DTF (DIGITAL Trace Facility) User Guide

Order Number: AA-R85DA-TE

Revision/Update Information:	This is a new manual
Software Version:	DTF V2.5

First Printing, August 1997

Preface

Overview

This manual describes how to install and use the DIGITAL Trace Facility (DTF) software. This is a host-based facility that allows you to trace packets within the protocol layers of a DTF-capable router.

Audience

This manual is intended for network managers.

The manual assumes that you understand and have some experience of multiprotocol networks.

Returning Comments About this Documentation

We would like to know what you think about this document and associated online help.

If you have any comments, or suggestions, please return them in any of the following ways:

- Send an electronic mail message to the Internet address books@reo.mts.dec.com
- Send an electronic mail message to the X.400 address S=IDC BOOKS; O=digital; OU1=reo; P=digital; A=CWMail; C=gb
- Send a fax to (+44)134 206018

Chapter 1

Introduction

1.1 Abstract

DTF is a facility which traces packets as they traverse the protocol layers within a router. DTF is a superset of Digital's CTF (Common Trace Facility), which supports multiple platforms and TCP/IP networks.

1.2 What Is DTF?

DTF is a host-based facility which allows packets to be traced as they traverse the protocol layers within a DTF-capable router. The packets can be analysed "live" as they are processed by each router software module, or they can be recorded in a data file and analysed later. DTF can be used in either DECnet™ or TCP/IP networks, and supports the following host platforms:

- Digital UNIX™ Alpha (formerly called DEC OSF/1)
- Alpha Linux® V1.2.8
- Intel® Linux V1.2.8
- OpenVMS™ VAX™ V6.0
- OpenVMS Alpha V6.1
- Windows 95®, all releases
- Windows NT™, V3.5-1 and above

(The Windows version of DTF (WinDTF) uses a graphical user interface, rather than a command-line interface.)

DTF supports the following routers:

- DEC RouteAbout™ family and DECSwitch 900EF (software version 2.0 or later required, TCP access only).
- DECNIS™ 500/600 (software version 3.1 or later required for TCP

access)

- DEC WANrouter 90

There is a bug in versions 1.3.2 and earlier of the WANrouter software, such that only a capture size of 188 and a buffer size of 512 are guaranteed to work (other sizes may or may not work). Since by default DTF uses larger sizes than these, you will need to set the capture and buffer size explicitly when tracing a WANrouter 90.

1.3 Tracepoints

The points within a router which can be traced are known as Tracepoints. Each protocol module within the router may have 0 or more tracepoints, for example, the PPP data link module may have a tracepoint for each interface. A tracepoint is either active or inactive. Whenever a packet passes through an active tracepoint it is traced. The tracepoints specified by a user are activated when DTF connects to the router, and de-activated when DTF disconnects from the router. Tracepoints are named according to a class/instance scheme.

The general form of a tracepoint name is:

class [instance] [[sub-class instance]...]

where:

- **class** is the node class. Typically, this is the name of the protocol module, for example, ROUTING, PPP, MODEM_CONNECT.
- **sub-class** is a specific subclass of the node class. Typically this might be a LINE, CIRCUIT or INTERFACE.
- **instance** is the entity instance name that uniquely identifies the particular tracepoint from others in the same sub-class.

The node class may have a null instance name but each sub-class must be followed by a sub-class instance name. For example, to trace the packets passing through the ROUTING module on the interface named interface-0, the tracepoint would be named:

class = ROUTING

sub-class = CIRCUIT

instance = Interface-0

That is,

ROUTING CIRCUIT Interface-0

The instance name can contain the wildcard character *, which will cause any tracepoint with a matching instance name to be traced. All known tracepoints for the sub-class can be traced by specifying a single * as the instance name. The class and sub-class names can be abbreviated; DTF will take the first matching valid name it finds. The tracepoints are defined in the DTF_TRACEPOINTS.DAT file. Note that this filename may be different on other platforms.

1.3.1 Example Tracepoints

The following are some example tracepoints. The wildcard forms are shown but you could substitute a specific tracepoint name for the * character.

- ROUTING CIRCUIT *.

  Trace all the routing circuits on the router.

- PPP LINK *.

  Trace all the PPP links on the router.

- ETHERNET INTERFACE Eth/*

  Trace all the Ethernet interfaces whose names match the wildcard string Eth/* (that is, Eth/0, Eth/1, etc.).

- NSP PORT *.

  Trace all the NSP ports on the router. Since NSP ports are created dynamically when an NSP connection is created these tracepoints may appear after the DTF trace session has started and may be deleted during the session.

1.3.2 Multiple Tracepoints

With version 2.1 of DTF, up to 4 tracepoints can be specified for the same trace session. DTF instructs the router to activate each tracepoint at the start of the trace session. The output from all tracepoints is displayed in the same order in which the data was traced.

1.4 Events

Each tracepoint defines a number of events (up to 64) and each packet traced is marked with one of these events by the router. When DTF initially connects to the router it can optionally instruct the router to filter the packets passing through the specified tracepoint and only trace the ones which match the specified events list. Typically, a tracepoint would define an event for a transmitted packet and another for a received packet. However, it may also define events for particular packet types (e.g. the "ROUTING CIRCUIT *" tracepoint has a unique event for OSPF, ARP, OSI and RIP packets). The events for each tracepoint are defined in the DTF_EVENTS.xxx files (where xxx is the id of the router, by default an id of 000 is assumed). Each router platform may support a different set of events for any particular tracepoint. Note that this filename may be different on other platforms. All the events files (including the router identifier to name mapping file DTF_EVENTS.DAT) must be in the same directory.

1.4.1 Command-Line Interface

By default, events are passed (that is, when a particular event is specified on the command line, it is traced; if it is not specified, it is not traced). However, an event can be blocked by prefixing its name with the "!" character. The special event name "*" is used to mean "all events". If a filter list (event list) is not specified, then DTF acts as if the "*" event had been specified (that is, all events are traced). If an event list is supplied, then DTF first initializes the event list to block all events before processing the list. Therefore in order to specify any blocking events, all events must first be turned back on.

When multiple tracepoints are used, the user interface allows a separate event list to be specified for each of the tracepoints.

1.4.2 Example Events

The following are example events:

- "ospfRx" - Trace only received OSPF packets
- "* !ospfRx !ospfTx" - Trace everything except OSPF packets
- "*" - Trace everything (this is equivalent to not supplying an events list)
- "!*" - Trace nothing (this is equivalent to an empty events list, not very useful)

1.5 Names

DTF stores its value-to-name translation tables in the DTF_NAMES.DAT file. These tables provide name translations for such things as MAC addresses, Ethernet OUIs, UDP/TCP ports, etc. Users can supplement these name tables by providing a DTF_LOCAL_NAMES.DAT file (which must reside in the same directory as the DTF_NAMES.DAT file) which contains additions or replacements for the tables. The format for the names file is described in the comments in the file itself. The following is an example of a local names file which defines the node names of three locally used MAC addresses and a definition for the UDP and TCP ports used by a local application.

```
#

# dtf_local_names.dat

#

[UnicastMacNames]! MAC address to Name

08-00-2B-B6-D8-E8     “JohnsNode”

08-00-2B-B3-6C-5A     “tonysNode”


[UdpPorts       ! UDP ports to Names]

5999        “My Application”


[TcpPorts]      ! TCP ports to Names

5999       “My Application”
```

1.6 How Does It Work ?

DTF uses a transport protocol (TCP or DECnet) to set up a reliable data connection between the host (running DTF) and the router. When the connection is made, DTF sends the parameters to use for the trace session to the router together with a list of tracepoints which are to be activated.

The router allocates the resources needed for the session (based on the parameters supplied by the host) and activates the tracepoints if they are available. Any tracepoints which are not available immediately will be activated later when they become available (assuming the trace session is still alive). Once this has been done, the trace data passing through an activated tracepoint is copied into a trace buffer (if one is available) and if the trace buffer is full it is queued for transmission to the host. When a transmit opportunity arises, the contents of the trace buffer are sent to the host where it is decoded and displayed (and possibly recorded), and the trace buffer is again available to be used for tracing.

1.6.1 Session Parameters

The following parameters are given to the router by the host at the start of the session. These parameters determine how much router resource is allocated to the trace session and how much of the data passing through the tracepoints is captured.

BUFFER COUNT specifies the number of trace buffers to be used to capture trace data during this session. The more trace buffers there are available, the less trace data will be lost.

BUFFER SIZE specifies the size (in bytes) of the trace buffers. This value governs the maximum amount of a data packet that can be traced, since data packets are not generally split across trace buffers.

CAPTURE SIZE specifies the number of bytes in the data packet that should be copied into the trace buffer. This determines how much of each data packet is traced.

1.6.2 Minimizing Trace Data Loss

Trace data loss occurs when there are no more trace buffers available to trace the next data packet, and so the packet is not traced. The following factors can reduce the amount of trace data lost. They are listed in order of effectiveness (that is, the ones most likely to reduce the data loss are listed first). Note, however, that for high throughput data it may be impossible to prevent data loss entirely.

FILTERS. Using filters restricts the amount of data being traced by tracing only the data packets which match the specified events list.

BUFFER COUNT. Increasing this parameter allows more trace data to be buffered within the router before trace data loss occurs. Note that allocating more buffers to the trace session will mean less buffers being available to the rest of the router.

CAPTURE SIZE. Reducing the number of bytes in each packet that are traced means that more packets can be contained in each trace buffer, effectively increasing the buffering available.

BUFFER SIZE. Increasing the size of the trace buffers again means more buffering available to the trace system. Note that this parameter may not have any effect on routers that use fixed buffer sizes.

RECORD ONLY. Most of the delays introduced at the host end are by terminal I/O. Therefore, not displaying (and hence not decoding) the trace data as it arrives can speed up the processing on the host end and possibly increases the effective throughput of the DTF transport connection.

Chapter 2

Installation Instructions and Platform Differences

2.1 OpenVMS and OpenVMS Alpha

DTF is supplied as a saveset (DTFVMS.A for OpenVMS VAX and DTFAXP.A for OpenVMS Alpha). This saveset contains the files described in Table 2-1. The OpenVMS kit supplies a CLD file which provides a standard DCL-style command line interface to DTF. The common UNIX-style interface can still be accessed by supplying the /UNIX qualifier immediately after the DTF command, as follows:

```
$ DTF/UNIX unix-command-line
```

Table 2-1: OpenVMS File List

Filename	Location	Description
DTF.EXE	SYS$SYSTEM	DTF image, activated by the DCL DTF command. You may place this file in another directory, provided that you define the DTF logical to point to it.
DTF_VMS_COMMANDS.CLD	SYS$SYSTEM	DTF CLI definition.
DTF_TRACEPOINTS.DAT	SYS$LIBRARY	Tracepoints file, containing the list of supported tracepoint names. You may place this file in another directory.
DTF_EVENTS.DAT	SYS$LIBRARY	Events file, contains the mapping between router identifiers and names. You may place this file in another directory provided that you define the DTF_EVENTS logical to point to it.
DTF_EVENTS.000	SYS$LIBRARY	Events file, contains the list of supported events for each tracepoint for the DECNIS and WANrouter 90 routers (router identifier 000). You may place this file in another directory provided that you define the DTF_EVENTS_000 logical to point to it, otherwise this file must reside in the same directory as the DTF_EVENTS.DAT file.
DTF_EVENTS.001	SYS$LIBRARY	Events file for the DEC RouteAbout family of routers (see DTF_EVENTS.000 row for details).

Filename	Location	Description
DTF_NAMES.DAT	SYS$LIBRARY	Names file, contains a list of name translation tables. You may place this file in another directory provided that you define the DTF_NAMES logical to point to it, otherwise this file must reside in the same directory as the DTF_EVENTS.DAT file. Users may add to the name tables by supplying their own in a file named DTF_LOCAL_NAMES.DAT, which must reside in the same directory as the DTF_NAMES.DAT file.
DTF.HTML	SYS$HELP	DTF Release Notes and installation instructions in HTML format.
DTFUG.PS	SYS$HELP	DTF User Guide.

NOTE

DTF can be installed in a private directory (rather than system wide), provided that the DTF, DTF_TRACEPOINTS and DTF_EVENTS logicals are defined to point to the corresponding EXE and DAT files.

2.1.1 Command-Line Interface

The DCL command-line interface to DTF maps the internal UNIX-style options to command-line qualifiers and parameters. Table 2-2 describes the mapping.

The general format for a 'live' trace is:

```
$ DTF/LIVE [qualifiers] node-name"username
password"::"tracepoint,...”
```

For an analysis of a pre-recorded data file, it is:

```
$ DTF/ANALYZE [qualifiers] [data-file]
```

For pipe mode, it is:

```
$ DTF/PIPE/TYPE=packet-type ["hex-string" | /INPUT=input-
file]
```

Table 2-2: VMS Command Line Syntax

OpenVMS	Corresponding UNIX Option
/UNIX	Rest of the line is in internal UNIX format.
/ANALYZE	-a
/BACKWARDS	-B
/BUFFER	-b
/CAPTURE=size	-c
/{NO}COLORING	-C
/DATA={ASCII|HEX|NONE|<type>} (D=HEX)	-Z
/EXCLUDE="regexp"	-x
/FILTER="filter-list,..."	-f
/FULL	-v
/{NO}HEADERS	-H
/{NO}HIGHLIGHT	-D
/INPUT=input-file	-I
/LIVE	-l
/MAXIMUM_BUFFERS=number	-m
/NARROW _STYLE	-N
/OUTPUT=file	-o
/PAGE=number (D=0)	-w
/PIPE	-P
/PROTOCOL="protocol,..."	-p
/RAW	-r
/RECORD=file	data-file parameter
/RELATIVE_TIME	-R
/SEARCH="regexp"	-s

OpenVMS	Corresponding UNIX Option
/TYPE=packet-type	-T
/TRANSLATE	-t
/WIDTH=number (D=0)	-W

2.1.2 Restriction: DECnet Addresses

In the OpenVMS version of DTF, translation of DECnet addresses to node names is not supported.

2.2 Digital UNIX, ULTRIX and Linux

DTF is supplied as a tar file, DTFULTRIX.TAR, DTFOSF.TAR or DTFLINUX.TAR, which contains the files described in Table 2-3.

Table 2-3: UNIX File List

Filename	Location	Description
dtf	/usr/dtf	DTF image.
dtf_tracepoints.dat	/usr/dtf	Tracepoints file, contains the list of supported tracepoint names. You may place this file in another directory provided that you define the DTF_TRACEPOINTS environment variable to point to it.
dtf_events.dat	/usr/dtf	Events file, contains the mapping between router identifiers and names. You may place this file in another directory provided that you define the DTF_EVENTS environment variable to point to it.
dtf_events.000	/usr/dtf	Events file, contains the list of supported events for each tracepoint for the DECNIS and WANrouter 90 routers (router identifier 000). You may place this file in another directory provided that you define the DTF_EVENTS_000 environment variable to point to it,otherwise this file must reside in the same directory as the dtf_events.dat file.
dtf_events.001	/usr/dtf	Events file for the DEC RouteAbout family of routers (see the dtf_events.000 row for details).

Filename	Location	Description
dtf_names.dat	/usr/dtf	Names file, contains a list of name translation tables. You may place this file in another directory provided that you define the DTF_NAMES environment variable to point to itt. Otherwise this file must reside in the same directory as the DTF_EVENTS.DAT file. Users may add to the name tables by supplying their own in a file named dtf_local_names.dat, which must reside in the same directory as the dtf_names.dat file.
dtf.html	/usr/dtf	DTF Release Notes and installation instructions in HTML format.
dtfug.ps	/usr/dtf	DTF User Guide.

NOTE

DTF can be installed in a private directory (rather than system wide) provided that the DTF_TRACEPOINTS and DTF_EVENTS environment variables are defined to point to the corresponding DAT files.

2.3 Windows 95/NT

WinDTF is supplied as part of the clearVISN DECNIS Configurator and clearVISN Router Configurator, which are freely available on the Internet (start at http://www.networks.digital.com). As the name suggests, it has a windows interface rather than a command-line interface. Once the configurator is installed, WinDTF can be started by selecting it from the 'Tools' menu on the Browser. On-line help is then available to explain how to select tracepoints, filters, and so on.

2.3.1 Restrictions

The following restrictions apply to the Windows 95/NT platform version of DTF:

- DECnet access is not supported. The DTF connection must be made using TCP/IP.
- Translation of DECnet addresses to node names is not supported.
- Recording of trace data to a binary file, without simultaneously analysing it, is not supported.
- Negated events such as '!ospfRx' are not explicitly supported.
- VNSwitch tracepoints are not supported.

Chapter 3

Common Command Line and Configuration

3.1 Command-Line Interface

DTF uses a UNIX-style command-line interface internally. For the Digital UNIX and ULTRIX platforms, this is also the interface which is presented to the user. For OpenVMS, the user is presented with a DCL command-line interface. For Windows 95/NT there is no command-line interface; instead the user is presented with a Windows-style interface. This section documents the UNIX command-line interface.

DTF performs two main functions:

- Analysis of trace data. This function analyses the data from either the 'live' system or from a previously recorded data file.
- Recording of trace data. This function saves the trace data in a file so that it can be analysed later.

These functions can be performed simultaneously. The format for the command line changes depending on whether the source of the trace is a router or a previously recorded data file. When the source is the router, the format is:

```
dtf [options] nodename[/username/
password]:[:]"tracepoint,..." [data-file]
```

When the source is a data file, the format is:

```
dtf -a [options] [data-file]
```

When the source is an input file or a pipe, the format is:

```
dtf -P -T"packet-type" [options] ["hex-string" | -I"input-
file"]
```

Table 3-1 shows the command-line parameters. Table 3-2 shows the command-line options.

Table 3-1: UNIX Command-Line Parameters

Parameter	Default	Description
nodename		Node name or address (IP or DECnet dotted format) of router including username/password access control and the tracepoint name list.
data-file	dtf.dat	Name of the file in which to record the trace data. This parameter is optional when analysing a 'live' system, so if it is not supplied, then the trace data will not be recorded. The data file can also be specified in the DTF_DATA environment variable.

Table 3-2: UNIX Command-Line Options

Option	Parameter	Description
-a		Analyse data file. The data file name is supplied as the first parameter.
-b	size	Buffer size the router is to use for its trace buffers. The default is 1569.
-c	size	Capture size the router is to use. This is the number of bytes in each packet which should be traced. The buffer size (-b option) will be adjusted accordingly. The default is 1500.
-f	"f1 f2 f3, ..."	Filter list. This is a space (or semi-colon) separated list of the filters to apply to each tracepoint in the tracepoint list. By default, no filters are applied and so all packets passing through the tracepoint are traced. Filter names may be abbreviated; DTF will take the first matching filter name it finds in the Events file. When using multiple tracepoints, this parameter consists of a comma separated list of filter lists (one filter list for each tracepoint).
-h		Output brief help information.
-l		Live tracing. This option causes the trace data to be analysed and displayed as it is received. This option and the -a option are mutually exclusive.
-m	number	Maximum number of trace buffers the router should allocate. The default is 5.
-o	file	Output filename. The default is to display the analysed data to stdout.

Option	Parameter	Description
-p	“protocol, ...”	Specifies the protocol header tp be displayed. By default, all protocol headers are displayed. When specified, this switch causes DTF to display only those packets which contain the specified protocol header and to suppress all other headers in the packet. When using multiple tracepoints, this parameter consists of a comma separated list of protocol names (one for each tracepoint). The following protocols are supported: 802.2, 802.3, AARP, APPLETALK, APPLICATION, ARP, ASN1, BGP, BOOTP, BRIDGE, CHDLC, CMIP, DATALINK, DDCMP, DECNET, ETHERNET, FR, FDDI, HDLC, ICC, ICMP, IGMP, IP, IPX, LAPB, LAPBE, LAT, LLC2, MOP, NSP, OSI, OSPF, PIM, PPP, PRAM, PSL, RIP, SNMP, TCP, TRANSPORT, UDP, X25L3.
-r		Raw data mode. The output data is not formatted when this option is supplied.
-s	"regexp"	Regular expression match. Only if the output data matches the regexp is it displayed; otherwise the trace output is discarded.
-t		Translate addresses. Any IP or DECnet addresses in the output data will be translated to node names before being displayed.
-v		Verbose mode. When supplied, the packet is analysed fully and a detailed output produced. When absent, the output is brief and restricted to a single line for each packet.
-w	number	Wait mode. Causes the output to pause between packets (and prompt the user to press Return before continuing). In brief mode, the parameter specifies the number of lines to be output before the prompt is issued; in verbose mode, the parameter is ignored and the prompt is issued after every packet. Supplying a value of 0 in brief mode is equivalent to supplying a value of 22.
-x	"regexp"	Regular expression no-match. Only if the output data fails to match the regexp is it displayed; otherwise the trace output is discarded. This option and the -s option act in conjunction, so if both are supplied then the output is only displayed if the -s regular expression matches AND the -x regular expression fails.

Option	Parameter	Description
-z		Unsupported, use -Za instead (see below).
-B		The data from the input stream is presented backwards (i.e. the last byte in the packet is first and the first byte in the packet is last). (Pipe mode only.)
-C		Suppresses coloring of output. If coloring is suppressed then some portions of the output which would otherwise have been coloured will be bolded. Use this option if your terminal does not support ANSI colors since that will cause the headers to be highlighted instead. Colors can be modified us-ing the DTF_COLORS environment variable.
-D		Suppresses highlighting and coloring of output.
-H		Suppresses the DTF headers on output.
-I	input-file	Source of the hex string data is the input-file. The data is stored as hex strings in the input-file. (Pipe mode only.)
-N		Output is in narrow form. The default is wide, which assumes a 132-column output display.
-P		Pipe mode. In this mode, the data is presented as hex strings, either as the first parameter on the command line or in the input-file (see -I switch). End of packet is denoted by the end of the file or a newline. Packets may be continued onto another line by terminating the current line with a '\' character.
-R		Time is displayed as relative to the last trace record received. If this switch is not present, time is displayed as an absolute time as received from the router. The time on the first packet is always shown as an absolute time.

Option	Parameter	Description
-T	packet-type	The type of packet presented in the hex data string. The type name can be abbreviated; DTF will take the first matching type it finds (the valid types are searched alphabetically). The following types are supported: ARP, ASN1, BRIDGE , CMIP, CSMA-CD, DDCMP, FDDI, FDDIcanonical,FR,HDLC, ICMP, IP, IPX, LAPB, LAPBE, MOP, NSP, OSI, OSPF, PPP, PSL (Proteon Serial Link) , PhaseIV, RIP, SNMP, TCP, UDP, X25L3 (Pipe mode only.)
-Z	One of: a h 0 packet-type	Specifies the output format for undecoded data. By default, this data is printed as a series of hex bytes. a = output bytes as ASCII h = output bytes as HEX 0 = suppress undecoded data output packet-type = initiate a second decode using the string as a packet-type indication (see -T). This allows undecodable client data to be automatically parsed, for example, -Zcmip would allow an NSP session carrying management traffic to be completely decoded.

3.1.1 Example Command Lines

The following are example command lines and their use:

- `dtf -l nis100:"routing circuit *"`

  Trace all Routing circuits on node NIS100 (use TCP as the transport protocol), analyse the data (in brief mode) as it arrives and display it on the standard output device.

- `dtf -l -f"ospfTx;ospfRx" nis100:"routing circuit *"`

  As above, but now apply a filter set which excludes all packets except transmitted and received OSPF packets.

- `dtf -l -f"ospftx;ospfrx" nis100/x/y:"routing circuit *" ospf.dat`

  As above, but now supply a username and password of x and y and also record the trace records to a file called ospf.dat.

- `dtf -lvw0 -o output.txt 1.100::"ppp link l601-8-0"`

  Trace all PPP links on node 1.100 (use DECnet as the transport protocol), analyse the data (in verbose mode) as it arrives and write it to the file output.txt.

- `dtf nis100:"ppp link *" ppp.dat`

  Trace all PPP links on node NIS100 (use TCP as the transport protocol), and record the data in the file ppp.dat.

- `dtf -avtw0 ppp.dat`

  Analyse the data in the previously recorded ppp.dat file, use verbose format output, translate addresses to names and pause between every 22 lines of output.

- `dtf -a`

  Analyse the data in the previously recorded dtf.dat file, use brief format output.

- `dtf -a -s"BGP:"`

  As above, but now filter the output such that only packets containing the string "BGP:" are displayed.

- `dtf -a -s"BGP:" -x"KEEPALIVE"`

  As above, but now filter the output further to exclude packets which contain the string KEEPALIVE.

- `dtf -P -T"ip" "45C0005400170000"`

  Pipe mode. The supplied hex string is decoded as an IP packet.

- `dtf -P -T"ip" -I"in.txt"`

  As above, but now the hex strings are contained in the file in.txt.

- `dtf -l -f”ospxtx;ospfrx,,iptx,iprx” nis100:”ospf i *,ether i *,fddi i *”`

  This example demonstrates the use of multiple tracepoints. Three tracepoint classes are being traced: all the OSPF interfaces, all the Ethernet interfaces and all the FDDI interfaces. The ospftx and ospfrx filters are applied to the OSPF tracepoints, no filters are applied to the Ethernet tracepoints, and the ipTx and ipRx filters are applied to the FDDI tracepoints.

3.2 Supplying Node Names and Addresses

The format of the router node name supplied determines which transport protocol is used to connect to the router, what the username and password is, and also the tracepoint to be traced. The general format is:

```
node[/username/password]:[:]"tracepoint, ..."
```

When an address is supplied (DECnet or IP), DTF determines from the number of dots (.) present whether to use DECnet or TCP. When a node name is supplied, DTF uses TCP by default if the name is followed by a single colon character, and DECnet if the name is followed by a double colon. For example, the following formats will cause the connection to be made via DECnet:

```
1.100

nis100::
```

The following will cause the connection to be made via TCP:

```
16.36.16.100

nis100.reo.dec.com

nis100:

nis100
```

The following example shows a complete nodename specifier for tracing the tracepoint "ROUTING CIRCUIT *" on node nis100 using TCP as the transport protocol:

```
nis100:"routing circuit *"
```

The next example is the same as the one above but this time a username and password are supplied and the transport used is DECnet.

```
nis100/me/hushhush::"routing circuit *"
```

3.3 Output Customization

Most customization is done by defining environment variables. The following sections define these variables.

3.3.1 Output Coloring

On terminals which support ANSI color escape sequences, DTF will color various fields in the output. The supported colors (and highlighting options) are listed in Table 3-3.

Table 3-3: ANSI Color Escape Sequences

Color (Option)	ANSI Value
Default	0
Bold	1
Underline	4
Flashing	5
Reverse	7
Black	30
Red	31
Green	32
Blue	33
Yellow	34
Purple	35
Cyan	36
White	37

The DTF output can be grouped into named fields, and each field can have a different color associated with it. Table 3-4 describes the output field names and the default color used for that field.

Table 3-4: Output Field Names and Default Colors

Field	Default	Description
Headers	Green	The DTF header, from the -DTF- prefix through to the underline.
Protocol-headers	Cyan	The protocol section headers.
Search-strings	Reverse	Highlighted search strings.
Comments	Green	Comments. (Used in Pipe mode only.)
Tx-events	Normal	The body of the output for all event names which end with the tx string, for example, OspfTx.
Rx-events	Normal	The body of the output for all event names which end with the rx string, for example, OspfRx.
Labels	Green	Field labels within the body of the output.
Strings	Normal	The body of the output which does not come under any of the above categories.

The colors can be modified by defining the DTF_COLORS environment variable. The variable is defined to be a string of the form:

"field1=foreground-color;background-color;option1;option2,field2=..."

where field is one of the field names described in Table 3-4 and foreground-color, background-color, option1, option2 are one of the color or option names, or the numerical ANSI value for the color or option. Only the foreground color need be defined; the others can be omitted. The following example shows the string required to color all rx events in white and to remove the coloring of protocol headers:

"rx-events=white,protocol=0"

Both the field and color names may be abbreviated.

3.3.2 Separator Character

The separator character used to separate the label field from the value field can be changed by defining the DTF_SEPARATOR environment variable to the desired character. By default, the separator is the SPACE character.

3.3.3 Brief Header Format

The format and contents of the DTF headers output in brief mode can be customized by specifying the positions and sizes of the various fields in the DTF_BRIEF_FORMAT environment variable. The format of this variable is:

"field1[=width]|field2[=width]|..."

Each field specifier has an optional width and is separated from other field specifiers by a | character. The field specifiers are shown in Table 3-5. Note that specifiers cannot be abbreviated in the environment variable.

Table 3-5: Field Specifiers

Field	Description
SEQUENCE	Trace data sequence number
TIME	Trace data timestamp
NUMBER	Tracepoint number (when using multiple tracepoints)
NAME	Tracepoint instance name
EVENT	Event name
SIZE	Packet size

For instance, the default brief header format can be achieved using the following DTF_BRIEF_FORMAT string. Note that a negative field width indicates that the contents of the field should be aligned to the left.

“time|number|event=-8|name=-8"

3.4 What Does the Output Mean?

DTF uses two forms of output, brief and verbose. Brief mode uses a single line for each traced packet and displays a minimum amount of information about each packet. Verbose mode uses multiple lines for each traced packet and formats all the fields.

3.4.1 Brief Mode

The following is an example of the brief mode output. This output was generated using the following command line:

```
dtf -lw0 -c188 nis100:"routing circuit *"
```

```
12:23:17.20|0|Rx    |L601-8-0  |OSI L2 LAN hello 08-00-2B-A5-F4-40
12:23:17.27|0|Rx    |L601-8-0  |OSI L1 LAN hello 08-00-2B-A0-E4-10
12:23:17.29|0|Rx    |L601-8-0  |PIV L1 routing 1.191
12:23:17.31|0|Rx    |L601-8-0  |PIV LAN Router Hello AA-00-04-00-68-04
12:23:17.34|0|Rx    |L601-8-0  |PIV LAN Router Hello AA-00-04-00-CC-04
12:23:17.39|0|ipRx  |L601-8-0  |IP 16.36.16.203 224.0.0.2 UDP VRC
```

The output shows six packets which passed through the tracepoints in the Routing Circuits on node NIS100. Each trace record is time-stamped (by the router) and lists the filter name applied to the trace record by the router (this name can be used in a filter list to restrict the traced packets). The fields for the sixth packet are described in detail below.

12:23:17.39|	Timestamp applied to the trace record by the router.
0|	This is the tracepoint number. Tracepoints are numbered starting from zero. Since there was only one tracepoint specified, this field will always be zero in this case.
ipRx|	Filter name. This particular packet has been tagged with the "ipRx" filter.
L601-8-0 |	Tracepoint name. This is the name of the tracepoint through which the packet passed.
IP ...	Body of the packet. This particular packet is a UDP packet to the VRC port, sent with a source IP address of 16.36.16.203 to the multicast address 224.0.0.2.

3.4.2 Verbose Mode

The following is an example of the verbose mode output. This output was generated using the command line:

```
dtf -NDlvw0 -f"ospftx" nis100:"routing circuit *"
```

```
-DTF- 0 ospfTx      84 of 84   at  09:53:03.46        L601-8-0    3
     Status: 8063DAE0        Context: 00000002    Function: 8000B685
--------------------------------------------------------------

       routing context: IP
       -------------------

       IP: Common
       ----------
       type of service              0xC0
       total length                 84
       packet identifier            0x002D
       time to live                 1
       Source Address               16.36.16.100
       Destination Address          224.0.0.5

       IP protocol: OSPF
       -----------------
       Version                      V2
       Packet Length                64
       Router ID                    16.36.16.100
       Area ID                      1.1.1.1
       Checksum                     0xF406
       Authentication type          None
       Authentication               0x0000000000000000

       OSPF type: HELLO
       ----------------
       Options                      0x02 E
       Designated router            16.36.16.118
       Backup router                16.36.16.223
       Network mask                 255.255.255.0
       Hello interval               10
       Router priority              1
```

```
Dead interval                40
OSPF neighbors ...           5
                             16.36.16.118
                             16.36.16.182
                             16.36.16.206
                             16.36.16.207
```

The output shows an OSPF Hello packet transmitted on the L601-8-0 tracepoint. The output consists of two parts, the header and the body:

- The header is above the dashed line and contains general information about the traced record.
- The body is below the dashed line and contains the formatted packet.

The fields for the header are described in detail below (reading from top to bottom and left to right).

0	Tracepoint number. Since there was only one tracepoint specified on the command line, this field will always be zero in this case.
ospfTx	Filter name. This particular packet has been tagged with the "ospfTx" filter indicating that it is a transmitted OSPF packet.
84 of 84	This field shows the number of bytes captured versus the original packet size (the first number is the captured size and the second the original size). This example indicates that all of the original 84 bytes were captured.
at 09:53:03.46	Timestamp applied by the router. This example shows that this packet was captured at 9:53 am.
L601-8-0	Tracepoint name.
3	Sequence number. This number is incremented for every packet traced on a particular tracepoint for this session. In this example, this is the third packet traced on this tracepoint.
Status: 8063DAE0	Status field. This value is returned by the router as part of the trace record, and is used by the decode routines to help correctly identify the packet.

Context: 00000002	Context field. This value is returned by the router as part of the trace record, and is used by the decode routines to help correctly identify the packet.
Function: 8000B685	Function field. This value is returned by the router as part of the trace record, and is used by the decode routines to help correctly identify the packet.

The body of the output is separated into sections for each Protocol header in the packet. In this example, the first protocol header is IP; this in turn encapsulates an OSPF protocol header, which is followed by the OSPF HELLO packet.

3.5 Time Formats

DTF uses a variety of formats for displaying the time in the output, depending on the time specification used by the router being traced and whether the user has requested Absolute or Relative times. The timestamp is applied to the trace data by the router. This timestamp may be either an absolute UTC time or a relative count of the number of timer ticks that have occurred since some event. In the latter case DTF cannot display an absolute time, so it simply displays the timestamp value returned from the router. Table 3-6 describes each format

Table 3-6: Timestamp Formats.

Format	Description
hh:mm:ss.mm	UTC time. This is the time of day at the router when the trace record was created. If Relative time was specified on the command line, then this is the time difference (according to the router) since the last displayed record.
dddddd	The timestamp applied by the router, no units are implied.
+dddddd	As above, but Relative time was specified on the command line. This shows the number of timer ticks (according to the router) since the last displayed record.
hh:mm:ssL	Local time (i.e. host time). This is used when there is no timestamp on the traced data. (Pipe mode only.)
hh:mm:ss s	As above, but the time is expressed as a relative time since the last displayed record.

Appendix A

Compatibility and Regular Expressions

A.1 Compatibility with CTF

DTF uses the same protocol as CTF; therefore, any router which supports CTF should also work with DTF. The format of the data file used is the same for DTF and CTF, so DTF can read a CTF$TRACE.DAT file produced by CTF (although the converse is not necessarily true).

A.2 Regular Expression Syntax

A regular expression is zero or more branches, separated by |. It matches anything that matches one of the branches.

A branch is zero or more pieces, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an atom possibly followed by *, +, or ?. An atom followed by * matches a sequence of 0 or more matches of the atom. An atom followed by + matches a sequence of 1 or more matches of the atom. An atom followed by ? matches a match of the atom, or the null string.

An atom is a regular expression in parentheses (matching a match for the regular expression), a range (see below), . (matching any single character), ^ (matching the null string at the beginning of the input string), $ (matching the null string at the end of the input string), a \ followed by a single character (matching that character), or a single character with no other significance (matching that character).

A range is a sequence of characters enclosed in [ ]. It normally matches any single character from the sequence. If the sequence begins with ^, it matches any single character not from the rest of the sequence.

If two characters in the sequence are separated by -, this is short-hand for the full list of ASCII characters between them (e.g. [0-9] matches any decimal digit). To include a literal] in the sequence, make it the first character (following a possible ^). To include a literal - in the sequence, make it the first or last character.

A.2.1 Ambiguity

If a regular expression matches two different parts of the input string, it will match the one which begins earliest. If both begin in the same place but match different lengths, or match the same length in different ways, the outcome is more complex, as follows.

In general, the possibilities in a list of branches are considered in left-to-right order, the possibilities for *, +, and ? are considered longest-first, nested constructs are considered from the outermost in, and concatenated constructs are considered leftmost-first.

The match that will be chosen is the one that uses the earliest possibility in the first choice that has to be made. If there is more than one choice, the next will be made in the same manner (earliest possibility) subject to the decision on the first choice. And so forth.

For example, (ab | a)b*c could match abc in one of two ways. The first choice is between ab and a; since ab is earlier, and does lead to a successful overall match, it is chosen. Since the b is already spoken for, the b* must match its last possibility - the empty string since it must respect the earlier choice.

In the particular case where no | characters are present and there is only one *, +, or ?, the net effect is that the longest possible match will be chosen. So ab*, presented with xabbbby, will match abbbb. Note that if ab* is tried against xabyabbbz, it will match ab just after x, due to the begins-earliest rule. (In effect, the decision on where to start the match is the first choice to be made, hence subsequent choices must respect it even if this leads them to less preferred alternatives.)