; Session ID: ( 6 / -1) Label: TESTCERT
17:07:05 Client Socket: 9.60.67.137..1392
17:07:05 Server Socket: 9.60.28.52..992
17:07:05 DTCSSL020I ( 1008) Connection accepted by server; Session ID: ( 6 / 7)
17:07:05 Client Socket: 9.60.67.137..1392
17:07:05 Server Socket: 9.60.28.52..992
17:07:05 DTCSSL024I ( 1008) Clear data sent to remote client; Session ID: ( 6 / 7) Bytes: 6
17:07:05 Client Socket: 9.60.67.137..1392
17:07:05 Server Socket: 9.60.28.52..992
17:07:05 DTCSSL021I ( 1008) Handshake complete with remote client; Session ID: ( 6 / 7) Cipher: RC4_128_SHA
17:07:05 Client Socket: 9.60.67.137..1392
17:07:05 Server Socket: 9.60.28.52..992
17:07:05 DTCSSL057I ( 1008) CNAME of the validated peer certificate: TCPTEST T1 Client Cert Session ID: ( 6 / 7)
17:07:05 Client Socket: 9.60.67.137..1392
17:07:05 Server Socket: 9.60.28.52..992
17:07:05 DTCSSL025I ( 1008) Data received from local server; Session ID: ( 6 / 7) Bytes: 3
17:07:05 Client Socket: 9.60.67.137..1392
17:07:05 Server Socket: 9.60.28.52..992
17:07:05 00000000: ff fd 28 |..( | |... |
17:07:05 DTCSSL024I ( 1008) Protected data sent to remote client; Session ID: ( 6 / 7) Bytes: 3
17:07:05 Client Socket: 9.60.67.137..1392
17:07:05 Server Socket: 9.60.28.52..992
17:07:05 00000000: ff fd 28 |..( | |... |
17:07:06 DTCSSL025I ( 1008) Protected data received from remote client; Session ID: ( 6 / 7) Bytes: 3
17:07:06 Client Socket: 9.60.67.137..1392
17:07:06 Server Socket: 9.60.28.52..992
17:07:06 00000000: ff fc 28 |..( | |... |
17:07:06 DTCSSL024I ( 1008) Data sent to local server; Session ID: ( 6 / 7) Bytes: 3
17:07:06 Client Socket: 9.60.67.137..1392
17:07:06 Server Socket: 9.60.28.52..992
17:07:06 00000000: ff fc 28 |..( | |... |
17:07:06 DTCSSL025I ( 1008) Data received from local server; Session ID: ( 6 / 7) Bytes: 3
17:07:06 Client Socket: 9.60.67.137..1392
17:07:06 Server Socket: 9.60.28.52..992
17:07:06 00000000: ff fd 18 |... | |... |
17:07:06 DTCSSL024I ( 1008) Protected data sent to remote client; Session ID: ( 6 / 7) Bytes: 3
17:07:06 Client Socket: 9.60.67.137..1392
17:07:06 Server Socket: 9.60.28.52..992
17:07:06 00000000: ff fd 18 |... | |... |
17:07:06 DTCSSL025I ( 1008) Protected data received from remote client; Session ID: ( 6 / 7) Bytes: 3
17:07:06 Client Socket: 9.60.67.137..1392
17:07:06 Server Socket: 9.60.28.52..992
17:07:06 00000000: ff fb 18 |... | |... |
17:07:06 DTCSSL024I ( 1008) Data sent to local server; Session ID: ( 6 / 7) Bytes: 3
17:07:06 Client Socket: 9.60.67.137..1392
```

```

17:07:06 Server Socket: 9.60.28.52..992
17:07:06 00000000: ff fb 18 |... |...
17:07:06 DTCSSL025I ( 1008) Data received from local server; Session ID: ( 6 / 7) Bytes: 6
17:07:06 Client Socket: 9.60.67.137..1392
17:07:06 Server Socket: 9.60.28.52..992
17:07:06 00000000: ff fa 18 01 ff f0 |..... |.....0 |
17:07:06 DTCSSL024I ( 1008) Protected data sent to remote client; Session ID: ( 6 / 7) Bytes: 6
17:07:06 Client Socket: 9.60.67.137..1392
17:07:06 Server Socket: 9.60.28.52..992
17:07:06 00000000: ff fa 18 01 ff f0 |..... |.....0 |
17:07:06 DTCSSL025I ( 1008) Protected data received from remote client; Session ID: ( 6 / 7) Bytes: 18
17:07:06 Client Socket: 9.60.67.137..1392
17:07:06 Server Socket: 9.60.28.52..992
17:07:06 00000000: ff fa 18 00 49 42 4d 2d |...IBM-| |.....(
17:07:06 00000008: 33 32 37 38 2d 34 2d 45 |3278-4-E| |.....|
17:07:06 00000010: ff f0 |.. |.0 |
17:07:06 DTCSSL024I ( 1008) Data sent to local server; Session ID: ( 6 / 7) Bytes: 18
17:07:06 Client Socket: 9.60.67.137..1392
17:07:06 Server Socket: 9.60.28.52..992
17:07:06 00000000: ff fa 18 00 49 42 4d 2d |...IBM-| |.....(
17:07:06 00000008: 33 32 37 38 2d 34 2d 45 |3278-4-E| |.....|
17:07:06 00000010: ff f0 |.. |.0 |
17:07:06 DTCSSL025I ( 1008) Data received from local server; Session ID: ( 6 / 7) Bytes: 12
...

```

Explanation

- 1** Identifies a client that has connected to the system (specifically, the SSL server). The **label** of the certificate used for authentication is included in this trace entry. A connection to a local application has not yet been created.
- 2** Indicates that a connection has been accepted by the local host application server (here, the Telnet server), with a connection established between it and the SSL server. An association between this connection and that with the remote client has not yet been established.
- 3** Identifies that the SSL server is processing handshake operations and might be requesting the client to present a valid certificate. Data is being sent in clear text because handshaking has not been completed yet.
- 4** Handshake operations between the SSL server and the client have completed, with the agreed upon cipher suite cited by this entry. Association of the two connections, to form the primary client-to-application connection, has completed. If instead the handshake had failed, this would have been indicated by a respective trace entry.
- 5** The client's certificate has been presented and validated by the SSL server. This always follows the validation of the server certificate and would not appear if handshaking had failed. If the client certificate had been rejected, that would have been indicated by this trace entry. The certificate is identified by the Common Name, per X.509 format standards.
- 6** The SSL server has received data from the local application server.
- 7** The data noted by entry **6** has been encrypted by the SSL server and sent to the remote client.
- 8** Encrypted data has been received by the SSL server from the remote client.
- 9** The data noted by entry **8** has been decrypted by the SSL server and sent to the local application server.
- 10** This, and the remaining trace entries continue to show the exchange of data between the local server and the remote client (repeating the events noted by entries **6**, **7**, **8**, and **9**).

Trace FLOW

Administrative Console

```
ssladmin trace flow (ssl ssl00001
DTCSSL2404I Sending command to server(s): SSL00001
SSL00001: Setting trace complete
SSL00001: DTCSSL133I SSLADMIN request processed; RC: 0
```

SSL Server Console

```
14:17:25 DTCSSL003I SSLADMIN request received from TCPMAINT: TRACE    FLOW ALL
14:17:25 SSLTRACE C A1:258 set_trace_filter: >>exit
14:17:25 SSLADMIN C A1:220 cmd_trace: >>exit
14:17:25 SSLADMIN C A1:162 ar_finalize: <<enter
14:17:25 SSLADMIN C A1:164 ar_finalize: Buflen=512, wrtoff=83, rc=0
14:17:25 SSLADMIN C A1:170 ar_finalize: >>exit
14:17:25 SSLADMIN C A1:604 process_admin_command: >>exit
14:17:34 SSLTRISIT C A1:514 h_acc_main: <<enter
1 14:17:34 SSLTRISIT C A1:532 h_acc_main: accepted conn 1499 from 9.60.67.164:1545 to 9.60.28.52:992
    label "TESTCERT" (TCB 0x03203be0)
14:17:34 SSLSCBEX C A1:139 get_scbx: <<enter
14:17:34 SSLSCBEX C A1:144 get_scbx: about to run tsearch, index=F642030, key=0x03203be0
14:17:34 SSLSCBEX C A1:148 get_scbx: get_scbx adding new SCBX with key 0x03203be0 at F642D40
14:17:34 SSLSCBEX C A1:195 update_session_count: session count up to 2, stop when empty is OFF
14:17:34 SSLSCBEX C A1:155 get_scbx: >>exit
14:17:34 SSLTRISIT C A1:431 l_connect: <<enter
14:17:34 SSLTRISIT C A1:454 l_connect: about to call setibmssockopt(9,...SO_SSL) (local conn socket)
14:17:34 SSLTRISIT C A1:467 l_connect: setibmssockopts OK
14:17:34 SSLTRISIT C A1:490 l_connect: wait for asynchronous connect on fd 9
14:17:34 SSLTRISIT C A1:493 l_connect: >>exit
14:17:34 SSLTRISIT C A1:401 h_conn: <<enter
14:17:34 SSLTRISIT C A1:375 connectcomplete: <<enter
14:17:34 SSLTRISIT C A1:379 connectcomplete: scb (9,8), scbx 0x0f642d40
14:17:34 SSLTRISIT C A1:386 connectcomplete: proceed with SSL handshake in listen conn
14:17:34 SSLGSKCF C A1:458 sslinit: <<enter
2 14:17:34 SSLGSKCF C A1:464 sslinit: scb (9,8), protecting server, fd 8, reqcert no validate no
3 14:17:34 SSLGSKCF C A1:514 sslinit: handshake will use cert label "TESTCERT"
14:17:34 SSLGSKCF C A1:539 sslinit: >>exit
14:17:34 SSLTRISIT C A1:390 connectcomplete: >>exit
14:17:34 SSLTRISIT C A1:419 h_conn: >>exit
4 14:17:34 SSLGSKCF C A1:262 h_hshake: <<enter
14:17:34 SSLGSKCF C A1:266 h_hshake: gsk_secure_socket_init rc=503 (Socket read request would block)
14:17:34 SSLGSKCF C A1:373 h_hshake: >>exit
14:17:34 SSLGSKCF C A1:262 h_hshake: <<enter
14:17:34 SSLCACHE C A1:731 gsk_cch_put: Put ver=30 len=32
    id="00004807093C43A4060900000000000000000000000000004BD72A3E00000002"
14:17:34 SSLGSKCF C A1:266 h_hshake: gsk_secure_socket_init rc=0 ()
5 14:17:34 SSLGSKCF C A1:270 h_hshake: SSL handshake complete on listen fd 8
14:17:34 SSLGSKCF C A1:296 h_hshake: established session is SSLv3, cipher=05:RC4_128_SHA cl=2 hs=0
al=2 kl=128
14:17:34 SSLCTLIO C A1:118 report_statechange: <<enter
14:17:34 SSLCTLIO C A1:149 report_statechange: >>exit
14:17:34 SSLMNTOR C A1:290 mon_startsession: <<enter
6 14:17:34 SSLMNTOR C A1:304 mon_startsession: hs params: dynamic=no, resumed=no, bits=128
14:17:34 SSLMNTOR C A1:171 GetCertKeyLength: <<enter
7 14:17:34 SSLMNTOR C A1:223 GetCertKeyLength: GetCertKeyLength() returning: 1024
14:17:34 SSLMNTOR C A1:225 GetCertKeyLength: >>exit
14:17:34 SSLMNTOR C A1:339 mon_startsession: >>exit
8 14:17:34 SSLGSKCF C A1:373 h_hshake: >>exit
14:17:45 SSLADMIO C A1:106 trap_routine: <<enter
14:17:45 SSLADMIO C A1:120 trap_routine: EventTest returned flag=4
14:17:45 SSLADMIO C A1:136 trap_routine: EventRetrieve returned 4 bytes
14:17:45 SSLADMIO C A1:167 trap_routine: >>exit
```

Explanation

The following can be used as a general guideline when TRACE FLOW output is reviewed and evaluated:

- The first token is a time stamp, in the format **hh:mm:ss**
- The second through fifth tokens provide information about the source of the trace entry. This information is comprised of a (compilation) **source file name**, **file type** and **file mode** (combined with a **source line number**), along with a **routine** (or, **function**) name.
- The remainder of the trace entry contains descriptive information or other pertinent data.

Note that entry to, and exit from, a function or routine is designated by trace entries of the form:

```
routine_name: <<enter
...
routine_name: >>exit
```

Descriptions of various entries for the preceding TRACE FLOW example follow:

- 1** Indicates that a connection from a remote client has been accepted, which will be authenticated using the certificate with the indicated label.
- 2** Identifies the session ID that pertains to the new connection, along with additional status and attribute information.
- 3** Provides information about the certificate label that will be used during handshake operations.
- 4** Indicates that the routine to perform handshake operations for the secure session now is being used.
- 5** This and the next trace entry indicate completion of the session handshake, and that the established session will be protected using the named cipher suite.
- 6 and 7** The lines represented by **6** and **7** identify attributes about this session — such as whether the connection is implicit (dynamic=no) or explicit (dynamic=yes), and the key bit length of the certificate associated with this session — that are used for creating monitor data.
- 8** Indicates that the routine to perform handshake operations as completed operation.

Displaying Local Host Information

There are times when it might be helpful to use the NETSTAT command to display information about active TCP/IP host connections, as well as the SSL server configuration.

The following is an example of output displayed upon invoking the NETSTAT CONN command.

netstat conn					
VM TCP/IP Netstat function level 730			TCP/IP Server Name: TCPIP		
Active IPv4 Transmission Blocks:					
	User Id	Conn	Local Socket	Foreign Socket	State
	-----	----	-----	-----	-----
	FTPSSERVE	1501	*..990	*..*	Listen
	INTCLIEN	1001	*..TELNET	*..*	Listen
	INTCLIEN	1003	*..992	*..*	Listen
1	INTCLIEN	1498	9.60.28.52..992	9.60.67.164..1544	Established
	INTCLIEN	1500	9.60.28.52..992	9.60.67.164..1545	Established
	SSL00001	1496	*..1539	*..*	Listen
	SSL00001	1005	127.0.0.1..1540	127.0.0.1..1541	Established
2	SSL00001	1009	9.60.28.52..1539	9.60.67.164..1544	Established
		1497			
	SSL00001	1499	9.60.28.52..1539	9.60.67.164..1545	Established
		1503			
3	SSL00001	1497	9.60.28.52..1554	9.60.28.52..992	Established
		1009			
	SSL00001	1503	9.60.28.52..1555	9.60.28.52..992	Established
		1499			
4	SSL00002	1010	*..1545	*..*	Listen
	SSL00002	1002	127.0.0.1..1546	127.0.0.1..1547	Established
	SSL00004	1012	*..1542	*..*	Listen
	SSL00004	1008	127.0.0.1..1543	127.0.0.1..1544	Established
	SSL00003	1006	*..1548	*..*	Listen
	SSL00003	1004	127.0.0.1..1549	127.0.0.1..1550	Established
	SSL00005	1007	*..1551	*..*	Listen
	SSL00005	1495	127.0.0.1..1552	127.0.0.1..1553	Established

Active IPv6 Transmission Blocks: None

Explanation

1

This line shows a connection between the Telnet server (running on a z/VM host with IP address 9.60.28.52) and a remote client (IP address of 9.60.67.164). Both the client and application server (here, the Telnet server) share this view of the connection.

2 and 3

The lines represented by **2** and **3**, respectively, show the further breakdown of the primary connection into two connections: the line identified by **2** being the connection from the SSL server and the remote client, and the line identified by **3** as being the connection between the SSL server to the application (Telnet) server.

4

This line reflects the posting of a listen by the SSL00002 server, for incoming connections from a remote client.

Note that similar connection information (in a somewhat more concise form, but with no state information) can be obtained using the NETSTAT IDENTIFY SSL command. Sample output for this command — for the same connections shown in the previous example — is illustrated here:

```
netstat identify ssl
VM TCP/IP Netstat function level 730      TCP/IP Server Name: TCPIP
1 9.60.28.52 992 9.60.67.164 1544 INTCLIEN I SSL00001
9.60.28.52 992 9.60.67.164 1545 INTCLIEN I SSL00001
2 9.60.28.52 1539 9.60.67.164 1544 SSL00001 I SSL00001 TESTCERT SSLV3 SHA1 RC4
9.60.28.52 1539 9.60.67.164 1545 SSL00001 I SSL00001 TESTCERT SSLV3 SHA1 RC4
```

Explanation

1

This line shows a connection between the Telnet server and a remote client. Again, both the client and application server (here, the Telnet server) share this view of the connection.

2

This line shows the connection between the SSL server and the remote client. The connection between the SSL server and the application (Telnet) server is not included in the NETSTAT IDENTIFY SSL command output.

Chapter 14. 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.

VM NFS Client Support

The following is client support for VM network file system.

Activating Traces for NFS Client

Debugging the NFS client is activated by the OPENVM DEBUG command. For more information on the OPENVM DEBUG command, see the [z/VM: OpenExtensions Commands Reference](#).

VM NFS Server Support

The following is server support for VM network file system.

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 PCNFSD 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.

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 178](#).

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 [z/VM: TCP/IP User's Guide](#) or 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 any accessed file mode of the NFS server virtual machine:

VMNFS DUMP_REQ

The NFS server will use the first file named VMNFS DUMP_REQ found in its search order. The first line of this file should contain the mask value of FFFFFFFF which 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.

Activating Traces for NFS Server

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 104 on page 183](#).

Additional Trace Options

Additional trace options for the NFS server are described in the following sections.

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 104 on page 183 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 11, “RPC Programs,” on page 145).

- 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
Acid.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
Acid.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 104. A Sample of an NFS Trace of a Bad Mount

Chapter 15. 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.

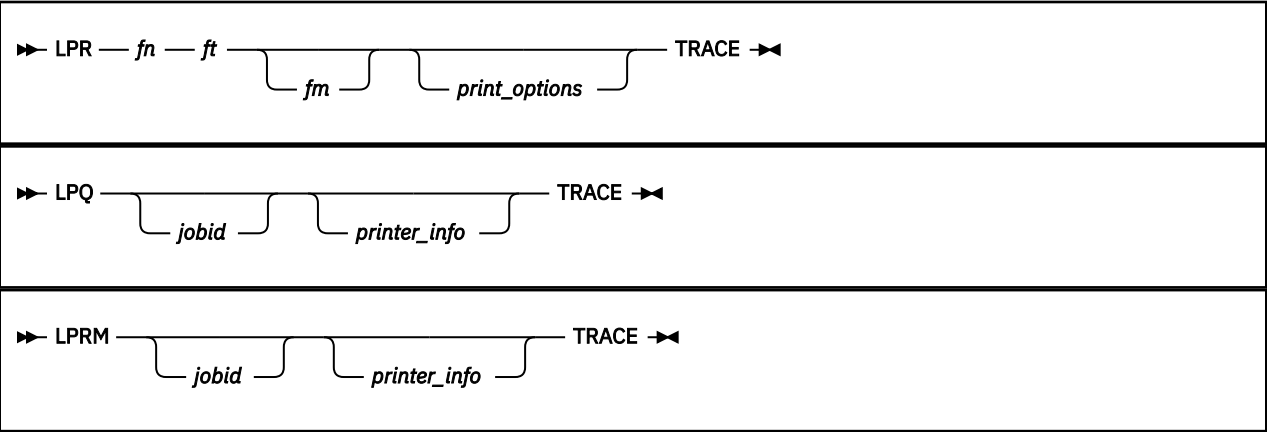
Remote Printing Client Traces

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:



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 the [z/VM: TCP/IP 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 105 on page 186 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 105. A Sample of an LPR Client Trace (Part 1 of 2)

```

----- 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 "fdA164VM1"
Queuing control line "UdA164VM1"
Queuing control line "NDOIT.EXEC"
Sending command 2 argument: "74 cA164VM1"
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 106. A Sample of an LPR Client Trace (Part 2 of 2)

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 (LPSEVERE) 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 "cA" (control file) or "dA" (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.

Chapter 16. Remote Execution Protocol Traces

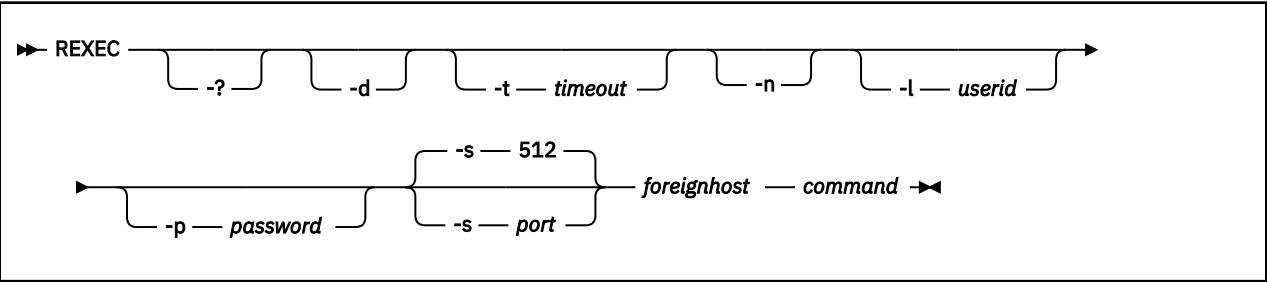
The following sections describe the tracing capabilities available in the client and server functions provided with the Remote Execution Protocol implementation in TCP/IP for VM.

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:



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 [z/VM: TCP/IP User's Guide](#) for information on the usage of the other parameters.

Remote Execution Protocol Client Trace Output

Figure 107 on page 190 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)
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
TCPMAINT - 602, VMNFS - 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

```

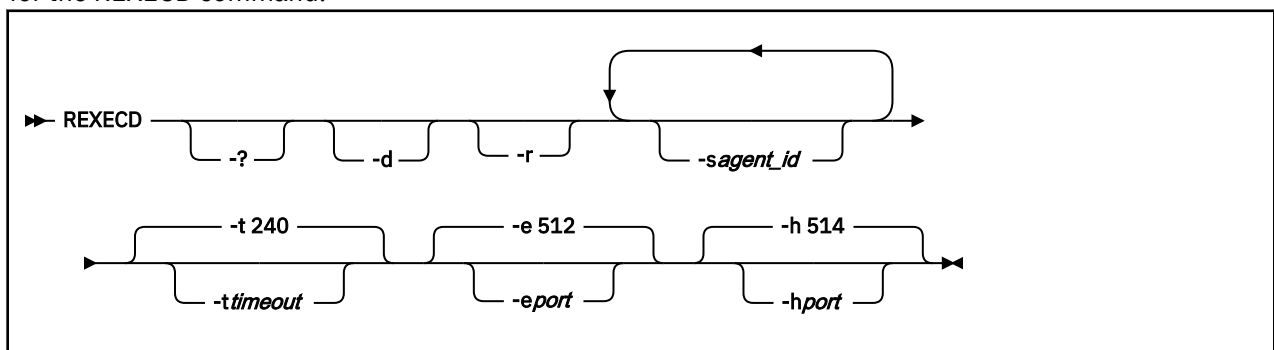
Figure 107. A Sample of a Remote Execution Client Trace

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:



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 [z/VM: TCP/IP Planning and Customization](#) for information on the usage of the other parameters.

Remote Execution Protocol Server Trace Output

Figure 108 on page 191 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 encountered at 160600
received IUCV interrupt - from user RXAGENT1
iucv type is - pending connectionNotification: IUCV interrupt
    IUCV interrupt encountered at 160600
received IUCV interrupt - from user
iucv type is - pending (priority) msgclearing actconn 3
    Notification: IUCV interrupt
    IUCV interrupt encountered 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 108. A Sample of a Remote Execution Protocol Server Trace

Chapter 17. Hardware Trace Functions

This chapter describes PCCA 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 109 on page 193](#) 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.

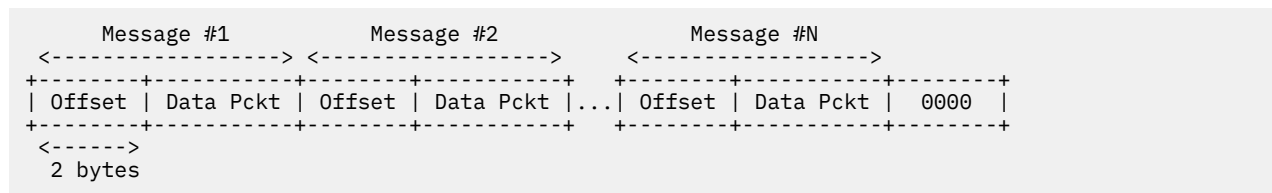


Figure 109. PCCA Block Structure

The PCCA block can be divided into two modes. [Figure 110 on page 193](#) shows a sample of a PCCA block with a series of messages. All highlighted halfwords in [Figure 110 on page 193](#) 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 110. 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 111 on page 194](#) shows the structure of a PCCA control message, which has three fields.

The following are descriptions of the fields shown in [Figure 111 on page 194](#).

- 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 111. 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 112 on page 195 shows a sample of a trace started by a CPTRAP IO command issued on a VM/SP6 system.

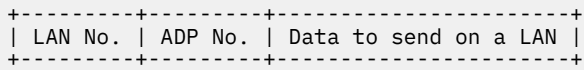


Figure 112. PCCA LAN Messages Structure

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 113 on page 195 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.

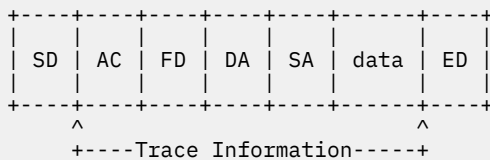


Figure 113. Common Layout of a Token-Ring Packet

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 21 on page 196 lists the functions of the PCCA CCW codes.

Table 21. 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 114 on page 197 and Figure 115 on page 198 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 114 on page 197 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 114. A Sample of an ARP Frame on a PCCA Token-Ring

Figure 115 on page 198 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.

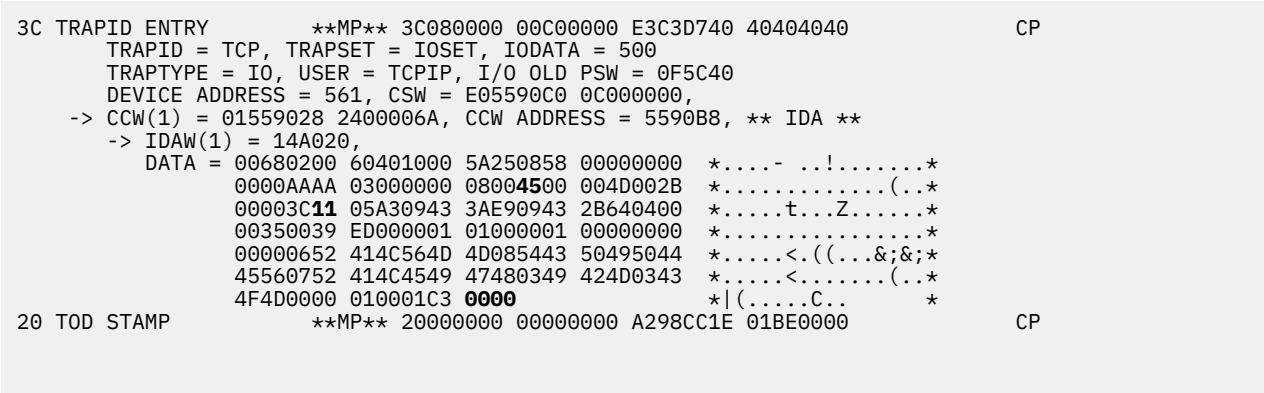


Figure 115. A Sample of an IP/ICMP Packet on a PCCA Token-Ring

Figure 116 on page 199 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. |

Figure 117 on page 199 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 116 on page 199.

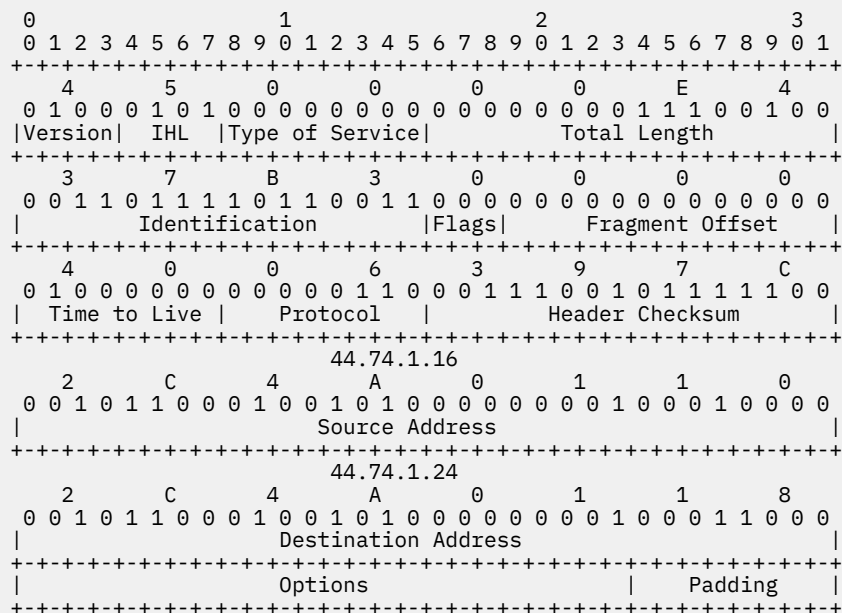


Figure 118 on page 200 shows the TCP header format.

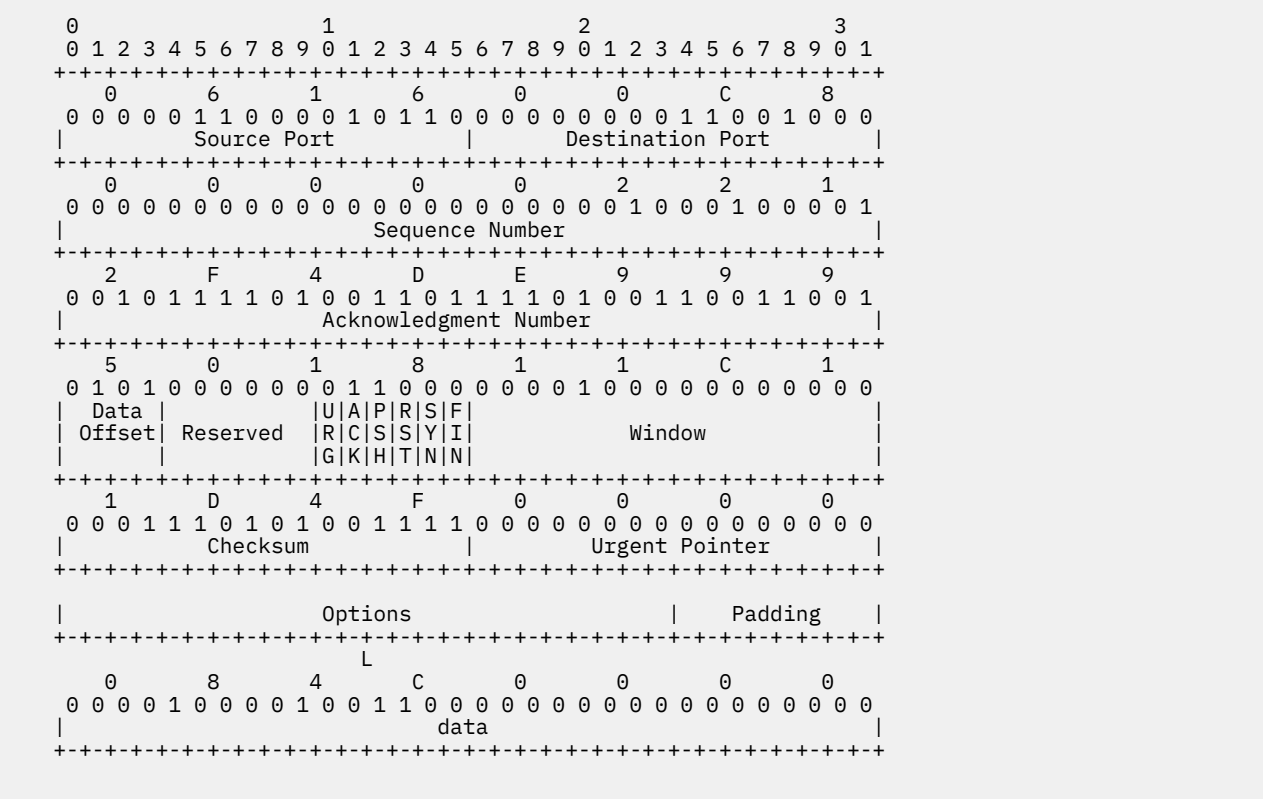


Figure 118. TCP Header Format

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 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.

NETSTAT OSAINFO

With the NETSTAT OSAINFO command, the installation of OSA/SF for OSA-Express 3 devices to satisfy diagnostic requirements is not necessary any longer.

The NETSTAT OSAINFO command provides z/VM TCP/IP with the capability to display the content of the OSA Address Table (OAT) via the QDIO interface for both, OSA cards and VSWITCH controllers.

See the [z/VM: TCP/IP User's Guide](#) for a complete description of the NETSTAT OSAINFO command.



- The following is an example of using the NETSTAT OSINFO command:
- For Layer 2:

```

NETSTAT OSAINFO (SELECT DEVB100
VM TCP/IP Netstat Level 620 TCP/IP Server Name: TCPIPBX

Device DEVB100: data as of 06/06/10 09:49:28
  VMAC address:      02-09-57-60-00-63
  VLAN ID:          1

```

- For Layer 3:

```

NETSTAT OSAINFO (SELECT DEVB100
VM TCP/IP Netstat Level 620 TCP/IP Server Name: TCPIPBX

Device DEVB100: data as of 06/06/10 09:49:28
  IPv4 Address:
  -----
  9.100.200.123
  9.100.200.124
  9.100.200.125

  IPv4 Multicast Address:      MAC Address:
  -----
  224.0.0.1                    02-09-57-60-00-63
  224.0.0.5                    02-09-57-60-00-63

```


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 22 on page 203 describes the return codes sent by TCP/IP to servers and clients through the Virtual Machine Communication Facility (VMCF).

<i>Table 22. TCP/IP Return Codes Sent to Servers and Clients</i>		
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	

<i>Table 22. TCP/IP Return Codes Sent to Servers and Clients (continued)</i>		
Return Message	Value	Description
NOTyetBEGUN	-25	Client has not called BeginTcpIp.
NOTyetOPEN	-26	Client has not called TcpOpen.
OPENrejected	-27	
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.
REMOTrereset	-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.
YOURend	-55	
ZEROresources	-56	

UDP Error Return Codes

Table 23 on page 204 describes errors that are specific to UDP.

<i>Table 23. UDP Error Return Codes</i>		
Return Message	Value	Description
UDPlocalADDRESS	-57	Invalid local address.

<i>Table 23. UDP Error Return Codes (continued)</i>		
Return Message	Value	Description
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

Many features of TCP/IP for z/VM are based on the following RFCs:

RFC	Title	Author
768	<i>User Datagram Protocol</i>	J.B. Postel
791	<i>Internet Protocol</i>	J.B. Postel
792	<i>Internet Control Message Protocol</i>	J.B. Postel
793	<i>Transmission Control Protocol</i>	J.B. Postel
821	<i>Simple Mail Transfer Protocol</i>	J.B. Postel
822	<i>Standard for the Format of ARPA Internet Text Messages</i>	D. Crocker
823	<i>DARPA Internet Gateway</i>	R.M. Hinden, A. Sheltzer
826	<i>Ethernet Address Resolution Protocol: or Converting Network Protocol Addresses to 48.Bit Ethernet Address for Transmission on Ethernet Hardware</i>	D.C. Plummer
854	<i>Telnet Protocol Specification</i>	J.B. Postel, J.K. Reynolds
856	<i>Telnet Binary Transmission</i>	J.B. Postel, J.K. Reynolds
857	<i>Telnet Echo Option</i>	J.B. Postel, J.K. Reynolds
877	<i>Standard for the Transmission of IP Datagrams over Public Data Networks</i>	J.T. Korb
885	<i>Telnet End of Record Option</i>	J.B. Postel
903	<i>Reverse Address Resolution Protocol</i>	R. Finlayson, T. Mann, J.C. Mogul, M. Theimer
904	<i>Exterior Gateway Protocol Formal Specification</i>	D.L. Mills
919	<i>Broadcasting Internet Datagrams</i>	J.C. Mogul
922	<i>Broadcasting Internet Datagrams in the Presence of Subnets</i>	J.C. Mogul
950	<i>Internet Standard Subnetting Procedure</i>	J.C. Mogul, J.B. Postel
952	<i>DoD Internet Host Table Specification</i>	K. Harrenstien, M.K. Stahl, E.J. Feinler
959	<i>File Transfer Protocol</i>	J.B. Postel, J.K. Reynolds
974	<i>Mail Routing and the Domain Name System</i>	C. Partridge
1009	<i>Requirements for Internet Gateways</i>	R.T. Braden, J.B. Postel
1014	<i>XDR: External Data Representation Standard</i>	Sun Microsystems Incorporated
1027	<i>Using ARP to Implement Transparent Subnet Gateways</i>	S. Carl-Mitchell, J.S. Quarterman
1032	<i>Domain Administrators Guide</i>	M.K. Stahl
1033	<i>Domain Administrators Operations Guide</i>	M. Lottor
1034	<i>Domain Names—Concepts and Facilities</i>	P.V. Mockapetris

RFC	Title	Author
1035	<i>Domain Names—Implementation and Specification</i>	P.V. Mockapetris
1042	<i>Standard for the Transmission of IP Datagrams over IEEE 802 Networks</i>	J.B. Postel, J.K. Reynolds
1055	<i>Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP</i>	J.L. Romkey
1057	<i>RPC: Remote Procedure Call Protocol Version 2 Specification</i>	Sun Microsystems Incorporated
1058	<i>Routing Information Protocol</i>	C.L. Hedrick
1091	<i>Telnet Terminal-Type Option</i>	J. VanBokkelen
1094	<i>NFS: Network File System Protocol Specification</i>	Sun Microsystems Incorporated
1112	<i>Host Extensions for IP Multicasting</i>	S. Deering
1118	<i>Hitchhikers Guide to the Internet</i>	E. Krol
1122	<i>Requirements for Internet Hosts-Communication Layers</i>	R.T. Braden
1123	<i>Requirements for Internet Hosts-Application and Support</i>	R.T. Braden
1155	<i>Structure and Identification of Management Information for TCP/IP-Based Internets</i>	M.T. Rose, K. McCloghrie
1156	<i>Management Information Base for Network Management of TCP/IP-based Internets</i>	K. McCloghrie, M.T. Rose
1157	<i>Simple Network Management Protocol (SNMP),</i>	J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin
1179	<i>Line Printer Daemon Protocol</i>	The Wollongong Group, L. McLaughlin III
1180	<i>TCP/IP Tutorial,</i>	T. J. Socolofsky, C.J. Kale
1183	<i>New DNS RR Definitions (Updates RFC 1034, RFC 1035)</i>	C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris,
1187	<i>Bulk Table Retrieval with the SNMP</i>	M.T. Rose, K. McCloghrie, J.R. Davin
1207	<i>FYI on Questions and Answers: Answers to Commonly Asked Experienced Internet User Questions</i>	G.S. Malkin, A.N. Marine, J.K. Reynolds
1208	<i>Glossary of Networking Terms</i>	O.J. Jacobsen, D.C. Lynch
1213	<i>Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II,</i>	K. McCloghrie, M.T. Rose
1215	<i>Convention for Defining Traps for Use with the SNMP</i>	M.T. Rose
1228	<i>SNMP-DPI Simple Network Management Protocol Distributed Program Interface</i>	G.C. Carpenter, B. Wijnen
1229	<i>Extensions to the Generic-Interface MIB</i>	K. McCloghrie
1267	<i>A Border Gateway Protocol 3 (BGP-3)</i>	K. Loughheed, Y. Rekhter
1268	<i>Application of the Border Gateway Protocol in the Internet</i>	Y. Rekhter, P. Gross

RFC	Title	Author
1269	<i>Definitions of Managed Objects for the Border Gateway Protocol (Version 3)</i>	S. Willis, J. Burruss
1293	<i>Inverse Address Resolution Protocol</i>	T. Bradley, C. Brown
1270	<i>SNMP Communications Services</i>	F. Kastenholz, ed.
1323	<i>TCP Extensions for High Performance</i>	V. Jacobson, R. Braden, D. Borman
1325	<i>FYI on Questions and Answers: Answers to Commonly Asked New Internet User Questions</i>	G.S. Malkin, A.N. Marine
1351	<i>SNMP Administrative Model</i>	J. Davin, J. Galvin, K. McCloghrie
1352	<i>SNMP Security Protocols</i>	J. Galvin, K. McCloghrie, J. Davin
1353	<i>Definitions of Managed Objects for Administration of SNMP Parties</i>	K. McCloghrie, J. Davin, J. Galvin
1354	<i>IP Forwarding Table MIB</i>	F. Baker
1387	<i>RIP Version 2 Protocol Analysis</i>	G. Malkin
1389	<i>RIP Version 2 MIB Extension</i>	G. Malkin
1393	<i>Traceroute Using an IP Option</i>	G. Malkin
1397	<i>Default Route Advertisement In BGP2 And BGP3 Versions of the Border Gateway Protocol</i>	D. Haskin
1398	<i>Definitions of Managed Objects for the Ethernet-like Interface Types</i>	F. Kastenholz
1440	<i>SIFT/UFT:Sender-Initiated/Unsolicited File Transfer</i>	R. Troth
1493	<i>Definition of Managed Objects for Bridges</i>	E. Decker, P. Langille, A. Rijssinghani, K. McCloghrie
1540	<i>IAB Official Protocol Standards</i>	J.B. Postel
1583	<i>OSPF Version 2</i>	J.Moy
1647	<i>TN3270 Enhancements</i>	B. Kelly
1700	<i>Assigned Numbers</i>	J.K. Reynolds, J.B. Postel
1723	<i>RIP Version 2 – Carrying Additional Information</i>	G. Malkin
1738	<i>Uniform Resource Locators (URL)</i>	T. Berners-Lee, L. Masinter, M. McCahill
1813	<i>NFS Version 3 Protocol Specification</i>	B. Callaghan, B. Pawlowski, P. Stauback, Sun Microsystems Incorporated
1823	<i>The LDAP Application Program Interface</i>	T. Howes, M. Smith
2460	<i>Internet Protocol, Version 6 (IPv6) Specification</i>	S. Deering, R. Hinden
2052	<i>A DNS RR for specifying the location of services (DNS SRV)</i>	A. Gulbrandsen, P. Vixie

RFC	Title	Author
2104	<i>HMAC: Keyed-Hashing for Message Authentication</i>	H. Krawczyk, M. Bellare, R. Canetti
2222	<i>Simple Authentication and Security Layer (SASL)</i>	J. Myers
2247	<i>Using Domains in LDAP/X.500 Distinguished Names</i>	S. Kille, M. Wahl, A. Grimstad, R. Huber, S. Sataluri
2251	<i>Lightweight Directory Access Protocol (v3)</i>	M. Wahl, T. Howes, S. Kille
2252	<i>Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions</i>	M. Wahl, A. Coulbeck, T. Howes, S. Kille
2253	<i>Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names</i>	M. Wahl, S. Kille, T. Howes
2254	<i>The String Representation of LDAP Search Filters</i>	T. Howes
2255	<i>The LDAP URL Format</i>	T. Howes, M. Smith
2256	<i>A Summary of the X.500 (96) User Schema for use with LDAPv3</i>	M. Wahl
2279	<i>UTF-8, a transformation format of ISO 10646</i>	F. Yergeau
2373	<i>IP Version 6 Addressing Architecture</i>	R. Hinden, S. Deering
2461	<i>Neighbor Discovery for IP Version 6 (IPv6)</i>	T. Narten, E. Nordmark, W. Simpson
2462	<i>IPv6 Stateless Address Autoconfiguration</i>	S. Thomson, T. Narten
2463	<i>Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification</i>	A. Conta, S. Deering
2710	<i>Multicast Listener Discovery (MLD) for IPv6</i>	S. Deering, W. Fenner, B. Haberman
2713	<i>Schema for Representing Java Objects in an LDAP Directory</i>	V. Ryan, S. Seligman, R. Lee
2714	<i>Schema for Representing CORBA Object References in an LDAP Directory</i>	V. Ryan, R. Lee, S. Seligman
2732	<i>Format for Literal IPv6 Addresses in URLs</i>	R. Hinden, B. Carpenter, L. Masinter
2743	<i>Generic Security Service Application Program Interface Version 2, Update 1</i>	J. Linn
2744	<i>Generic Security Service API Version 2 : C-bindings</i>	J. Wray
2820	<i>Access Control Requirements for LDAP</i>	E. Stokes, D. Byrne, B. Blakley, P. Behera
2829	<i>Authentication Methods for LDAP</i>	M. Wahl, H. Alvestrand, J. Hodges, R. Morgan
2830	<i>Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security</i>	J. Hodges, R. Morgan, M. Wahl
2831	<i>Using Digest Authentication as a SASL Mechanism</i>	P. Leach, C. Newman
2849	<i>The LDAP Data Interchange Format (LDIF)</i>	G. Good

RFC	Title	Author
2873	<i>TCP Processing of the IPv4 Precedence Field</i>	X. Xiao, A. Hannan, V. Paxson, E. Crabble
3377	<i>Lightweight Directory Access Protocol (v3): Technical Specification</i>	J. Hodges, R. Morgan
3484	<i>Default Address Selection for Internet Protocol version 6 (IPv6)</i>	R. Draves
3513	<i>Internet Protocol Version 6 (IPv6) Addressing Architecture</i>	R. Hinden, S. Deering
4191	<i>Default Router Preferences and More-Specific Routes</i>	R. Draves, D. Thaler
4517	<i>LDAP Syntaxes and Matching Rules</i>	S. Legg
4523	<i>LDAP Schema Definitions for X.509 Certificates</i>	K. Zeilenga
5095	<i>Deprecation of Type 0 Routing Headers in IPv6</i>	J. Abley, P. Savola, G. Neville-Nei
5175	<i>IPv6 Router Advertisement Flags Option</i>	B. Haberman, R. Hinden
5722	<i>Handling of Overlapping IPv6 Fragments</i>	S. Krishnan
6946	<i>Processing of IPv6 "Atomic" Fragments</i>	F. Gont
6980	<i>Security Implications of IPv6 Fragmentation with IPv6</i>	F. Gont

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 at [Internet Engineering Task Force](http://www.ietf.org) (www.ietf.org).

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US*

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

*Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

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 jurisdictions 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 in 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide 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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US*

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.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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 of non-IBM products should be addressed to the suppliers of those products.

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

This information may contain 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 actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Programming Interface Information

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

Trademarks

IBM, the IBM logo, and [ibm.com](https://www.ibm.com)® are trademarks or registered trademarks of International Business Machines Corp., in the United States and/or other countries. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on [IBM Copyright and trademark information](https://www.ibm.com/legal/copytrade) (<https://www.ibm.com/legal/copytrade>).

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Terms and Conditions for Product Documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal Use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see:

- The section entitled **IBM Websites** at [IBM Privacy Statement](https://www.ibm.com/privacy) (<https://www.ibm.com/privacy>)
- [Cookies and Similar Technologies](https://www.ibm.com/privacy#Cookies_and_Similar_Technologies) (https://www.ibm.com/privacy#Cookies_and_Similar_Technologies)

Bibliography

This topic lists the publications in the z/VM library. For abstracts of the z/VM publications, see [z/VM: General Information](#).

Where to Get z/VM Information

The current z/VM product documentation is available in [IBM Documentation - z/VM \(https://www.ibm.com/docs/en/zvm\)](https://www.ibm.com/docs/en/zvm).

z/VM Base Library

Overview

- [z/VM: License Information](#), GI13-4377
- [z/VM: General Information](#), GC24-6286

Installation, Migration, and Service

- [z/VM: Installation Guide](#), GC24-6292
- [z/VM: Migration Guide](#), GC24-6294
- [z/VM: Service Guide](#), GC24-6325
- [z/VM: VMSES/E Introduction and Reference](#), GC24-6336

Planning and Administration

- [z/VM: CMS File Pool Planning, Administration, and Operation](#), SC24-6261
- [z/VM: CMS Planning and Administration](#), SC24-6264
- [z/VM: Connectivity](#), SC24-6267
- [z/VM: CP Planning and Administration](#), SC24-6271
- [z/VM: Getting Started with Linux on IBM Z](#), SC24-6287
- [z/VM: Group Control System](#), SC24-6289
- [z/VM: I/O Configuration](#), SC24-6291
- [z/VM: Running Guest Operating Systems](#), SC24-6321
- [z/VM: Saved Segments Planning and Administration](#), SC24-6322
- [z/VM: Secure Configuration Guide](#), SC24-6323

Customization and Tuning

- [z/VM: CP Exit Customization](#), SC24-6269
- [z/VM: Performance](#), SC24-6301

Operation and Use

- [z/VM: CMS Commands and Utilities Reference](#), SC24-6260
- [z/VM: CMS Primer](#), SC24-6265
- [z/VM: CMS User's Guide](#), SC24-6266
- [z/VM: CP Commands and Utilities Reference](#), SC24-6268

- [z/VM: System Operation](#), SC24-6326
- [z/VM: Virtual Machine Operation](#), SC24-6334
- [z/VM: XEDIT Commands and Macros Reference](#), SC24-6337
- [z/VM: XEDIT User's Guide](#), SC24-6338

Application Programming

- [z/VM: CMS Application Development Guide](#), SC24-6256
- [z/VM: CMS Application Development Guide for Assembler](#), SC24-6257
- [z/VM: CMS Application Multitasking](#), SC24-6258
- [z/VM: CMS Callable Services Reference](#), SC24-6259
- [z/VM: CMS Macros and Functions Reference](#), SC24-6262
- [z/VM: CMS Pipelines User's Guide and Reference](#), SC24-6252
- [z/VM: CP Programming Services](#), SC24-6272
- [z/VM: CPI Communications User's Guide](#), SC24-6273
- [z/VM: ESA/XC Principles of Operation](#), SC24-6285
- [z/VM: Language Environment User's Guide](#), SC24-6293
- [z/VM: OpenExtensions Advanced Application Programming Tools](#), SC24-6295
- [z/VM: OpenExtensions Callable Services Reference](#), SC24-6296
- [z/VM: OpenExtensions Commands Reference](#), SC24-6297
- [z/VM: OpenExtensions POSIX Conformance Document](#), GC24-6298
- [z/VM: OpenExtensions User's Guide](#), SC24-6299
- [z/VM: Program Management Binder for CMS](#), SC24-6304
- [z/VM: Reusable Server Kernel Programmer's Guide and Reference](#), SC24-6313
- [z/VM: REXX/VM Reference](#), SC24-6314
- [z/VM: REXX/VM User's Guide](#), SC24-6315
- [z/VM: Systems Management Application Programming](#), SC24-6327
- [z/VM: z/Architecture Extended Configuration \(z/XC\) Principles of Operation](#), SC27-4940

Diagnosis

- [z/VM: CMS and REXX/VM Messages and Codes](#), GC24-6255
- [z/VM: CP Messages and Codes](#), GC24-6270
- [z/VM: Diagnosis Guide](#), GC24-6280
- [z/VM: Dump Viewing Facility](#), GC24-6284
- [z/VM: Other Components Messages and Codes](#), GC24-6300
- [z/VM: VM Dump Tool](#), GC24-6335

z/VM Facilities and Features

Data Facility Storage Management Subsystem for z/VM

- [z/VM: DFSMS/VM Customization](#), SC24-6274
- [z/VM: DFSMS/VM Diagnosis Guide](#), GC24-6275
- [z/VM: DFSMS/VM Messages and Codes](#), GC24-6276
- [z/VM: DFSMS/VM Planning Guide](#), SC24-6277

- *z/VM: DFSMS/VM Removable Media Services*, SC24-6278
- *z/VM: DFSMS/VM Storage Administration*, SC24-6279

Directory Maintenance Facility for z/VM

- *z/VM: Directory Maintenance Facility Commands Reference*, SC24-6281
- *z/VM: Directory Maintenance Facility Messages*, GC24-6282
- *z/VM: Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6283

Open Systems Adapter

- Open Systems Adapter/Support Facility on the Hardware Management Console (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC14-7580-02.pdf), SC14-7580
- Open Systems Adapter-Express ICC 3215 Support (<https://www.ibm.com/docs/en/zos/2.3.0?topic=osa-icc-3215-support>), SA23-2247
- Open Systems Adapter Integrated Console Controller User's Guide (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC27-9003-02.pdf), SC27-9003
- Open Systems Adapter-Express Customer's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/iaa2z1f0.pdf), SA22-7935

Performance Toolkit for z/VM

- *z/VM: Performance Toolkit Guide*, SC24-6302
- *z/VM: Performance Toolkit Reference*, SC24-6303

The following publications contain sections that provide information about z/VM Performance Data Pump, which is licensed with Performance Toolkit for z/VM.

- *z/VM: Performance*, SC24-6301. See *z/VM Performance Data Pump*.
- *z/VM: Other Components Messages and Codes*, GC24-6300. See *Data Pump Messages*.

RACF® Security Server for z/VM

- *z/VM: RACF Security Server Auditor's Guide*, SC24-6305
- *z/VM: RACF Security Server Command Language Reference*, SC24-6306
- *z/VM: RACF Security Server Diagnosis Guide*, GC24-6307
- *z/VM: RACF Security Server General User's Guide*, SC24-6308
- *z/VM: RACF Security Server Macros and Interfaces*, SC24-6309
- *z/VM: RACF Security Server Messages and Codes*, GC24-6310
- *z/VM: RACF Security Server Security Administrator's Guide*, SC24-6311
- *z/VM: RACF Security Server System Programmer's Guide*, SC24-6312
- *z/VM: Security Server RACROUTE Macro Reference*, SC24-6324

Remote Spooling Communications Subsystem Networking for z/VM

- *z/VM: RSCS Networking Diagnosis*, GC24-6316
- *z/VM: RSCS Networking Exit Customization*, SC24-6317
- *z/VM: RSCS Networking Messages and Codes*, GC24-6318
- *z/VM: RSCS Networking Operation and Use*, SC24-6319
- *z/VM: RSCS Networking Planning and Configuration*, SC24-6320

TCP/IP for z/VM

- *z/VM: TCP/IP Diagnosis Guide*, GC24-6328
- *z/VM: TCP/IP LDAP Administration Guide*, SC24-6329
- *z/VM: TCP/IP Messages and Codes*, GC24-6330
- *z/VM: TCP/IP Planning and Customization*, SC24-6331
- *z/VM: TCP/IP Programmer's Reference*, SC24-6332
- *z/VM: TCP/IP User's Guide*, SC24-6333

Prerequisite Products

Device Support Facilities

- Device Support Facilities (ICKDSF): User's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ickug00_v2r5.pdf), GC35-0033

Environmental Record Editing and Printing Program

- Environmental Record Editing and Printing Program (EREP): Reference (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ifc2000_v2r5.pdf), GC35-0152
- Environmental Record Editing and Printing Program (EREP): User's Guide (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ifc1000_v2r5.pdf), GC35-0151

Related Products

XL C++ for z/VM

- *XL C/C++ for z/VM: Runtime Library Reference*, SC09-7624
- *XL C/C++ for z/VM: User's Guide*, SC09-7625

z/OS

IBM Documentation - z/OS (<https://www.ibm.com/docs/en/zos>)

Other TCP/IP Related Publications

This section lists other publications, outside the z/VM 7.3 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. Warriar.
- "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

802.2 LLC frame [195](#)

A

abend
 described [4](#)
 problem category [4](#)
abends
 MPRoute [149](#)
activating traces
 directing output
 to a file [47](#)
 to the screen [47](#)
 first-level trace [45](#)
 second-level trace [46](#)
ALL process [91](#)
applications, functions, and protocols
 FTP [125](#)
 NFS [147](#)
 Remote Printing [185](#)
 REXEC [189](#), [191](#)
 RPC [145](#), [148](#)
 SMTP [137](#), [143](#)
 Telnet [125](#)
ARP
 frame [197](#)
 process [33](#), [49](#), [51](#)
attacks, denial-of-service (DOS) [53](#)

B

Blat attack [53](#)

C

CCS
 process [51](#)
 role in VM structure [15](#)
CCW
 general information [195](#)
 matching traces with TCP/IP traces [200](#)
 samples of CCW traces [196](#), [200](#)
commands
 DUMP [9](#)
 PORT [125](#)
 VMDUMP [9](#)
 VMFPLC2 [9](#)
commonly used trace options [98](#)
congestion process [52](#), [98](#)
CONNECT request [39](#), [40](#)
Connection States
 as know by Pascal/VMCF applications [106](#)
 as know by socket applications [106](#)
 as know by TCP [104](#)

CONSISTENCYCHECKER process [33](#), [52](#)

D

Data Transfer Process (DTP) [125](#)
DEBUG,
 FTP subcommand [127](#)
 NFS subcommand [179](#)
debugging
 in VM
 executing traces [45](#)
denial-of-service (DOS) attacks [53](#)
diagnostic task
 Step 1. Does the problem originate from TCP/IP [1](#)
 Step 2. Try to fix the problem [2](#)
 Step 3. Describe the problem using categories
 abend [4](#)
 documentation [8](#)
 incorrect output [6](#)
 loop [5](#)
 message [4](#)
 performance [7](#)
 wait state [6](#)
 Step 4. Reporting the problem to Service Support [2](#)
 Step 5. Implement the solution [2](#)
directing output
 to a file [47](#)
 to the screen [47](#)
documentation problems [8](#)
dropped [54](#)
Dump Viewing Facility [9](#)

E

error return codes
 UDP [204](#)
EXTERNALHANDLER process [91](#)

F

FILE statement [47](#)
first-level trace [45](#)
Fraggle attack [53](#)
frame
 802.2 LLC [195](#)
 ARP [197](#)
 IP [199](#)
 TCP [199](#)
 token-ring [195](#)
FTP
 client traces
 activating traces [127](#)
 trace output [127](#)
 connection [125](#)
 DEBUG subcommand [127](#)
 DTP [125](#)

FTP (*continued*)
model [125](#)
PI [125](#)
PORT command [125](#)
server traces
 activating traces [132](#)
 trace output [133](#)

G

GATEWAY statement
 use with MPRoute [149](#)
group processes
 ALL [91](#)
 HANDLERS [91](#)
 IUCV [92](#), [94](#)
 PCCA [94](#), [98](#)
 RAWIP [98](#)
 TCP [98](#)
 TCPIP [98](#)
 UDP [98](#)

H

HANDLERS process [91](#)
header
 IP [199](#)
 TCP [199](#)

I

I/O
 IUCV links
 PVM IUCV [39](#)
ICMP process [55](#), [92](#)
IGMP process [55](#), [56](#)
incorrect output problems [6](#)
INITIALIZE process [56](#), [58](#)
internal
 activities [36–39](#)
 procedures [33–35](#)
 queues [35](#), [36](#)
internal tracing statements
 FILE [47](#)
 in TCPIP.PROFILE.TCPIP [45](#)
 LESSTRACE [46](#), [49](#), [91](#)
 MORETRACE [46](#), [49](#), [91](#)
 NOTRACE [46](#), [49](#), [91](#)
 SCREEN [47](#)
 TRACE [45](#), [49](#), [91](#)
Internet
 protocols, ICMP [92](#)
IOHANDLER process [91](#)
IP
 frame [199](#)
 header [199](#)
IPDOWN process [33](#), [58](#), [98](#)
IPFORMAT [109](#)
IPREQUEST process [98](#)
IPUP process [33](#), [59](#), [98](#)
IUCV
 links
 PVM [39](#)

IUCV (*continued*)
 process [92](#), [94](#)
 role in VM structure [15](#)
 trace output [92](#)
IUCVHANDLER process [91](#)

K

Kiss-of-Death (KOD) attack [53](#)
KOX attack [53](#)

L

LAN
 messages [194](#)
 support devices for [193](#)
Land attack [53](#)
LDSF
 role in VM structure [15](#)
LESSTRACE statement [46](#), [49](#), [91](#)
LLC [195](#)
loop problems [5](#)

M

machine readable documentation guidelines [9](#)
message examples, notation used in [xx](#)
message problems [4](#)
MONITOR process [33](#), [59](#), [61](#)
MORETRACE statement [46](#), [49](#), [91](#)
MPRoute
 abends [149](#)
 client cannot reach destination [150](#)
 connection problems [149](#)
 overview [149](#)
MULTICAST process [61](#), [62](#)

N

netstat command
 MPRoute problem diagnosis [150](#)
NETSTAT OSAINFO [200](#)
NFS
 activating traces [179](#)
 function [147](#)
 trace output [180](#)
NOPROCESS process [62](#)
notation used in message and response examples [xx](#)
NOTIFY process [33](#), [62](#), [64](#), [98](#)
NOTRACE statement [46](#), [49](#), [91](#)

O

OBEYFILE [45](#), [65](#)
open shortest path first (OSPF) [149](#)
OSA address table [200](#)
OSD process [64](#)
OSPF (open shortest path first) [149](#)
output,
 directing to a file [47](#)
 directing to the screen [47](#)
 problem category [6](#)

P

- PARSE-TCP process [65](#)
- Pascal [33–35](#)
- PCCA
 - CCW
 - general information [195](#)
 - matching traces with TCP/IP traces [200](#)
 - samples of CCW traces [196, 200](#)
 - devices [193, 200](#)
 - PCCA block structure
 - 802.2 LLC frame [195](#)
 - control messages [193](#)
 - general information [193](#)
 - information about token-ring frames [195](#)
 - LAN messages [194](#)
 - process [94, 98](#)
- performance problems [7](#)
- PING command [42](#)
- PING process [42](#)
- Ping-o-Death attack [53](#)
- PORT command [125](#)
- Portmapper [148](#)
- preface [xvii](#)
- problem categories
 - abend [4](#)
 - documentation [8](#)
 - incorrect output [6](#)
 - loop [5](#)
 - message [4](#)
 - performance [7](#)
 - wait state [6](#)
- processes
 - group
 - ALL [91](#)
 - HANDLERS [91](#)
 - IUCV [92, 94](#)
 - PCCA [94, 98](#)
 - RAWIP [98](#)
 - TCP [98](#)
 - TCPIP [98](#)
 - UDP [98](#)
 - single
 - ARP [33, 49, 51](#)
 - CCS [51](#)
 - CONGESTION [52, 98](#)
 - CONSISTENCYCHECKER [33, 52](#)
 - EXTERNALHANDLER [91](#)
 - ICMP [55, 92](#)
 - IGMP [55, 56](#)
 - INITIALIZE [56, 58](#)
 - IOHANDLER [91](#)
 - IPDOWN [33, 58, 98](#)
 - IPREQUEST [98](#)
 - IPUP [33, 59, 98](#)
 - IUCVHANDLER [91](#)
 - MONITOR [33, 59, 61](#)
 - MULTICAST [61, 62](#)
 - NOPROCESS [62](#)
 - NOTIFY [33, 62, 64, 98](#)
 - OSD [64](#)
 - PARSE-TCP [65](#)
 - PING [65, 92](#)
 - QDIO [67](#)

- processes (*continued*)
 - single (*continued*)
 - RAWIPREQUEST [33, 98](#)
 - RAWIPUP [98](#)
 - RETRANSMIT [98](#)
 - REXMIT [98](#)
 - ROUNDTRIP [67, 98](#)
 - SCHEDULER [33, 67](#)
 - SHUTDOWN [33, 69](#)
 - SNMPDPI [70](#)
 - SOCKET [70](#)
 - STATUSOUT [33](#)
 - TCPDOWN [33, 72, 73, 98](#)
 - TCPREQUEST [33, 77, 79, 98](#)
 - TCPUP [33, 73, 77, 98](#)
 - TELNET [79, 86](#)
 - TIMER [33, 86](#)
 - TOIUCV [33](#)
 - UDPREQUEST [33, 88, 98](#)
 - UDPUP [90, 98](#)
- PROFILE TCPIP [45, 65](#)
- Protocol Interpreter (PI) [125](#)
- Pseudo-state, connection
 - CONNECTIONclosing [106](#)
 - LISTENING [106](#)
 - NONEXISTENT [106](#)
 - OPEN [106](#)
 - RECEIVINGonly [106](#)
 - SENDINGonly [106](#)
 - TRYINGtoOPEN [106](#)
- PVM
 - CONNECT request [39, 40](#)
 - local [40](#)
 - remote [39](#)

Q

- QDIO process [67](#)
- queues [35, 36](#)

R

- R4P3D attack [53](#)
- RAWIP process [98](#)
- RAWIPREQUEST process [33, 98](#)
- RAWIPUP process [98](#)
- related protocols [207](#)
- remote printing
 - client traces
 - activating traces [185](#)
 - trace output [185](#)
- response examples, notation used in [xx](#)
- RETRANSMIT process [98](#)
- return codes
 - TCP/IP [203](#)
 - UDP Error [204](#)
- REXEC
 - activating traces [189](#)
 - trace output [189](#)
- REXECd
 - activating traces [190](#)
 - trace output [191](#)
- REXMIT process [98](#)

- RIP (routing information protocol)
 - MPRoute implementation [149](#)
- ROUNDTRIP process [67](#), [98](#)
- routing information protocol (RIP)
 - MPRoute implementation [149](#)
- RPC programs
 - call messages [145](#)
 - function [145](#)
 - Portmapper [148](#)
 - reply messages
 - accepted [146](#)
 - rejected [147](#)
 - support [148](#)

S

- SCHEDULER process [33](#), [67](#)
- SCREEN statement [47](#)
- second-level trace [46](#)
- SHUTDOWN process [33](#), [69](#)
- SMSG command
 - with MPRoute [151](#)
- SMTP
 - client traces
 - activating traces [137](#)
 - querying SMTP queues [137](#)
 - server traces
 - activating traces [138](#)
 - commands [138](#)
- Smurf-IC attack [53](#)
- Smurf-OB attack [53](#)
- Smurf-RP attack [53](#)
- SNMPDPI process [70](#)
- SOCKET process [70](#)
- SSL
 - Diagnosing problems [163](#)
 - trace output [169](#)
- state, connection
 - CLOSE-WAIT [105](#)
 - CLOSED [105](#)
 - CLOSING [105](#)
 - ESTABLISHED [104](#)
 - FIN-WAIT-1 [104](#)
 - FIN-WAIT-2 [105](#)
 - LAST-ACK [105](#)
 - LISTEN [104](#)
 - SYN-RECEIVED [104](#)
 - SYN-SENT [104](#)
 - TIME-WAIT [105](#)
- statements
 - FILE [47](#)
 - GATEWAY [42](#)
 - LESSTRACE [46](#), [49](#), [91](#)
 - MORETRACE [46](#), [49](#), [91](#)
 - NOTRACE [46](#), [49](#), [91](#)
 - SCREEN [47](#)
 - TRACE [49](#), [91](#)
- STATUSOUT process [33](#)
- Stream attack [53](#)
- Synflood attack [53](#)
- syntax diagrams, how to read [xvii](#)

T

- TCP
 - frame [199](#)
 - header [199](#)
 - process [98](#)
- TCP/IP
 - internal
 - activities [36–39](#)
 - procedures [33–35](#)
 - queues [35](#), [36](#)
 - matching traces with CCW traces [200](#)
 - nodes, failure to connect [41](#), [43](#)
 - return codes [203](#)
- TCPDOWN process [33](#), [72](#), [73](#), [98](#)
- TCPIP
 - process [98](#)
- TCPREQUEST process [33](#), [77](#), [79](#), [98](#)
- TCPUP process [73](#), [77](#), [98](#)
- Telnet
 - failure to connect [41](#), [43](#)
 - process [79](#), [86](#)
- TIMER process [33](#), [86](#)
- TOIUCV process [33](#)
- token-ring [195](#)
- trace
 - first-level [45](#)
 - FTP
 - client [127](#), [132](#)
 - server [132](#)
 - IUCV [92](#)
 - remote printing [185](#)
 - REXEC [189](#), [190](#)
 - REXECD [190](#), [191](#)
 - second-level [46](#)
 - SMTP
 - client [137](#)
 - server [138](#), [143](#)
 - TCPIP [98](#)
 - Telnet [79](#)
- TRACE statement [45](#), [49](#), [91](#)
- TRACERTE command [107](#)
- traces, dropped [54](#)
- trademarks [214](#)

U

- UDP
 - error return codes [204](#)
- UDPREQUEST process [33](#), [88](#), [98](#)
- UDPUP process [90](#), [98](#)

V

- virtual machines [13](#)
- VM
 - debugging
 - executing traces [45](#)
 - structure
 - CCS and LDSF [15](#)
 - IUCV [15](#)
 - virtual machines [13](#)
 - VMCF [14](#)

VMCF

role in VM structure [14](#)

W

wait state problems [6](#)

worksheet for reporting problems [11](#), [12](#)



Product Number: 5741-A09

Printed in USA

GC24-6328-73

