NAME

tshark - Dump and analyze network traffic

SYNOPSIS

tshark [**-2**] [**-a** <capture autostop condition>] ... [**-b** <capture ring buffer option>] ... [**-B** <capture buffer size>] [**-c** <capture packet count>] [**-C** <configuration profile>] [**-d** <layer type>==<selector>,<decode-as protocol>] [**-D**] [**-e** <field>] [**-E** <field print option>] [**-f** <capture filter>] [**-F** <file format>] [**-g**] [**-h**] [**-H** <input hosts file>] [**-i** <capture interface>|-] [**-I**] [**-K** <keytab>] [**-l**] [**-L**] [**-n**] [**-N** <name resolving flags>] [**-o** <preference setting>] ... [**-O** <protocols>] [**-p**] [**-P**] [**-q**] [**-Q**] [**-r** <infile>] [**-R** <Read filter>] [**-s** <capture snaplen>] [**-S** <separator>] [**-t** a|ad|adoy|d|dd|e|r|u|ud|udoy] [**-T** fields|pdml|ps|psml|text] [**-u** <seconds type>] [**-v**] [**-V**] [**-w** <outfile>|-] [**-W** <file format option>] [**-x**] [**-X** <eXtension option>] [**-y** <capture link type>] [**-Y** <displaY filter>] [**-z** <statistics>] [**--capture-comment** <comment>] [<capture filter>]

tshark -G [column-formats|currentprefs|decodes|defaultprefs|fields|ftypes|heuristic-decodes|plugins|protocols|values]

DESCRIPTION

TShark is a network protocol analyzer. It lets you capture packet data from a live network, or read packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file. **TShark**'s native capture file format is **pcap** format, which is also the format used by **tcpdump** and various other tools.

Without any options set, **TShark** will work much like **tcpdump**. It will use the pcap library to capture traffic from the first available network interface and displays a summary line on stdout for each received packet.

TShark is able to detect, read and write the same capture files that are supported by **Wireshark**. The input file doesn't need a specific filename extension; the file format and an optional gzip compression will be automatically detected. Near the beginning of the DESCRIPTION section of `wireshark(1)` or https://www.wireshark.org/docs/man-pages/wireshark.html is a detailed description of the way **Wireshark** handles this, which is the same way **Tshark** handles this.

Compressed file support uses (and therefore requires) the zlib library. If the zlib library is not present, **TShark** will compile, but will be unable to read compressed files.

If the **-w** option is not specified, **TShark** writes to the standard output the text of a decoded form of the packets it captures or reads. If the **-w** option is specified, **TShark** writes to the file specified by that option the raw data of the packets, along with the packets' time stamps.

When writing a decoded form of packets, **TShark** writes, by default, a summary line containing the fields specified by the preferences file (which are also the fields displayed in the packet list pane in **Wireshark**), although if it's writing packets as it captures them, rather than writing packets from a saved capture file, it won't show the "frame number" field. If the **-V** option is specified, it writes instead a view of the details of the packet, showing all the fields of all protocols in the packet. If the **-O** option is specified, it will only show the full protocols specified. Use the output of "**tshark -G protocols**" to find the abbreviations of the protocols you can specify.

If you want to write the decoded form of packets to a file, run **TShark** without the **-w** option, and redirect its standard output to the file (do *not* use the **-w** option).

When writing packets to a file, **TShark**, by default, writes the file in **pcap** format, and writes all of the packets it sees to the output file. The **-F** option can be used to specify the format in which to write the file. This list of available file formats is displayed by the **-F** flag without a value. However, you can't specify a file format for a live capture.

Read filters in **TShark**, which allow you to select which packets are to be decoded or written to a file, are very powerful; more fields are filterable in **TShark** than in other protocol analyzers, and the syntax you can use to create your filters is richer. As **TShark** progresses, expect more and more protocol fields to be allowed in read filters.

Packet capturing is performed with the pcap library. The capture filter syntax follows the rules of the pcap library. This syntax is different from the read filter syntax. A read filter can also be specified when capturing, and only packets that pass the read filter will be displayed or saved to the output file; note, however, that capture filters are much more efficient than read filters, and it may be more difficult for **TShark** to keep up with a busy network if a read filter is specified for a live capture.

A capture or read filter can either be specified with the **-f** or **-R** option, respectively, in which case the entire filter expression must be specified as a single argument (which means that if it contains spaces, it must be quoted), or can be specified with command-line arguments after the option arguments, in which case all the arguments after the filter arguments are treated as a filter expression. Capture filters are supported only when doing a live capture; read filters are supported when doing a live capture and when reading a capture file, but require TShark to do more work when filtering, so you might be more likely to lose packets under heavy load if you're using a read filter. If the filter is specified with command-line arguments after the option arguments, it's a capture filter if a capture is being done (i.e., if no **-r** option was specified) and a read filter if a capture file is being read (i.e., if a **-r** option was specified).

The **-G** option is a special mode that simply causes **Tshark** to dump one of several types of internal glossaries and then exit.

OPTIONS

-2

Perform a two-pass analysis. This causes tshark to buffer output until the entire first pass is done, but allows it to fill in fields that require future knowledge, such as 'response in frame #' fields. Also permits reassembly frame dependencies to be calculated correctly.

-a <capture autostop condition>

Specify a criterion that specifies when **TShark** is to stop writing to a capture file. The criterion is of the form *test*:*value*, where *test* is one of:

duration:*value* Stop writing to a capture file after *value* seconds have elapsed.

filesize:*value* Stop writing to a capture file after it reaches a size of *value* kB. If this option is used together with the -b option, **TShark** will stop writing to the current capture file and switch to the next one if filesize is reached. When reading a capture file, **TShark** will stop reading the file after the number of bytes read exceeds this number (the complete packet will be read, so more bytes than this number may be read). Note that the filesize is limited to a maximum value of 2 GiB.

files:*value* Stop writing to capture files after *value* number of files were written.

-b <capture ring buffer option>

Cause **TShark** to run in "multiple files" mode. In "multiple files" mode, **TShark** will write to several capture files. When the first capture file fills up, **TShark** will switch writing to the next file and so on.

The created filenames are based on the filename given with the **-w** option, the number of the file and on the creation date and time, e.g. outfile_00001_20050604120117.pcap, outfile_00002_20050604120523.pcap, ...

With the *files* option it's also possible to form a "ring buffer". This will fill up new files until the number of files specified, at which point **TShark** will discard the data in the first file and start writing to that file and so on. If the *files* option is not set, new files filled up until one of the capture stop conditions match (or until the disk is full).

The criterion is of the form *key*:*value*, where *key* is one of:

duration:*value* switch to the next file after *value* seconds have elapsed, even if the current file is not completely filled up.

filesize:*value* switch to the next file after it reaches a size of *value* kB. Note that the filesize is limited to a maximum value of 2 GiB.

files:*value* begin again with the first file after *value* number of files were written (form a ring buffer). This value must be less than 100000. Caution should be used when using large numbers of files: some filesystems do not handle many files in a single directory well. The **files** criterion requires either **duration** or **filesize** to be specified to control when to go to the next file. It should be noted that each **-b** parameter takes exactly one criterion; to specify two criterion, each must be preceded by the **-b** option.

Example: **-b filesize:1000 -b files:5** results in a ring buffer of five files of size one megabyte each.

-B <capture buffer size>

Set capture buffer size (in MiB, default is 2 MiB). This is used by the capture driver to buffer packet data until that data can be written to disk. If you encounter packet drops while capturing, try to increase this size. Note that, while **Tshark** attempts to set the buffer size to 2 MiB by default, and can be told to set it to a larger value, the system or interface on which you're capturing might silently limit the capture buffer size to a lower value or raise it to a higher value.

This is available on UNIX systems with libpcap 1.0.0 or later and on Windows. It is not available on UNIX systems with earlier versions of libpcap.

This option can occur multiple times. If used before the first occurrence of the **-i** option, it sets the default capture buffer size. If used after an **-i** option, it sets the capture buffer size for the interface specified by the last **-i** option occurring before this option. If the capture buffer size is not set specifically, the default capture buffer size is used instead.

-c <capture packet count>

Set the maximum number of packets to read when capturing live data. If reading a capture file, set the maximum number of packets to read.

-C <configuration profile>

Run with the given configuration profile.

-d <layer type>==<selector>,<decode-as protocol>

Like Wireshark's **Decode As...** feature, this lets you specify how a layer type should be dissected. If the layer type in question (for example, **tcp.port** or **udp.port** for a TCP or UDP port number) has the specified selector value, packets should be dissected as the specified protocol.

Example: **-d tcp.port==8888,http** will decode any traffic running over TCP port 8888 as HTTP.

Example: **-d tcp.port==8888:3,http** will decode any traffic running over TCP ports 8888, 8889 or 8890 as HTTP.

Example: **-d tcp.port==8888-8890,http** will decode any traffic running over TCP ports 8888, 8889 or 8890 as HTTP.

Using an invalid selector or protocol will print out a list of valid selectors and protocol names, respectively.

Example: **-d .** is a quick way to get a list of valid selectors.

Example: **-d ethertype==0x0800.** is a quick way to get a list of protocols that can be selected with an ethertype.

-D

Print a list of the interfaces on which **TShark** can capture, and exit. For each network interface, a number and an interface name, possibly followed by a text description of the interface, is printed. The interface name or the number can be supplied to the **-i** option to specify an interface on which to capture.

This can be useful on systems that don't have a command to list them (e.g., Windows systems, or UNIX systems lacking **ifconfig -a**); the number can be useful on Windows 2000 and later systems, where the interface name is a somewhat complex string.

Note that "can capture" means that **TShark** was able to open that device to do a live capture. Depending on your system you may need to run tshark from an account with special privileges (for example, as root) to be able to capture network traffic. If **TShark -D** is not run from such an account, it will not list any interfaces.

-e <field>

Add a field to the list of fields to display if **-T fields** is selected. This option can be used multiple times on the command line. At least one field must be provided if the **-T fields** option is selected. Column names may be used prefixed with "_ws.col."

Example: **-e frame.number -e ip.addr -e udp -e _ws.col.info**

Giving a protocol rather than a single field will print multiple items of data about the protocol as a single field. Fields are separated by tab characters by default. **-E** controls the format of the printed fields.

-E <field print option>

Set an option controlling the printing of fields when **-T fields** is selected.

Options are:

header=y|n If **y**, print a list of the field names given using **-e** as the first line of the output; the field name will be separated using the same character as the field values. Defaults to **n**.

separator=/t|/s|<character> Set the separator character to use for fields. If **/t** tab will be used (this is the default), if **/s**, a single space will be used. Otherwise any character that can be accepted by the command line as part of the option may be used.

occurrence=f|l|a Select which occurrence to use for fields that have multiple occurrences. If **f** the first occurrence will be used, if **l** the last occurrence will be used and if **a** all occurrences will be used (this is the default).

aggregator=,|/s|<character> Set the aggregator character to use for fields that have multiple occurrences. If **,** a comma will be used (this is the default), if **/s**, a single space will be used. Otherwise any character that can be accepted by the command line as part of the option may be used.

quote=d|s|n Set the quote character to use to surround fields. **d** uses double-quotes, **s** single-quotes, **n** no quotes (the default).

-f <capture filter>

Set the capture filter expression.

This option can occur multiple times. If used before the first occurrence of the **-i** option, it sets the default capture filter expression. If used after an **-i** option, it sets the capture filter expression for the interface specified by the last **-i** option occurring before this option. If the capture filter expression is not set specifically, the default capture filter expression is used if provided.

-F <file format>

Set the file format of the output capture file written using the **-w** option. The output written with the **-w** option is raw packet data, not text, so there is no **-F** option to request text output. The option **-F** without a value will list the available formats.

-g

This option causes the output file(s) to be created with group-read permission (meaning that the output file(s) can be read by other members of the calling user's group).

-G [column-formats|currentprefs|decodes|defaultprefs|fields|ftypes|heuristic-decodes|plugins|protocols|values]

The **-G** option will cause **Tshark** to dump one of several types of glossaries and then exit. If no specific glossary type is specified, then the **fields** report will be generated by default.

The available report types include:

column-formats Dumps the column formats understood by tshark. There is one record per line. The fields are tab-delimited.

```
* Field 1 = format string (e.g. "%rD")
* Field 2 = text description of format string (e.g. "Dest port (resolved)")
```

currentprefs Dumps a copy of the current preferences file to stdout.

decodes Dumps the "layer type"/"decode as" associations to stdout. There is one record per line. The fields are tab-delimited.

```
* Field 1 = layer type, e.g. "tcp.port"
* Field 2 = selector in decimal
* Field 3 = "decode as" name, e.g. "http"
```

defaultprefs Dumps a default preferences file to stdout.

fields Dumps the contents of the registration database to stdout. An independent program can take this output and format it into nice tables or HTML or whatever. There is one record per line. Each record is either a protocol or a header field, differentiated by the first field. The fields are tab-delimited.

```
* Protocols
* ---------
* Field 1 = 'P'
* Field 2 = descriptive protocol name
* Field 3 = protocol abbreviation
*
* Header Fields
* -------------
* Field 1 = 'F'
* Field 2 = descriptive field name
* Field 3 = field abbreviation
* Field 4 = type ( textual representation of the ftenum type )
* Field 5 = parent protocol abbreviation
* Field 6 = base for display (for integer types); "parent bitfield width" for FT_BOOLEAN
* Field 7 = bitmask: format: hex: 0x....
* Field 8 = blurb describing field
```

ftypes Dumps the "ftypes" (fundamental types) understood by tshark. There is one record per line. The fields are tab-delimited.

```
* Field 1 = FTYPE (e.g "FT_IPv6")
* Field 2 = text description of type (e.g. "IPv6 address")
```

heuristic-decodes Dumps the heuristic decodes currently installed. There is one record per line. The fields are tab-delimited.

```
* Field 1 = underlying dissector (e.g. "tcp")
* Field 2 = name of heuristic decoder (e.g. ucp")
* Field 3 = heuristic enabled (e.g. "T" or "F")
```

plugins Dumps the plugins currently installed. There is one record per line. The fields are tab-delimited.

```
* Field 1 = plugin library (e.g. "gryphon.so")
* Field 2 = plugin version (e.g. 0.0.4)
* Field 3 = plugin type (e.g. "dissector" or "tap")
* Field 4 = full path to plugin file
```

protocols Dumps the protocols in the registration database to stdout. An independent program can take this output and format it into nice tables or HTML or whatever. There is one record per line. The fields are tab-delimited.

```
* Field 1 = protocol name
* Field 2 = protocol short name
* Field 3 = protocol filter name
```

values Dumps the value_strings, range_strings or true/false strings for fields that have them. There is one record per line. Fields are tab-delimited. There are three types of records: Value String, Range String and True/False String. The first field, 'V', 'R' or 'T', indicates the type of record.

```
* Value Strings
* -------------
* Field 1 = 'V'
* Field 2 = field abbreviation to which this value string corresponds
* Field 3 = Integer value
* Field 4 = String
*
* Range Strings
* -------------
* Field 1 = 'R'
* Field 2 = field abbreviation to which this range string corresponds
* Field 3 = Integer value: lower bound
* Field 4 = Integer value: upper bound
* Field 5 = String
*
* True/False Strings
* ------------------
* Field 1 = 'T'
* Field 2 = field abbreviation to which this true/false string corresponds
* Field 3 = True String
* Field 4 = False String
```

-h

Print the version and options and exits.

-H <input hosts file>

Read a list of entries from a "hosts" file, which will then be written to a capture file. Implies **-W n**. Can be called multiple times.

The "hosts" file format is documented at http://en.wikipedia.org/wiki/Hosts_(file).

-i <capture interface> | **-**

Set the name of the network interface or pipe to use for live packet capture.

Network interface names should match one of the names listed in "**tshark -D**" (described above); a number, as reported by "**tshark -D**", can also be used. If you're using UNIX, "**netstat -i**" or "**ifconfig -a**" might also work to list interface names, although not all versions of UNIX support the **-a** option to **ifconfig**.

If no interface is specified, **TShark** searches the list of interfaces, choosing the first non-loopback interface if there are any non-loopback interfaces, and choosing the first loopback interface if there are no non-loopback interfaces. If there are no interfaces at all, **TShark** reports an error and doesn't start the capture.

Pipe names should be either the name of a FIFO (named pipe) or ``-" to read data from the standard input. Data read from pipes must be in standard pcap format.

This option can occur multiple times. When capturing from multiple interfaces, the capture file will be saved in pcap-ng format.

Note: the Win32 version of **TShark** doesn't support capturing from pipes!

-I

Put the interface in "monitor mode"; this is supported only on IEEE 802.11 Wi-Fi interfaces, and supported only on some operating systems.

Note that in monitor mode the adapter might disassociate from the network with which it's associated, so that you will not be able to use any wireless networks with that adapter. This could prevent accessing files on a network server, or resolving host names or network addresses, if you are capturing in monitor mode and are not connected to another network with another adapter.

This option can occur multiple times. If used before the first occurrence of the **-i** option, it enables the monitor mode for all interfaces. If used after an **-i** option, it enables the monitor mode for the interface specified by the last **-i** option occurring before this option.

-K <keytab>

Load kerberos crypto keys from the specified keytab file. This option can be used multiple times to load keys from several files.

Example: **-K krb5.keytab**

-l

Flush the standard output after the information for each packet is printed. (This is not, strictly speaking, line-buffered if **-V** was specified; however, it is the same as line-buffered if **-V** wasn't specified, as only one line is printed for each packet, and, as **-l** is normally used when piping a live capture to a program or script, so that output for a packet shows up as soon as the packet is seen and dissected, it should work just as well as true line-buffering. We do this as a workaround for a deficiency in the Microsoft Visual C++ C library.)

This may be useful when piping the output of **TShark** to another program, as it means that the program to which the output is piped will see the dissected data for a packet as soon as **TShark** sees the packet and generates that output, rather than seeing it only when the standard output buffer containing that data fills up.

-L

List the data link types supported by the interface and exit. The reported link types can be used for the **-y** option.

-n

Disable network object name resolution (such as hostname, TCP and UDP port names); the **-N** flag might override this one.

-N <name resolving flags>

Turn on name resolving only for particular types of addresses and port numbers, with name resolving for other types of addresses and port numbers turned off. This flag overrides **-n** if both **-N** and **-n** are present. If both **-N** and **-n** flags are not present, all name resolutions are turned on.

The argument is a string that may contain the letters:

C to enable concurrent (asynchronous) DNS lookups

m to enable MAC address resolution

n to enable network address resolution

N to enable using external resolvers (e.g., DNS) for network address resolution

t to enable transport-layer port number resolution

-o <preference>:<value>

Set a preference value, overriding the default value and any value read from a preference file. The argument to the option is a string of the form *prefname***:***value*, where *prefname* is the name of the preference (which is the same name that would appear in the preference file), and *value* is the value to which it should be set.

-O <protocols>

Similar to the **-V** option, but causes **TShark** to only show a detailed view of the comma-separated list of *protocols* specified, rather than a detailed view of all protocols. Use the output of "**tshark -G protocols**" to find the abbreviations of the protocols you can specify.

-p

Don't put the interface into promiscuous mode. Note that the interface might be in promiscuous mode for some other reason; hence, **-p** cannot be used to ensure that the only traffic that is captured is traffic sent to or from the machine on which **TShark** is running, broadcast traffic, and multicast traffic to addresses received by that machine.

This option can occur multiple times. If used before the first occurrence of the **-i** option, no interface will be put into the promiscuous mode. If used after an **-i** option, the interface specified by the last **-i** option occurring before this option will not be put into the promiscuous mode.

-P

Decode and display the packet summary, even if writing raw packet data using the **-w** option.

-q

When capturing packets, don't display the continuous count of packets captured that is normally shown when saving a capture to a file; instead, just display, at the end of the capture, a count of packets captured. On systems that support the SIGINFO signal, such as various BSDs, you can cause the current count to be displayed by typing your "status" character (typically control-T, although it might be set to "disabled" by default on at least some BSDs, so you'd have to explicitly set it to use it).

When reading a capture file, or when capturing and not saving to a file, don't print packet information; this is useful if you're using a **-z** option to calculate statistics and don't want the packet information printed, just the statistics.

-Q

When capturing packets, only display true errors. This outputs less than the **-q** option, so the interface name and total packet count and the end of a capture are not sent to stderr.

-r <infile>

Read packet data from *infile*, can be any supported capture file format (including gzipped files). It is possible to use named pipes or stdin (**-**) here but only with certain (not compressed) capture file formats (in particular: those that can be read without seeking backwards).

-R <Read filter>

Cause the specified filter (which uses the syntax of read/display filters, rather than that of capture filters) to be applied during the first pass of analysis. Packets not matching the filter are not considered for future passes. Only makes sense with multiple passes, see -2. For regular filtering on single-pass dissect see -Y instead.

Note that forward-looking fields such as 'response in frame #' cannot be used with this filter, since they will not have been calculate when this filter is applied.

-s <capture snaplen>

Set the default snapshot length to use when capturing live data. No more than *snaplen* bytes of each network packet will be read into memory, or saved to disk. A value of 0 specifies a snapshot length of 65535, so that the full packet is captured; this is the default.

This option can occur multiple times. If used before the first occurrence of the **-i** option, it sets the default snapshot length. If used after an **-i** option, it sets the snapshot length for the interface specified by the last **-i** option occurring before this option. If the snapshot length is not set specifically, the default snapshot length is used if provided.

-S <separator>

Set the line separator to be printed between packets.

-t a|ad|adoy|d|dd|e|r|u|ud|udoy

Set the format of the packet timestamp printed in summary lines. The format can be one of:

a absolute: The absolute time, as local time in your time zone, is the actual time the packet was captured, with no date displayed

ad absolute with date: The absolute date, displayed as YYYY-MM-DD, and time, as local time in your time zone, is the actual time and date the packet was captured

adoy absolute with date using day of year: The absolute date, displayed as YYYY/DOY, and time, as local time in your time zone, is the actual time and date the packet was captured

d delta: The delta time is the time since the previous packet was captured

dd delta_displayed: The delta_displayed time is the time since the previous displayed packet was captured

e epoch: The time in seconds since epoch (Jan 1, 1970 00:00:00)

r relative: The relative time is the time elapsed between the first packet and the current packet

u UTC: The absolute time, as UTC, is the actual time the packet was captured, with no date displayed

ud UTC with date: The absolute date, displayed as YYYY-MM-DD, and time, as UTC, is the actual time and date the packet was captured

udoy UTC with date using day of year: The absolute date, displayed as YYYY/DOY, and time, as UTC, is the actual time and date the packet was captured

The default format is relative.

-T fields|pdml|ps|psml|text

Set the format of the output when viewing decoded packet data. The options are one of:

fields The values of fields specified with the **-e** option, in a form specified by the **-E** option. For example,

```
-T fields -E separator=, -E quote=d
```

would generate comma-separated values (CSV) output suitable for importing into your favorite spreadsheet program.

pdml Packet Details Markup Language, an XML-based format for the details of a decoded packet. This information is equivalent to the packet details printed with the **-V** flag.

ps PostScript for a human-readable one-line summary of each of the packets, or a multi-line view of the details of each of the packets, depending on whether the **-V** flag was specified.

psml Packet Summary Markup Language, an XML-based format for the summary information of a decoded packet. This information is equivalent to the information shown in the one-line summary printed by default.

text Text of a human-readable one-line summary of each of the packets, or a multi-line view of the details of each of the packets, depending on whether the **-V** flag was specified. This is the default.

-u <seconds type>

Specifies the seconds type. Valid choices are:

s for seconds

hms for hours, minutes and seconds

-v

Print the version and exit.

-V

Cause **TShark** to print a view of the packet details.

-w <outfile> | -

Write raw packet data to *outfile* or to the standard output if *outfile* is '-'.

NOTE: -w provides raw packet data, not text. If you want text output you need to redirect stdout (e.g. using '>'), don't use the **-w** option for this.

-W <file format option>

Save extra information in the file if the format supports it. For example,

```
-F pcapng -W n
```

will save host name resolution records along with captured packets.

Future versions of Wireshark may automatically change the capture format to **pcapng** as needed.

The argument is a string that may contain the following letter:

n write network address resolution information (pcapng only)

-x

Cause **TShark** to print a hex and ASCII dump of the packet data after printing the summary and/or details, if either are also being displayed.

-X <eXtension options>

Specify an option to be passed to a **TShark** module. The eXtension option is in the form *extension_key***:***value*, where *extension_key* can be:

lua_script:*lua_script_filename* tells **TShark** to load the given script in addition to the default Lua scripts.

lua_script*num*:*argument* tells **TShark** to pass the given argument to the lua script identified by 'num', which is the number indexed order of the 'lua_script' command. For example, if only one script was loaded with '-X lua_script:my.lua', then '-X lua_script1:foo' will pass the string 'foo' to the 'my.lua' script. If two scripts were loaded, such as '-X lua_script:my.lua'

and '-X lua_script:other.lua' in that order, then a '-X lua_script2:bar' would pass the string 'bar' to the second lua script, namely 'other.lua'.

read_format:*file_format* tells **TShark** to use the given file format to read in the file (the file given in the **-r** command option). Providing no *file_format* argument, or an invalid one, will produce a file of available file formats to use.

-y <capture link type>

Set the data link type to use while capturing packets. The values reported by **-L** are the values that can be used.

This option can occur multiple times. If used before the first occurrence of the **-i** option, it sets the default capture link type. If used after an **-i** option, it sets the capture link type for the interface specified by the last **-i** option occurring before this option. If the capture link type is not set specifically, the default capture link type is used if provided.

-Y <displaY filter>

Cause the specified filter (which uses the syntax of read/display filters, rather than that of capture filters) to be applied before printing a decoded form of packets or writing packets to a file. Packets matching the filter are printed or written to file; packets that the matching packets depend upon (e.g., fragments), are not printed but are written to file; packets not matching the filter nor depended upon are discarded rather than being printed or written.

Use this instead of -R for filtering using single-pass analysis. If doing two-pass analysis (see -2) then only packets matching the read filter (if there is one) will be checked against this filter.

-z <statistics>

Get **TShark** to collect various types of statistics and display the result after finishing reading the capture file. Use the **-q** flag if you're reading a capture file and only want the statistics printed, not any per-packet information.

Note that the **-z proto** option is different - it doesn't cause statistics to be gathered and printed when the capture is complete, it modifies the regular packet summary output to include the values of fields specified with the option. Therefore you must not use the **-q** option, as that option would suppress the printing of the regular packet summary output, and must also not use the **-V** option, as that would cause packet detail information rather than packet summary information to be printed.

Currently implemented statistics are:

-z help

Display all possible values for **-z**.

-z afp,srt[,*filter*]

-z camel,srt

-z compare,*start*,*stop*,*ttl[0|1]*,*order[0|1]*,*variance*[,*filter*]

If the optional *filter* is specified, only those packets that match the filter will be used in the calculations.

-z conv,*type*[,*filter*]

Create a table that lists all conversations that could be seen in the capture. *type* specifies the conversation endpoint types for which we want to generate the statistics; currently the supported ones are:

```
"eth"   Ethernet addresses
"fc"    Fibre Channel addresses
"fddi"  FDDI addresses
"ip"    IPv4 addresses
"ipv6"  IPv6 addresses
"ipx"   IPX addresses
"tcp"   TCP/IP socket pairs  Both IPv4 and IPv6 are supported
"tr"    Token Ring addresses
"udp"   UDP/IP socket pairs  Both IPv4 and IPv6 are supported
```

If the optional *filter* is specified, only those packets that match the filter will be used in the calculations.

The table is presented with one line for each conversation and displays the number of packets/bytes in each direction as well as the total number of packets/bytes. The table is sorted according to the total number of frames.

-z dcerpc,srt,*uuid*,*major.minor*[,*filter*]

Collect call/reply SRT (Service Response Time) data for DCERPC interface *uuid*, version *major.minor*. Data collected is the number of calls for each procedure, MinSRT, MaxSRT and AvgSRT.

Example: **-z dcerpc,srt,12345778-1234-abcd-ef00-0123456789ac,1.0** will collect data for the CIFS SAMR Interface.

This option can be used multiple times on the command line.

If the optional *filter* is provided, the stats will only be calculated on those calls that match that filter.

Example: **-z dcerpc,srt,12345778-1234-abcd-ef00-0123456789ac,1.0,ip.addr==1.2.3.4** will collect SAMR SRT statistics for a specific host.

-z diameter,avp[,*cmd.code*,*field*,*field*,...]

This option enables extraction of most important diameter fields from large capture files. Exactly one text line for each diameter message with matched **diameter.cmd.code** will be printed.

Empty diameter command code or '*' can be specified to mach any **diameter.cmd.code**

Example: **-z diameter,avp** extract default field set from diameter messages.

Example: **-z diameter,avp,280** extract default field set from diameter DWR messages.

Example: **-z diameter,avp,272** extract default field set from diameter CC messages.

Extract most important fields from diameter CC messages:

tshark -r file.cap.gz -q -z diameter,avp,272,CC-Request-Type,CC-Request-Number,Session-Id,Subscription-Id-Data,Rating-Group,Result-Code

Following fields will be printed out for each diameter message:

```
"frame"        Frame number.
"time"         Unix time of the frame arrival.
"src"          Source address.
"srcport"      Source port.
"dst"          Destination address.
"dstport"      Destination port.
"proto"        Constant string 'diameter', which can be used for post processing of tshark output.  E.g. grep/sed/awk.
"msgnr"        seq. number of diameter message within the frame.  E.g. '2' for the third diameter message in the same frame.
```

```
    "is_request"   '0' if message is a request, '1' if message is an answer.
    "cmd"          diameter.cmd_code, E.g. '272' for credit control messages.
    "req_frame"    Number of frame where matched request was found or '0'.
    "ans_frame"    Number of frame where matched answer was found or '0'.
    "resp_time"    response time in seconds, '0' in case if matched Request/Answer is not found in trace.  E.g. in the begin or end of captur
```

-z diameter,avp option is much faster than **-V -T text** or **-T pdml** options.

-z diameter,avp option is more powerful than **-T field** and **-z proto,colinfo** options.

Multiple diameter messages in one frame are supported.

Several fields with same name within one diameter message are supported, e.g. *diameter.Subscription-Id-Data* or *diameter.Rating-Group*.

Note: **tshark -q** option is recommended to suppress default **tshark** output.

-z expert[,*error*|,*warn*|,*note*|,*chat*][,*filter*]

Collects information about all expert info, and will display them in order, grouped by severity.

Example: **-z expert,sip** will show expert items of all severity for frames that match the sip protocol.

This option can be used multiple times on the command line.

If the optional *filter* is provided, the stats will only be calculated on those calls that match that filter.

Example: **-z "expert,note,tcp"** will only collect expert items for frames that include the tcp protocol, with a severity of note or higher.

-z follow,*prot*,*mode*,*filter*[,*range*]

Displays the contents of a TCP or UDP stream between two nodes. The data sent by the second node is prefixed with a tab to differentiate it from the data sent by the first node.

prot specifies the transport protocol. It can be one of: **tcp** TCP **udp** UDP **ssl** SSL

mode specifies the output mode. It can be one of: **ascii** ASCII output with dots for non-printable characters **hex** Hexadecimal and ASCII data with offsets **raw** Hexadecimal data

Since the output in **ascii** mode may contain newlines, the length of each section of output plus a newline precedes each section of output.

filter specifies the stream to be displayed. UDP streams are selected with IP address plus port pairs. TCP streams are selected with either the stream index or IP address plus port pairs. For example: **ip-addr0**:**port0**,**ip-addr1**:**port1 tcp-stream-index**

range optionally specifies which "chunks" of the stream should be displayed.

Example: **-z "follow,tcp,hex,1"** will display the contents of the first TCP stream in "hex" format.

```
    ===================================================================
    Follow: tcp,hex
    Filter: tcp.stream eq 1
    Node 0: 200.57.7.197:32891
    Node 1: 200.57.7.198:2906
    00000000  00 00 00 22 00 00 00 07  00 0a 85 02 07 e9 00 02  ..."....  ........
    00000010  07 e9 06 0f 00 0d 00 04  00 00 00 01 00 03 00 06  ........  ........
    00000020  1f 00 06 04 00 00                                  ......
        00000000  00 01 00 00                                        ....
        00000026  00 02 00 00
```

Example: **-z "follow,tcp,ascii,200.57.7.197:32891,200.57.7.198:2906"** will display the contents of a TCP stream between 200.57.7.197 port 32891 and 200.57.7.98 port 2906.

```
    ===================================================================
    Follow: tcp,ascii
    Filter: (omitted for readability)
    Node 0: 200.57.7.197:32891
    Node 1: 200.57.7.198:2906
    38
    ...".....
    ................
        4
        ....
```

-z h225,counter[,*filter*]

Count ITU-T H.225 messages and their reasons. In the first column you get a list of H.225 messages and H.225 message reasons, which occur in the current capture file. The number of occurrences of each message or reason is displayed in the second column.

Example: **-z h225,counter**.

If the optional *filter* is provided, the stats will only be calculated on those calls that match that filter. Example: use **-z "h225,counter,ip.addr==1.2.3.4"** to only collect stats for H.225 packets exchanged by the host at IP address 1.2.3.4 .

This option can be used multiple times on the command line.

-z h225,srt[,*filter*]

Collect requests/response SRT (Service Response Time) data for ITU-T H.225 RAS. Data collected is number of calls of each ITU-T H.225 RAS Message Type, Minimum SRT, Maximum SRT, Average SRT, Minimum in Packet, and Maximum in Packet. You will also get the number of Open Requests (Unresponded Requests), Discarded Responses (Responses without matching request) and Duplicate Messages.

Example: **-z h225,srt**

This option can be used multiple times on the command line.

If the optional *filter* is provided, the stats will only be calculated on those calls that match that filter.

Example: **-z "h225,srt,ip.addr==1.2.3.4"** will only collect stats for ITU-T H.225 RAS packets exchanged by the host at IP address 1.2.3.4 .

-z hosts[,ipv4][,ipv6]

Dump any collected IPv4 and/or IPv6 addresses in "hosts" format. Both IPv4 and IPv6 addresses are dumped by default.

Addresses are collected from a number of sources, including standard "hosts" files and captured traffic.

-z http,stat,

Calculate the HTTP statistics distribution. Displayed values are the HTTP status codes and the HTTP request methods.

-z http,tree

Calculate the HTTP packet distribution. Displayed values are the HTTP request modes and the HTTP status codes.

-z http_req,tree

Calculate the HTTP requests by server. Displayed values are the server name and the URI path.

-z http_srv,tree

Calculate the HTTP requests and responses by server. For the HTTP requests, displayed values are the server IP address and server hostname. For the HTTP responses, displayed values are the server IP address and status.

-z icmp,srt[,*filter*]

Compute total ICMP echo requests, replies, loss, and percent loss, as well as minimum, maximum, mean, median and sample standard deviation SRT statistics typical of what ping provides.

Example: **-z icmp,srt,ip.src==1.2.3.4** will collect ICMP SRT statistics for ICMP echo request packets originating from a specific host.

This option can be used multiple times on the command line.

-z icmpv6,srt[,*filter*]

Compute total ICMPv6 echo requests, replies, loss, and percent loss, as well as minimum, maximum, mean, median and sample standard deviation SRT statistics typical of what ping provides.

Example: **-z icmpv6,srt,ipv6.src==fe80::1** will collect ICMPv6 SRT statistics for ICMPv6 echo request packets originating from a specific host.

This option can be used multiple times on the command line.

-z io,phs[,*filter*]

Create Protocol Hierarchy Statistics listing both number of packets and bytes. If no *filter* is specified the statistics will be calculated for all packets. If a *filter* is specified statistics will only be calculated for those packets that match the filter.

This option can be used multiple times on the command line.

-z io,stat,*interval*[,*filter*][,*filter*][,*filter*]...

Collect packet/bytes statistics for the capture in intervals of *interval* seconds. *Interval* can be specified either as a whole or fractional second and can be specified with microsecond (us) resolution. If *interval* is 0, the statistics will be calculated over all packets.

If no *filter* is specified the statistics will be calculated for all packets. If one or more *filters* are specified statistics will be calculated for all filters and presented with one column of statistics for each filter.

This option can be used multiple times on the command line.

Example: **-z io,stat,1,ip.addr==1.2.3.4** will generate 1 second statistics for all traffic to/from host 1.2.3.4.

Example: **-z "io,stat,0.001,smb&&ip.addr==1.2.3.4"** will generate 1ms statistics for all SMB packets to/from host 1.2.3.4.

The examples above all use the standard syntax for generating statistics which only calculates the number of packets and bytes in each interval.

io,stat can also do much more statistics and calculate `COUNT()`, `SUM()`, `MIN()`, `MAX()`, `AVG()` and `LOAD()` using a slightly different filter syntax:

-z io,stat,*interval*,"[COUNT|SUM|MIN|MAX|AVG|LOAD](*field*)*filter*"

NOTE: One important thing to note here is that the filter is not optional and that the field that the calculation is based on MUST be part of the filter string or the calculation will fail.

So: **-z io,stat,0.010,AVG(smb.time)** does not work. Use **-z io,stat,0.010,AVG(smb.time)smb.time** instead. Also be aware that a field can exist multiple times inside the same packet and will then be counted multiple times in those packets.

NOTE: A second important thing to note is that the system setting for decimal separator must be set to "."! If it is set to "," the statistics will not be displayed per filter.

COUNT(*field*)*filter* - Calculates the number of times that the field *name* (not its value) appears per interval in the filtered packet list. "*field*" can be any display filter name.

Example: **-z io,stat,0.010,"COUNT(smb.sid)smb.sid"**

This will count the total number of SIDs seen in each 10ms interval.

SUM(*field*)*filter* - Unlike COUNT, the *values* of the specified field are summed per time interval. "*field*" can only be a named integer, float, double or relative time field.

Example: **-z io,stat,0.010,"SUM(frame.len)frame.len"**

Reports the total number of bytes that were transmitted bidirectionally in all the packets within a 10 millisecond interval.

MIN/MAX/AVG(*field*)*filter* - The minimum, maximum, or average field value in each interval is calculated. The specified field must be a named integer, float, double or relative time field. For relative time fields, the output is presented in seconds with six decimal digits of precision rounded to the nearest microsecond.

In the following example, the time of the first Read_AndX call, the last Read_AndX response values are displayed and the minimum, maximum, and average Read response times (SRTs) are calculated. NOTE: If the DOS command shell line continuation character, "^" is used, each line cannot end in a comma so it is placed at the beginning of each continuation line:

```
tshark -o tcp.desegment_tcp_streams:FALSE -n -q -r smb_reads.cap -z io,stat,0,
"MIN(frame.time_relative)frame.time_relative and smb.cmd==0x2e and smb.flags.response==0",
"MAX(frame.time_relative)frame.time_relative and smb.cmd==0x2e and smb.flags.response==1",
"MIN(smb.time)smb.time and smb.cmd==0x2e",
"MAX(smb.time)smb.time and smb.cmd==0x2e",
"AVG(smb.time)smb.time and smb.cmd==0x2e"
===================================================================================================
IO Statistics
Column #0: MIN(frame.time_relative)frame.time_relative and smb.cmd==0x2e and smb.flags.response==0
Column #1: MAX(frame.time_relative)frame.time_relative and smb.cmd==0x2e and smb.flags.response==1
Column #2: MIN(smb.time)smb.time and smb.cmd==0x2e
Column #3: MAX(smb.time)smb.time and smb.cmd==0x2e
Column #4: AVG(smb.time)smb.time and smb.cmd==0x2e
                |    Column #0    |    Column #1    |    Column #2    |    Column #3    |    Column #4    |
Time            |       MIN       |       MAX       |       MIN       |       MAX       |       AVG       |
000.000-                0.000000        7.704054        0.000072        0.005539        0.000295
===================================================================================================
```

The following command displays the average SMB Read response PDU size, the total number of read PDU bytes, the average SMB Write request PDU size, and the total number of bytes transferred in SMB Write PDUs:

```
tshark -n -q -r smb_reads_writes.cap -z io,stat,0,
"AVG(smb.file.rw.length)smb.file.rw.length and smb.cmd==0x2e and smb.response_to",
"SUM(smb.file.rw.length)smb.file.rw.length and smb.cmd==0x2e and smb.response_to",
"AVG(smb.file.rw.length)smb.file.rw.length and smb.cmd==0x2f and not smb.response_to",
"SUM(smb.file.rw.length)smb.file.rw.length and smb.cmd==0x2f and not smb.response_to"
================================================================================
IO Statistics
Column #0: AVG(smb.file.rw.length)smb.file.rw.length and smb.cmd==0x2e and smb.response_to
Column #1: SUM(smb.file.rw.length)smb.file.rw.length and smb.cmd==0x2e and smb.response_to
Column #2: AVG(smb.file.rw.length)smb.file.rw.length and smb.cmd==0x2f and not smb.response_to
Column #3: SUM(smb.file.rw.length)smb.file.rw.length and smb.cmd==0x2f and not smb.response_to
                |    Column #0    |    Column #1    |    Column #2    |    Column #3    |
Time            |       AVG       |       SUM       |       AVG       |       SUM       |
000.000-                  30018         28067522              72              3240
================================================================================
```

LOAD(*field*)*filter* - The LOAD/Queue-Depth in each interval is calculated. The specified field must be a relative time field that represents a response time. For example smb.time. For each interval the Queue-Depth for the specified protocol is calculated.

The following command displays the average SMB LOAD. A value of 1.0 represents one I/O in flight.

```
tshark -n -q -r smb_reads_writes.cap
-z "io,stat,0.001,LOAD(smb.time)smb.time"
================================================================================
IO Statistics
Interval:   0.001000 secs
Column #0: LOAD(smb.time)smb.time
                          |    Column #0    |
Time                      |       LOAD      |
0000.000000-0000.001000   |          1.000000
0000.001000-0000.002000   |          0.741000
0000.002000-0000.003000   |          0.000000
0000.003000-0000.004000   |          1.000000
```

FRAMES | BYTES[()*filter*] - Displays the total number of frames or bytes. The filter field is optional but if included it must be prepended with "()".

The following command displays five columns: the total number of frames and bytes (transferred bidirectionally) using a single comma, the same two stats using the FRAMES and BYTES subcommands, the total number of frames containing at least one SMB Read response, and the total number of bytes transmitted to the client (unidirectionally) at IP address 10.1.0.64.

```
tshark -o tcp.desegment_tcp_streams:FALSE -n -q -r smb_reads.cap -z io,stat,0,,FRAMES,BYTES,
"FRAMES()smb.cmd==0x2e and smb.response_to","BYTES()ip.dst==10.1.0.64"
================================================================================================================
IO Statistics
Column #0:
Column #1: FRAMES
Column #2: BYTES
Column #3: FRAMES()smb.cmd==0x2e and smb.response_to
Column #4: BYTES()ip.dst==10.1.0.64
                |                Column #0                |    Column #1    |    Column #2    |    Column #3    |    Column #4    |
Time            |     Frames      |      Bytes            |     FRAMES      |      BYTES      |     FRAMES      |      BYTES      |
000.000-                  33576         29721685              33576         29721685              870         29004801
================================================================================================================
```

-z mac-lte,stat[,*filter*]

This option will activate a counter for LTE MAC messages. You will get information about the maximum number of UEs/TTI, common messages and various counters for each UE that appears in the log.

Example: **-z mac-lte,stat**.

This option can be used multiple times on the command line.

If the optional *filter* is provided, the stats will only be calculated for those frames that match that filter. Example: **-z "mac-lte,stat,mac-lte.rnti**3000"> will only collect stats for UEs with an assigned RNTI whose value is more than 3000.

-z megaco,rtd[,*filter*]

Collect requests/response RTD (Response Time Delay) data for MEGACO. (This is similar to **-z smb,srt**). Data collected is the number of calls for each known MEGACO Type, MinRTD, MaxRTD and AvgRTD. Additionally you get the number of duplicate requests/responses, unresponded requests, responses, which don't match with any request. Example: **-z megaco,rtd**.

If the optional *filter* is provided, the stats will only be calculated on those calls that match that filter. Example: **-z "megaco,rtd,ip.addr==1.2.3.4"** will only collect stats for MEGACO packets exchanged by the host at IP address 1.2.3.4 .

This option can be used multiple times on the command line.

-z mgcp,rtd[,*filter*]

Collect requests/response RTD (Response Time Delay) data for MGCP. (This is similar to **-z smb,srt**). Data collected is the number of calls for each known MGCP Type, MinRTD, MaxRTD and AvgRTD. Additionally you get the number of duplicate requests/responses, unresponded requests, responses, which don't match with any request. Example: **-z mgcp,rtd**.

This option can be used multiple times on the command line.

If the optional *filter* is provided, the stats will only be calculated on those calls that match that filter. Example: **-z "mgcp,rtd,ip.addr==1.2.3.4"** will only collect stats for MGCP packets exchanged by the host at IP address 1.2.3.4 .

-z proto,colinfo,*filter*,*field*

Append all *field* values for the packet to the Info column of the one-line summary output. This feature can be used to append arbitrary fields to the Info column in addition to the normal content of that column. *field* is the display-filter name of a field which value should be placed in the Info column. *filter* is a filter string that controls for which packets the field value will be presented in the info column. *field* will only be presented in the Info column for the packets which match *filter*.

NOTE: In order for **TShark** to be able to extract the *field* value from the packet, *field* MUST be part of the *filter* string. If not, **TShark** will not be able to extract its value.

For a simple example to add the "nfs.fh.hash" field to the Info column for all packets containing the "nfs.fh.hash" field, use

-z proto,colinfo,nfs.fh.hash,nfs.fh.hash

To put "nfs.fh.hash" in the Info column but only for packets coming from host 1.2.3.4 use:

-z "proto,colinfo,nfs.fh.hash && ip.src==1.2.3.4,nfs.fh.hash"

This option can be used multiple times on the command line.

-z rlc-lte,stat[,*filter*]

This option will activate a counter for LTE RLC messages. You will get information about common messages and various counters for each UE that appears in the log.

Example: **-z rlc-lte,stat**.

This option can be used multiple times on the command line.

If the optional *filter* is provided, the stats will only be calculated for those frames that match that filter. Example: **-z "rlc-lte,stat,rlc-lte.ueid**3000"> will only collect stats for UEs with a UEId of more than 3000.

-z rpc,programs

Collect call/reply SRT data for all known ONC-RPC programs/versions. Data collected is number of calls for each protocol/version, MinSRT, MaxSRT and AvgSRT. This option can only be used once on the command line.

-z rpc,srt,*program*,*version*[,*filter*]

Collect call/reply SRT (Service Response Time) data for *program*/*version*. Data collected is the number of calls for each procedure, MinSRT, MaxSRT, AvgSRT, and the total time taken for each procedure.

Example: **-z rpc,srt,100003,3** will collect data for NFS v3.

This option can be used multiple times on the command line.

If the optional *filter* is provided, the stats will only be calculated on those calls that match that filter.

Example: **-z rpc,srt,100003,3,nfs.fh.hash==0x12345678** will collect NFS v3 SRT statistics for a specific file.

-z rtp,streams

Collect statistics for all RTP streams and calculate max. delta, max. and mean jitter and packet loss percentages.

-z scsi,srt,*cmdset*[,*filter*]

Collect call/reply SRT (Service Response Time) data for SCSI commandset *cmdset*.

Commandsets are 0:SBC 1:SSC 5:MMC

Data collected is the number of calls for each procedure, MinSRT, MaxSRT and AvgSRT.

Example: **-z scsi,srt,0** will collect data for SCSI BLOCK COMMANDS (SBC).

This option can be used multiple times on the command line.

If the optional *filter* is provided, the stats will only be calculated on those calls that match that filter.

Example: **-z scsi,srt,0,ip.addr==1.2.3.4** will collect SCSI SBC SRT statistics for a specific iscsi/ifcp/fcip host.

-z sip,stat[,*filter*]

This option will activate a counter for SIP messages. You will get the number of occurrences of each SIP Method and of each SIP Status-Code. Additionally you also get the number of resent SIP Messages (only for SIP over UDP).

Example: **-z sip,stat**.

This option can be used multiple times on the command line.

If the optional *filter* is provided, the stats will only be calculated on those calls that match that filter. Example: **-z "sip,stat,ip.addr==1.2.3.4"** will only collect stats for SIP packets exchanged by the host at IP address 1.2.3.4 .

-z smb,sids

When this feature is used **TShark** will print a report with all the discovered SID and account name mappings. Only those SIDs where the account name is known will be presented in the table.

For this feature to work you will need to either to enable "Edit/Preferences/Protocols/SMB/Snoop SID to name mappings" in the preferences or you can override the preferences by specifying **-o "smb.sid_name_snooping:TRUE"** on the **TShark** command line.

The current method used by **TShark** to find the SID->name mapping is relatively restricted with a hope of future expansion.

-z smb,srt[,*filter*]

Collect call/reply SRT (Service Response Time) data for SMB. Data collected is number of calls for each SMB command, MinSRT, MaxSRT and AvgSRT.

Example: **-z smb,srt**

The data will be presented as separate tables for all normal SMB commands, all Transaction2 commands and all NT Transaction commands. Only those commands that are seen in the capture will have its stats displayed. Only the first command in a xAndX command chain will be used in the calculation. So for common SessionSetupAndX + TreeConnectAndX chains, only the SessionSetupAndX call will be used in the statistics. This is a flaw that might be fixed in the future.

This option can be used multiple times on the command line.

If the optional *filter* is provided, the stats will only be calculated on those calls that match that filter.

Example: **-z "smb,srt,ip.addr==1.2.3.4"** will only collect stats for SMB packets exchanged by the host at IP address 1.2.3.4 .

--capture-comment <comment>

Add a capture comment to the output file.

This option is only available if a new output file in pcapng format is created. Only one capture comment may be set per output file.

CAPTURE FILTER SYNTAX

See the manual page of pcap-filter(7) or, if that doesn't exist, `tcpdump(8)`, or, if that doesn't exist, https://wiki.wireshark.org/CaptureFilters.

READ FILTER SYNTAX

For a complete table of protocol and protocol fields that are filterable in **TShark** see the wireshark-filter(4) manual page.

FILES

These files contains various **Wireshark** configuration values.

Preferences

The *preferences* files contain global (system-wide) and personal preference settings. If the system-wide preference file exists, it is read first, overriding the default settings. If the personal preferences file exists, it is read next, overriding any previous values. Note: If the command line option **-o** is used (possibly more than once), it will in turn override values from the preferences files.

The preferences settings are in the form *prefname***:***value*, one per line, where *prefname* is the name of the preference and *value* is the value to which it should be set; white space is allowed between **:** and *value*. A preference setting can be continued on subsequent lines by indenting the continuation lines with white space. A **#** character starts a comment that runs to the end of the line:

```
# Capture in promiscuous mode?
# TRUE or FALSE (case-insensitive).
capture.prom_mode: TRUE
```

The global preferences file is looked for in the *wireshark* directory under the *share* subdirectory of the main installation directory (for example, */usr/local/share/wireshark/preferences*) on UNIX-compatible systems, and in the main installation directory (for example, *C:\Program Files\Wireshark\preferences*) on Windows systems.

The personal preferences file is looked for in *$HOME/.wireshark/preferences* on UNIX-compatible systems and *%APPDATA%\Wireshark\preferences* (or, if %APPDATA% isn't defined, *%USERPROFILE%\Application Data\Wireshark\preferences*) on Windows systems.

Disabled (Enabled) Protocols

The *disabled_protos* files contain system-wide and personal lists of protocols that have been disabled, so that their dissectors are never called. The files contain protocol names, one per line, where the protocol name is the same name that would be used in a display filter for the protocol:

```
http
tcp     # a comment
```

The global *disabled_protos* file uses the same directory as the global preferences file.

The personal *disabled_protos* file uses the same directory as the personal preferences file.

Name Resolution (hosts)

If the personal *hosts* file exists, it is used to resolve IPv4 and IPv6 addresses before any other attempts are made to resolve them. The file has the standard *hosts* file syntax; each line contains one IP address and name, separated by whitespace. The same directory as for the personal preferences file is used.

Capture filter name resolution is handled by libpcap on UNIX-compatible systems and WinPcap on Windows. As such the Wireshark personal *hosts* file will not be consulted for capture filter name resolution.

Name Resolution (ethers)

The *ethers* files are consulted to correlate 6-byte hardware addresses to names. First the personal *ethers* file is tried and if an address is not found there the global *ethers* file is tried next.

Each line contains one hardware address and name, separated by whitespace. The digits of the hardware address are separated by colons (:), dashes (-) or periods (.). The same separator character must be used consistently in an address. The following three lines are valid lines of an *ethers* file:

```
ff:ff:ff:ff:ff:ff          Broadcast
c0-00-ff-ff-ff-ff          TR_broadcast
00.00.00.00.00.00          Zero_broadcast
```

The global *ethers* file is looked for in the */etc* directory on UNIX-compatible systems, and in the main installation directory (for example, *C:\Program Files\Wireshark*) on Windows systems.

The personal *ethers* file is looked for in the same directory as the personal preferences file.

Capture filter name resolution is handled by libpcap on UNIX-compatible systems and WinPcap on Windows. As such the Wireshark personal *ethers* file will not be consulted for capture filter name resolution.

Name Resolution (manuf)

The *manuf* file is used to match the 3-byte vendor portion of a 6-byte hardware address with the manufacturer's name; it can also contain well-known MAC addresses and address ranges specified with a netmask. The format of the file is the same as the *ethers* files, except that entries of the form:

```
00:00:0C      Cisco
```

can be provided, with the 3-byte OUI and the name for a vendor, and entries such as:

```
00-00-0C-07-AC/40     All-HSRP-routers
```

can be specified, with a MAC address and a mask indicating how many bits of the address must match. The above entry, for example, has 40 significant bits, or 5 bytes, and would match addresses from 00-00-0C-07-AC-00 through 00-00-0C-07-AC-FF. The mask need not be a multiple of 8.

The *manuf* file is looked for in the same directory as the global preferences file.

Name Resolution (ipxnets)

The *ipxnets* files are used to correlate 4-byte IPX network numbers to names. First the global *ipxnets* file is tried and if that address is not found there the personal one is tried next.

The format is the same as the *ethers* file, except that each address is four bytes instead of six. Additionally, the address can be represented as a single hexadecimal number, as is more common in the IPX world, rather than four hex octets. For example, these four lines are valid lines of an *ipxnets* file:

```
C0.A8.2C.00             HR
c0-a8-1c-00             CEO
00:00:BE:EF             IT_Server1
110f                    FileServer3
```

The global *ipxnets* file is looked for in the */etc* directory on UNIX-compatible systems, and in the main installation directory (for example, *C:\Program Files\Wireshark*) on Windows systems.

The personal *ipxnets* file is looked for in the same directory as the personal preferences file.

ENVIRONMENT VARIABLES

WIRESHARK_APPDATA

On Windows, Wireshark normally stores all application data in %APPDATA% or %USERPROFILE%. You can override the default location by exporting this environment variable to specify an alternate location.

WIRESHARK_DEBUG_EP_NO_CHUNKS

Normally per-packet memory is allocated in large "chunks." This behavior doesn't work well with debugging tools such as Valgrind or ElectricFence. Export this environment variable to force individual allocations. Note: disabling chunks also disables canaries (see below).

WIRESHARK_DEBUG_SE_NO_CHUNKS

Normally per-file memory is allocated in large "chunks." This behavior doesn't work well with debugging tools such as Valgrind or ElectricFence. Export this environment variable to force individual allocations. Note: disabling chunks also disables canaries (see below).

WIRESHARK_DEBUG_EP_NO_CANARY

Normally per-packet memory allocations are separated by "canaries" which allow detection of memory overruns. This comes at the expense of some extra memory usage. Exporting this environment variable disables these canaries.

WIRESHARK_DEBUG_SE_USE_CANARY

Exporting this environment variable causes per-file memory allocations to be protected with "canaries" which allow for detection of memory overruns. This comes at the expense of significant extra memory usage.

WIRESHARK_DEBUG_SCRUB_MEMORY

If this environment variable is set, the contents of per-packet and per-file memory is initialized to 0xBADDCAFE when the memory is allocated and is reset to 0xDEADBEEF when the memory is freed. This functionality is useful mainly to developers looking for bugs in the way memory is handled.

WIRESHARK_DEBUG_WMEM_OVERRIDE

Setting this environment variable forces the wmem framework to use the specified allocator backend for *all* allocations, regardless of which backend is normally specified by the code. This is mainly useful to developers when testing or debugging. See *README.wmem* in the source distribution for details.

WIRESHARK_RUN_FROM_BUILD_DIRECTORY

This environment variable causes the plugins and other data files to be loaded from the build directory (where the program was compiled) rather than from the standard locations. It has no effect when the program in question is running with root (or setuid) permissions on *NIX.

WIRESHARK_DATA_DIR

This environment variable causes the various data files to be loaded from a directory other than the standard locations. It has no effect when the program in question is running with root (or setuid) permissions on *NIX.

WIRESHARK_PYTHON_DIR

This environment variable points to an alternate location for Python. It has no effect when the program in question is running with root (or setuid) permissions on *NIX.

ERF_RECORDS_TO_CHECK

This environment variable controls the number of ERF records checked when deciding if a file really is in the ERF format. Setting this environment variable a number higher than the default (20) would make false positives less likely.

IPFIX_RECORDS_TO_CHECK

This environment variable controls the number of IPFIX records checked when deciding if a file really is in the IPFIX format. Setting this environment variable a number higher than the default (20) would make false positives less likely.

WIRESHARK_ABORT_ON_DISSECTOR_BUG

If this environment variable is set, **TShark** will call `abort(3)` when a dissector bug is encountered. `abort(3)` will cause the program to exit abnormally; if you are running **TShark** in a debugger, it should halt in the debugger and allow inspection of the process, and, if you are not running it in a debugger, it will, on some OSes, assuming your environment is configured correctly, generate a core dump file. This can be useful to developers attempting to troubleshoot a problem with a protocol dissector.

WIRESHARK_ABORT_ON_TOO_MANY_ITEMS

If this environment variable is set, **TShark** will call `abort(3)` if a dissector tries to add too many items to a tree (generally this is an indication of the dissector not breaking out of a loop soon enough). `abort(3)` will cause the program to exit abnormally; if you are running **TShark** in a debugger, it should halt in the debugger and allow inspection of the process, and, if you are not running it in a debugger, it will, on some OSes, assuming your environment is configured correctly, generate a core dump file. This can be useful to developers attempting to troubleshoot a problem with a protocol dissector.

WIRESHARK_EP_VERIFY_POINTERS

This environment variable, if present, causes certain uses of pointers to be audited to ensure they do not point to memory that is deallocated after each packet has been fully dissected. This can be useful to developers writing or auditing code.

WIRESHARK_SE_VERIFY_POINTERS

This environment variable, if present, causes certain uses of pointers to be audited to ensure they do not point to memory that is deallocated after when a capture file is closed. This can be useful to developers writing or auditing code.

WIRESHARK_ABORT_ON_OUT_OF_MEMORY

This environment variable, if present, causes `abort(3)` to be called if certain out-of-memory conditions (which normally result in an exception and an explanatory error message) are experienced. This can be useful to developers debugging out-of-memory conditions.

SEE ALSO

wireshark-filter(4), `wireshark(1)`, `editcap(1)`, `pcap(3)`, `dumpcap(1)`, text2pcap(1), `mergecap(1)`, pcap-filter(7) or `tcpdump(8)`

NOTES

TShark is part of the **Wireshark** distribution. The latest version of **Wireshark** can be found at https://www.wireshark.org.

HTML versions of the Wireshark project man pages are available at: https://www.wireshark.org/docs/man-pages.

AUTHORS

TShark uses the same packet dissection code that **Wireshark** does, as well as using many other modules from **Wireshark**; see the list of authors in the **Wireshark** man page for a list of authors of that code.