NAME

tcpdump - dump traffic on a network

SYNOPSIS

tcpdump [**-AbdDefhHIJKlLnNOpqStuUvxX#**] [**-B** *buffer_size*]
[**-c** *count*] [**--count**] [**-C** *file_size*]
[**-E** *spi@ipaddr algo:secret,...*]
[**-F** *file*] [**-G** *rotate_seconds*] [**-i** *interface*]
[**--immediate-mode**] [**-j** *tstamp_type*] [**-m** *module*]
[**-M** *secret*] [**--number**] [**--print**] [**-Q** *in*|*out*|*inout*]
[**-r** *file*] [**-s** *snaplen*] [**-T** *type*] [**--version**]
[**-V** *file*] [**-w** *file*] [**-W** *filecount*] [**-y** *datalinktype*]
[**-z** *postrotate-command*] [**-Z** *user*]
[**--time-stamp-precision=***tstamp_precision*]
[**--micro**] [**--nano**]
[*expression*]

DESCRIPTION

Tcpdump prints out a description of the contents of packets on a network interface that match the Boolean *expression* (see **pcap-filter**(7) for the *expression* syntax); the description is preceded by a time stamp, printed, by default, as hours, minutes, seconds, and fractions of a second since midnight. It can also be run with the **-w** flag, which causes it to save the packet data to a file for later analysis, and/or with the **-r** flag, which causes it to read from a saved packet file rather than to read packets from a network interface. It can also be run with the **-V** flag, which causes it to read a list of saved packet files. In all cases, only packets that match *expression* will be processed by *tcpdump*.

Tcpdump will, if not run with the **-c** flag, continue capturing packets until it is interrupted by a SIGINT signal (generated, for example, by typing your interrupt character, typically control-C) or a SIGTERM signal (typically generated with the **kill**(1) command); if run with the **-c** flag, it will capture packets until it is interrupted by a SIGINT or SIGTERM signal or the specified number of packets have been processed.

When *tcpdump* finishes capturing packets, it will report counts of:

packets ‘‘captured’’ (this is the number of packets that *tcpdump* has received and processed);

packets ‘‘received by filter’’ (the meaning of this depends on the OS on which you’re running *tcpdump*, and possibly on the way the OS was configured - if a filter was specified on the command line, on some OSes it counts packets regardless of whether they were matched by the filter expression and, even if they were matched by the filter expression, regardless of whether

tcpdump has read and processed them yet, on other OSes it counts only packets that were matched by the filter expression regardless of whether *tcpdump* has read and processed them yet, and on other OSes it counts only packets that were matched by the filter expression and were processed by *tcpdump*);

packets ‘‘dropped by kernel’’ (this is the number of packets that were dropped, due to a lack of buffer space, by the packet capture mechanism in the OS on which *tcpdump* is running, if the OS reports that information to applications; if not, it will be reported as 0).

On platforms that support the SIGINFO signal, such as most BSDs (including macOS) and Digital/Tru64 UNIX, it will report those counts when it receives a SIGINFO signal (generated, for example, by typing your ‘‘status’’ character, typically control-T, although on some platforms, such as macOS, the ‘‘status’’ character is not set by default, so you must set it with **stty**(1) in order to use it) and will continue capturing packets. On platforms that do not support the SIGINFO signal, the same can be achieved by using the SIGUSR1 signal.

Using the SIGUSR2 signal along with the **-w** flag will forcibly flush the packet buffer into the output file.

Reading packets from a network interface may require that you have special privileges; see the **pcap**(3PCAP) man page for details. Reading a saved packet file doesn’t require special privileges.

OPTIONS

-A Print each packet (minus its link level header) in ASCII. Handy for capturing web pages.

-b Print the AS number in BGP packets in ASDOT notation rather than ASPLAIN notation.

-B *buffer_size*
--buffer-size=*buffer_size*
Set the operating system capture buffer size to *buffer_size*, in units of KiB (1024 bytes).

-c *count*
Exit after receiving *count* packets.

--count
Print only on stdout the packet count when reading capture file(s) instead of parsing/printing the packets. If a filter is specified on the command line, *tcpdump* counts only packets that were matched by the filter expression.

-C *file_size*

Before writing a raw packet to a savefile, check whether the file is currently larger than *file_size* and, if so, close the current savefile and open a new one. Savefiles after the first savefile will have the name specified with the **-w** flag, with a number after it, starting at 1 and continuing upward. The units of *file_size* are millions of bytes (1,000,000 bytes, not 1,048,576 bytes).

-d Dump the compiled packet-matching code in a human readable form to standard output and stop.

Please mind that although code compilation is always DLT-specific, typically it is impossible (and unnecessary) to specify which DLT to use for the dump because *tcpdump* uses either the DLT of the input pcap file specified with **-r**, or the default DLT of the network interface specified with **-i**, or the particular DLT of the network interface specified with **-y** and **-i** respectively. In these cases the dump shows the same exact code that would filter the input file or the network interface without **-d**.

However, when neither **-r** nor **-i** is specified, specifying **-d** prevents *tcpdump* from guessing a suitable network interface (see **-i**). In this case the DLT defaults to EN10MB and can be set to another valid value manually with **-y**.

-dd Dump packet-matching code as a **C** program fragment.

-ddd
Dump packet-matching code as decimal numbers (preceded with a count).

-D
--list-interfaces
Print the list of the network interfaces available on the system and on which *tcpdump* can capture packets. For each network interface, a number and an interface name, possibly followed by a text description of the interface, are printed. The interface name or the number can be supplied to the **-i** flag 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.

The **-D** flag will not be supported if *tcpdump* was built with an older version of *libpcap* that lacks the **pcap_findalldevs**(3PCAP) function.

-e Print the link-level header on each dump line. This can be used, for example, to print MAC layer addresses for protocols such as Ethernet and IEEE 802.11.

-E Use *spi@ipaddr algo:secret* for decrypting IPsec ESP packets that are addressed to *addr* and contain Security Parameter Index value *spi*. This combination may be repeated with comma or newline separation.

Note that setting the secret for IPv4 ESP packets is supported at this time.

Algorithms may be **des-cbc**, **3des-cbc**, **blowfish-cbc**, **rc3-cbc**, **cast128-cbc**, or **none**. The default is **des-cbc**. The ability to decrypt packets is only present if *tcpdump* was compiled with cryptography enabled.

secret is the ASCII text for ESP secret key. If preceded by 0x, then a hex value will be read.

The option assumes RFC 2406 ESP, not RFC 1827 ESP. The option is only for debugging purposes, and the use of this option with a true ‘secret’ key is discouraged. By presenting IPsec secret key onto command line you make it visible to others, via *ps*(1) and other occasions.

In addition to the above syntax, the syntax *file name* may be used to have tcpdump read the provided file in. The file is opened upon receiving the first ESP packet, so any special permissions that tcpdump may have been given should already have been given up.

-f Print ‘foreign’ IPv4 addresses numerically rather than symbolically (this option is intended to get around serious brain damage in Sun’s NIS server -- usually it hangs forever translating non-local internet numbers).

The test for ‘foreign’ IPv4 addresses is done using the IPv4 address and netmask of the interface on that capture is being done. If that address or netmask are not available, either because the interface on that capture is being done has no address or netmask or because it is the "any" pseudo-interface, which is available in Linux and in recent versions of macOS and Solaris, and which can capture on more than one interface, this option will not work correctly.

-F *file*

Use *file* as input for the filter expression. An additional expression given on the command line is ignored.

-G *rotate_seconds*

If specified, rotates the dump file specified with the **-w** option every *rotate_seconds* seconds. Savefiles will have the name specified by **-w** which should include a time format as defined by **strftime**(3). If no time format is specified, each new file will overwrite the previous. Whenever a generated filename is not unique, tcpdump will overwrite the pre-existing data; providing a time specification that is coarser than the capture period is therefore not advised.

If used in conjunction with the **-C** option, filenames will take the form of ‘*file*<count>’.

-h
--help
Print the tcpdump and libpcap version strings, print a usage message, and exit.

--version
Print the tcpdump and libpcap version strings and exit.

-H Attempt to detect 802.11s draft mesh headers.

-i *interface*
--interface=*interface*
Listen, report the list of link-layer types, report the list of time stamp types, or report the results of compiling a filter expression on *interface*. If unspecified and if the **-d** flag is not given, *tcpdump* searches the system interface list for the lowest numbered, configured up interface (excluding loopback), which may turn out to be, for example, ‘‘eth0’’.

On Linux systems with 2.2 or later kernels and on recent versions of macOS and Solaris, an *interface* argument of ‘‘any’’ can be used to capture packets from all interfaces. Note that captures on the ‘‘any’’ pseudo-interface will not be done in promiscuous mode.

If the **-D** flag is supported, an interface number as printed by that flag can be used as the *interface* argument, if no interface on the system has that number as a name.

-I
--monitor-mode
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 flag will affect the output of the **-L** flag. If **-I** isn’t specified, only those link-layer types available when not in monitor mode will be shown; if **-I** is specified, only those link-layer types available when in monitor mode will be shown.

--immediate-mode

Capture in "immediate mode". In this mode, packets are delivered to tcpdump as soon as they arrive, rather than being buffered for efficiency. This is the default when printing packets rather than saving packets to a ‘‘savefile’’ if the packets are being printed to a terminal rather than to a file or pipe.

-j *tstamp_type*
--time-stamp-type=*tstamp_type*
Set the time stamp type for the capture to *tstamp_type*. The names to use for the time stamp types are given in **pcap-tstamp**(7); not all the types listed there will necessarily be valid for any given interface.

-J
--list-time-stamp-types
List the supported time stamp types for the interface and exit. If the time stamp type cannot be set for the interface, no time stamp types are listed.

--time-stamp-precision=*tstamp_precision*
When capturing, set the time stamp precision for the capture to *tstamp_precision*. Note that availability of high precision time stamps (nanoseconds) and their actual accuracy is platform and hardware dependent. Also note that when writing captures made with nanosecond accuracy to a savefile, the time stamps are written with nanosecond resolution, and the file is written with a different magic number, to indicate that the time stamps are in seconds and nanoseconds; not all programs that read pcap savefiles will be able to read those captures.

When reading a savefile, convert time stamps to the precision specified by *timestamp_precision*, and display them with that resolution. If the precision specified is less than the precision of time stamps in the file, the conversion will lose precision.

The supported values for *timestamp_precision* are **micro** for microsecond resolution and **nano** for nanosecond resolution. The default is microsecond resolution.

--micro
--nano
Shorthands for **--time-stamp-precision=micro** or **--time-stamp-precision=nano**, adjusting the time stamp precision accordingly. When reading packets from a savefile, using **--micro** truncates time stamps if the savefile was created with nanosecond precision. In contrast, a savefile created with microsecond precision will have trailing zeroes added to the time stamp when **--nano** is used.

-K
--dont-verify-checksums

Don’t attempt to verify IP, TCP, or UDP checksums. This is useful for interfaces that perform some or all of those checksum calculation in hardware; otherwise, all outgoing TCP checksums will be flagged as bad.

-l Make stdout line buffered. Useful if you want to see the data while capturing it. E.g.,

tcpdump -l | tee dat

or

tcpdump -l > dat & tail -f dat

Note that on Windows,‘‘line buffered’’ means ‘‘unbuffered’’, so that WinDump will write each character individually if **-l** is specified.

-U is similar to **-l** in its behavior, but it will cause output to be ‘‘packet-buffered’’, so that the output is written to stdout at the end of each packet rather than at the end of each line; this is buffered on all platforms, including Windows.

-L
--list-data-link-types

List the known data link types for the interface, in the specified mode, and exit. The list of known data link types may be dependent on the specified mode; for example, on some platforms, a Wi-Fi interface might support one set of data link types when not in monitor mode (for example, it might support only fake Ethernet headers, or might support 802.11 headers but not support 802.11 headers with radio information) and another set of data link types when in monitor mode (for example, it might support 802.11 headers, or 802.11 headers with radio information, only in monitor mode).

-m *module*

Load SMI MIB module definitions from file *module*. This option can be used several times to load several MIB modules into *tcpdump*.

-M *secret*

Use *secret* as a shared secret for validating the digests found in TCP segments with the TCP-MD5 option (RFC 2385), if present.

-n Don’t convert addresses (i.e., host addresses, port numbers, etc.) to names.

-N Don’t print domain name qualification of host names. E.g., if you give this flag then *tcpdump* will print ‘‘nic’’ instead of ‘‘nic.ddn.mil’’.

-#
--number
Print an optional packet number at the beginning of the line.

-O
--no-optimize
Do not run the packet-matching code optimizer. This is useful only if you suspect a bug in the optimizer.

-p
--no-promiscuous-mode
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 as an abbreviation for 'ether host {local-hw-addr} or ether broadcast'.

--print
Print parsed packet output, even if the raw packets are being saved to a file with the **-w** flag.

-Q *direction*
--direction=*direction*
Choose send/receive direction *direction* for which packets should be captured. Possible values are 'in', 'out' and 'inout'. Not available on all platforms.

-q Quick (quiet?) output. Print less protocol information so output lines are shorter.

-r *file*
Read packets from *file* (which was created with the **-w** option or by other tools that write pcap or pcapng files). Standard input is used if *file* is ''-''.

-S
--absolute-tcp-sequence-numbers
Print absolute, rather than relative, TCP sequence numbers.

-s *snaplen*
--snapshot-length=*snaplen*
Snarf *snaplen* bytes of data from each packet rather than the default of 262144 bytes. Packets truncated because of a limited snapshot are indicated in the output with ''[|*proto*]'', where *proto* is the name of the protocol level at which the truncation has occurred.

Note that taking larger snapshots both increases the amount of time it takes to process packets and,

effectively, decreases the amount of packet buffering. This may cause packets to be lost. Note also that taking smaller snapshots will discard data from protocols above the transport layer, which loses information that may be important. NFS and AFS requests and replies, for example, are very large, and much of the detail won't be available if a too-short snapshot length is selected.

If you need to reduce the snapshot size below the default, you should limit *snaplen* to the smallest number that will capture the protocol information you're interested in. Setting *snaplen* to 0 sets it to the default of 262144, for backwards compatibility with recent older versions of *tcpdump*.

-T *type*

Force packets selected by "*expression*" to be interpreted the specified *type*. Currently known types are **aodv** (Ad-hoc On-demand Distance Vector protocol), **carp** (Common Address Redundancy Protocol), **cnfp** (Cisco NetFlow protocol), **domain** (Domain Name System), **lmp** (Link Management Protocol), **pgm** (Pgmatic General Multicast), **pgm_zmtp1** (ZMTP/1.0 inside PGM/EPGM), **ptp** (Precision Time Protocol), **radius** (RADIUS), **resp** (REdis Serialization Protocol), **rpc** (Remote Procedure Call), **rtcp** (Real-Time Applications control protocol), **rtp** (Real-Time Applications protocol), **snmp** (Simple Network Management Protocol), **someip** (SOME/IP), **tftp** (Trivial File Transfer Protocol), **vat** (Visual Audio Tool), **vxlan** (Virtual eXtensible Local Area Network), **wb** (distributed White Board) and **zmtp1** (ZeroMQ Message Transport Protocol 1.0).

Note that the **pgm** type above affects UDP interpretation only, the native PGM is always recognised as IP protocol 113 regardless. UDP-encapsulated PGM is often called "EPGM" or "PGM/UDP".

Note that the **pgm_zmtp1** type above affects interpretation of both native PGM and UDP at once. During the native PGM decoding the application data of an ODATA/RDATA packet would be decoded as a ZeroMQ datagram with ZMTP/1.0 frames. During the UDP decoding in addition to that any UDP packet would be treated as an encapsulated PGM packet.

-t *Don't* print a timestamp on each dump line.

-tt Print the timestamp, as seconds since January 1, 1970, 00:00:00, UTC, and fractions of a second since that time, on each dump line.

-ttt Print a delta (microsecond or nanosecond resolution depending on the **--time-stamp-precision** option) between current and previous line on each dump line. The default is microsecond resolution.

-tttt Print a timestamp, as hours, minutes, seconds, and fractions of a second since midnight, preceded by the date, on each dump line.

-ttttt

Print a delta (microsecond or nanosecond resolution depending on the **--time-stamp-precision** option) between current and first line on each dump line. The default is microsecond resolution.

-u Print undecoded NFS handles.

-U
--packet-buffered

If the **-w** option is not specified, or if it is specified but the **--print** flag is also specified, make the printed packet output ‘‘packet-buffered’’; i.e., as the description of the contents of each packet is printed, it will be written to the standard output, rather than, when not writing to a terminal, being written only when the output buffer fills.

If the **-w** option is specified, make the saved raw packet output ‘‘packet-buffered’’; i.e., as each packet is saved, it will be written to the output file, rather than being written only when the output buffer fills.

The **-U** flag will not be supported if *tcpdump* was built with an older version of *libpcap* that lacks the **pcap_dump_flush**(3PCAP) function.

-v When parsing and printing, produce (slightly more) verbose output. For example, the time to live, identification, total length and options in an IP packet are printed. Also enables additional packet integrity checks such as verifying the IP and ICMP header checksum.

When writing to a file with the **-w** option and at the same time not reading from a file with the **-r** option, report to stderr, once per second, the number of packets captured. In Solaris, FreeBSD and possibly other operating systems this periodic update currently can cause loss of captured packets on their way from the kernel to tcpdump.

-vv Even more verbose output. For example, additional fields are printed from NFS reply packets, and SMB packets are fully decoded.

-vvv

Even more verbose output. For example, telnet **SB** ... **SE** options are printed in full. With **-X** Telnet options are printed in hex as well.

-V *file*

Read a list of filenames from *file*. Standard input is used if *file* is ‘‘-’’.

-w *file*

Write the raw packets to *file* rather than parsing and printing them out. They can later be printed with the -r option. Standard output is used if *file* is ‘‘-’’.

This output will be buffered if written to a file or pipe, so a program reading from the file or pipe may not see packets for an arbitrary amount of time after they are received. Use the **-U** flag to cause packets to be written as soon as they are received.

The MIME type *application/vnd.tcpdump.pcap* has been registered with IANA for *pcap* files. The filename extension *.pcap* appears to be the most commonly used along with *.cap* and *.dmp*. *Tcpdump* itself doesn’t check the extension when reading capture files and doesn’t add an extension when writing them (it uses magic numbers in the file header instead). However, many operating systems and applications will use the extension if it is present and adding one (e.g. .pcap) is recommended.

See **pcap-savefile**(5) for a description of the file format.

-W *filecount*
Used in conjunction with the **-C** option, this will limit the number of files created to the specified number, and begin overwriting files from the beginning, thus creating a ’rotating’ buffer. In addition, it will name the files with enough leading 0s to support the maximum number of files, allowing them to sort correctly.

Used in conjunction with the **-G** option, this will limit the number of rotated dump files that get created, exiting with status 0 when reaching the limit.

If used in conjunction with both **-C** and **-G,** the **-W** option will currently be ignored, and will only affect the file name.

-x When parsing and printing, in addition to printing the headers of each packet, print the data of each packet (minus its link level header) in hex. The smaller of the entire packet or *snaplen* bytes will be printed. Note that this is the entire link-layer packet, so for link layers that pad (e.g. Ethernet), the padding bytes will also be printed when the higher layer packet is shorter than the required padding. In the current implementation this flag may have the same effect as **-xx** if the packet is truncated.

-xx When parsing and printing, in addition to printing the headers of each packet, print the data of each packet, *including* its link level header, in hex.

-X When parsing and printing, in addition to printing the headers of each packet, print the data of each packet (minus its link level header) in hex and ASCII. This is very handy for analysing new

protocols. In the current implementation this flag may have the same effect as **-XX** if the packet is truncated.

-XX

When parsing and printing, in addition to printing the headers of each packet, print the data of each packet, *including* its link level header, in hex and ASCII.

-y *datalinktype*
--linktype=*datalinktype*

Set the data link type to use while capturing packets (see **-L**) or just compiling and dumping packet-matching code (see **-d**) to *datalinktype*.

-z *postrotate-command*

Used in conjunction with the **-C** or **-G** options, this will make *tcpdump* run " *postrotate-command file* " where *file* is the savefile being closed after each rotation. For example, specifying **-z gzip** or **-z bzip2** will compress each savefile using gzip or bzip2.

Note that tcpdump will run the command in parallel to the capture, using the lowest priority so that this doesn't disturb the capture process.

And in case you would like to use a command that itself takes flags or different arguments, you can always write a shell script that will take the savefile name as the only argument, make the flags & arguments arrangements and execute the command that you want.

-Z *user*
--relinquish-privileges=*user*

If *tcpdump* is running as root, after opening the capture device or input savefile, but before opening any savefiles for output, change the user ID to *user* and the group ID to the primary group of *user*.

This behavior can also be enabled by default at compile time.

expression

selects which packets will be dumped. If no *expression* is given, all packets on the net will be dumped. Otherwise, only packets for which *expression* is 'true' will be dumped.

For the *expression* syntax, see **pcap-filter**(7).

The *expression* argument can be passed to *tcpdump* as either a single Shell argument, or as multiple Shell arguments, whichever is more convenient. Generally, if the expression contains Shell metacharacters, such as backslashes used to escape protocol names, it is easier to pass it as a

single, quoted argument rather than to escape the Shell metacharacters. Multiple arguments are concatenated with spaces before being parsed.

EXAMPLES

To print all packets arriving at or departing from *sundown*:

tcpdump host sundown

To print traffic between *helios* and either *hot* or *ace*:

tcpdump host helios and \(hot or ace \)

To print all IP packets between *ace* and any host except *helios*:

tcpdump ip host ace and not helios

To print all traffic between local hosts and hosts at Berkeley:

tcpdump net ucb-ether

To print all ftp traffic through internet gateway *snup*: (note that the expression is quoted to prevent the shell from (mis-)interpreting the parentheses):

tcpdump 'gateway snup and (port ftp or ftp-data)'

To print traffic neither sourced from nor destined for local hosts (if you gateway to one other net, this stuff should never make it onto your local net).

tcpdump ip and not net *localnet*

To print the start and end packets (the SYN and FIN packets) of each TCP conversation that involves a non-local host.

tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net *localnet***'**

To print the TCP packets with flags RST and ACK both set. (i.e. select only the RST and ACK flags in the flags field, and if the result is "RST and ACK both set", match)

tcpdump 'tcp[tcpflags] & (tcp-rst|tcp-ack) == (tcp-rst|tcp-ack)'

To print all IPv4 HTTP packets to and from port 80, i.e. print only packets that contain data, not, for example, SYN and FIN packets and ACK-only packets. (IPv6 is left as an exercise for the reader.)

tcpdump 'tcp port 80 and (((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)'

To print IP packets longer than 576 bytes sent through gateway *snup*:

tcpdump 'gateway snup and ip[2:2] > 576'

To print IP broadcast or multicast packets that were *not* sent via Ethernet broadcast or multicast:

tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'

To print all ICMP packets that are not echo requests/replies (i.e., not ping packets):

tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply'

OUTPUT FORMAT

The output of *tcpdump* is protocol dependent. The following gives a brief description and examples of most of the formats.

Timestamps

By default, all output lines are preceded by a timestamp. The timestamp is the current clock time in the form

hh:mm:ss.frac

and is as accurate as the kernel's clock. The timestamp reflects the time the kernel applied a time stamp to the packet. No attempt is made to account for the time lag between when the network interface finished receiving the packet from the network and when the kernel applied a time stamp to the packet; that time lag could include a delay between the time when the network interface finished receiving a packet from the network and the time when an interrupt was delivered to the kernel to get it to read the packet and a delay between the time when the kernel serviced the 'new packet' interrupt and the time when it applied a time stamp to the packet.

Link Level Headers

If the '-e' option is given, the link level header is printed out. On Ethernets, the source and destination addresses, protocol, and packet length are printed.

On FDDI networks, the '-e' option causes *tcpdump* to print the 'frame control' field, the source and destination addresses, and the packet length. (The 'frame control' field governs the interpretation of the rest of the packet. Normal packets (such as those containing IP datagrams) are 'async' packets, with a priority value between 0 and 7; for example, '**async4**'. Such packets are assumed to contain an 802.2 Logical Link Control (LLC) packet; the LLC header is printed if it is *not* an ISO datagram or a so-called SNAP packet.

On Token Ring networks, the '-e' option causes *tcpdump* to print the 'access control' and 'frame control' fields, the source and destination addresses, and the packet length. As on FDDI networks, packets are assumed to contain an LLC packet. Regardless of whether the '-e' option is specified or not, the source routing information is printed for source-routed packets.

On 802.11 networks, the '-e' option causes *tcpdump* to print the 'frame control' fields, all of the

addresses in the 802.11 header, and the packet length. As on FDDI networks, packets are assumed to contain an LLC packet.

(N.B.: The following description assumes familiarity with the SLIP compression algorithm described in RFC 1144.)

On SLIP links, a direction indicator (‘‘I’’ for inbound, ‘‘O’’ for outbound), packet type, and compression information are printed out. The packet type is printed first. The three types are *ip*, *utcp*, and *ctcp*. No further link information is printed for *ip* packets. For TCP packets, the connection identifier is printed following the type. If the packet is compressed, its encoded header is printed out. The special cases are printed out as ***S+***n* and ***SA+***n*, where *n* is the amount by which the sequence number (or sequence number and ack) has changed. If it is not a special case, zero or more changes are printed. A change is indicated by U (urgent pointer), W (window), A (ack), S (sequence number), and I (packet ID), followed by a delta (+n or -n), or a new value (=n). Finally, the amount of data in the packet and compressed header length are printed.

For example, the following line shows an outbound compressed TCP packet, with an implicit connection identifier; the ack has changed by 6, the sequence number by 49, and the packet ID by 6; there are 3 bytes of data and 6 bytes of compressed header:

O ctcp * A+6 S+49 I+6 3 (6)

ARP/RARP Packets

ARP/RARP output shows the type of request and its arguments. The format is intended to be self explanatory. Here is a short sample taken from the start of an ‘rlogin’ from host *rtsg* to host *csam*:

```
arp who-has csam tell rtsg
arp reply csam is-at CSAM
```

The first line says that rtsg sent an ARP packet asking for the Ethernet address of internet host csam. Csam replies with its Ethernet address (in this example, Ethernet addresses are in caps and internet addresses in lower case).

This would look less redundant if we had done *tcpdump -n*:

```
arp who-has 128.3.254.6 tell 128.3.254.68
arp reply 128.3.254.6 is-at 02:07:01:00:01:c4
```

If we had done *tcpdump -e*, the fact that the first packet is broadcast and the second is point-to-point would be visible:

```
RTSG Broadcast 0806  64: arp who-has csam tell rtsg
CSAM RTSG 0806  64: arp reply csam is-at CSAM
```

For the first packet this says the Ethernet source address is RTSG, the destination is the Ethernet

broadcast address, the type field contained hex 0806 (type ETHER_ARP) and the total length was 64 bytes.

IPv4 Packets

If the link-layer header is not being printed, for IPv4 packets, **IP** is printed after the time stamp.

If the **-v** flag is specified, information from the IPv4 header is shown in parentheses after the **IP** or the link-layer header. The general format of this information is:

tos *tos*, ttl *ttl*, id *id*, offset *offset*, flags [*flags*], proto *proto*, length *length*, options (*options*)

tos is the type of service field; if the ECN bits are non-zero, those are reported as **ECT(1)**, **ECT(0)**, or **CE**. *ttl* is the time-to-live; it is not reported if it is zero. *id* is the IP identification field. *offset* is the fragment offset field; it is printed whether this is part of a fragmented datagram or not. *flags* are the MF and DF flags; **+** is reported if MF is set, and **DF** is reported if F is set. If neither are set, **.** is reported. *proto* is the protocol ID field. *length* is the total length field. *options* are the IP options, if any.

Next, for TCP and UDP packets, the source and destination IP addresses and TCP or UDP ports, with a dot between each IP address and its corresponding port, will be printed, with a > separating the source and destination. For other protocols, the addresses will be printed, with a > separating the source and destination. Higher level protocol information, if any, will be printed after that.

For fragmented IP datagrams, the first fragment contains the higher level protocol header; fragments after the first contain no higher level protocol header. Fragmentation information will be printed only with the **-v** flag, in the IP header information, as described above.

TCP Packets

(N.B.:The following description assumes familiarity with the TCP protocol described in RFC 793. If you are not familiar with the protocol, this description will not be of much use to you.)

The general format of a TCP protocol line is:

src > *dst*: Flags [*tcpflags*], seq *data-seqno*, ack *ackno*, win *window*, urg *urgent*, options [*opts*], length *len*

Src and *dst* are the source and destination IP addresses and ports. *Tcpflags* are some combination of S (SYN), F (FIN), P (PSH), R (RST), U (URG), W (CWR), E (ECE) or ‘.’ (ACK), or ‘none’ if no flags are set. *Data-seqno* describes the portion of sequence space covered by the data in this packet (see example below). *Ackno* is sequence number of the next data expected the other direction on this connection. *Window* is the number of bytes of receive buffer space available the other direction on this connection. *Urg* indicates there is ‘urgent’ data in the packet. *Opts* are TCP options (e.g., mss 1024). *Len* is the length of payload data.

Iptype, *Src*, *dst*, and *flags* are always present. The other fields depend on the contents of the packet's TCP protocol header and are output only if appropriate.

Here is the opening portion of an rlogin from host *rtsg* to host *csam*.

IP rtsg.1023 > csam.login: Flags [S], seq 768512:768512, win 4096, opts [mss 1024]
IP csam.login > rtsg.1023: Flags [S.], seq, 947648:947648, ack 768513, win 4096, opts [mss 1024]
IP rtsg.1023 > csam.login: Flags [.], ack 1, win 4096
IP rtsg.1023 > csam.login: Flags [P.], seq 1:2, ack 1, win 4096, length 1
IP csam.login > rtsg.1023: Flags [.], ack 2, win 4096
IP rtsg.1023 > csam.login: Flags [P.], seq 2:21, ack 1, win 4096, length 19
IP csam.login > rtsg.1023: Flags [P.], seq 1:2, ack 21, win 4077, length 1
IP csam.login > rtsg.1023: Flags [P.], seq 2:3, ack 21, win 4077, urg 1, length 1
IP csam.login > rtsg.1023: Flags [P.], seq 3:4, ack 21, win 4077, urg 1, length 1

The first line says that TCP port 1023 on rtsg sent a packet to port *login* on csam. The **S** indicates that the *SYN* flag was set. The packet sequence number was 768512 and it contained no data. (The notation is 'first:last' which means 'sequence numbers *first* up to but not including *last*'.) There was no piggy-backed ACK, the available receive window was 4096 bytes and there was a max-segment-size option requesting an MSS of 1024 bytes.

Csam replies with a similar packet except it includes a piggy-backed ACK for rtsg's SYN. Rtsg then ACKs csam's SYN. The '.' means the ACK flag was set. The packet contained no data so there is no data sequence number or length. Note that the ACK sequence number is a small integer (1). The first time *tcpdump* sees a TCP 'conversation', it prints the sequence number from the packet. On subsequent packets of the conversation, the difference between the current packet's sequence number and this initial sequence number is printed. This means that sequence numbers after the first can be interpreted as relative byte positions in the conversation's data stream (with the first data byte each direction being '1'). '-S' will override this feature, causing the original sequence numbers to be output.

On the 6th line, rtsg sends csam 19 bytes of data (bytes 2 through 20 in the rtsg -> csam side of the conversation). The PSH flag is set in the packet. On the 7th line, csam says it's received data sent by rtsg up to but not including byte 21. Most of this data is apparently sitting in the socket buffer since csam's receive window has gotten 19 bytes smaller. Csam also sends one byte of data to rtsg in this packet. On the 8th and 9th lines, csam sends two bytes of urgent, pushed data to rtsg.

If the snapshot was small enough that *tcpdump* didn't capture the full TCP header, it interprets as much of the header as it can and then reports ''[|*tcp*]'' to indicate the remainder could not be interpreted. If the header contains a bogus option (one with a length that's either too small or beyond the end of the header), *tcpdump* reports it as ''[*bad opt*]'' and does not interpret any further options (since it's impossible to tell where they start). If the header length indicates options are present but the IP datagram length is not long enough for the options to actually be there, *tcpdump* reports it as ''[*bad hdr*

length]’’.

Capturing TCP packets with particular flag combinations (SYN-ACK, URG-ACK, etc.)

There are 8 bits in the control bits section of the TCP header:

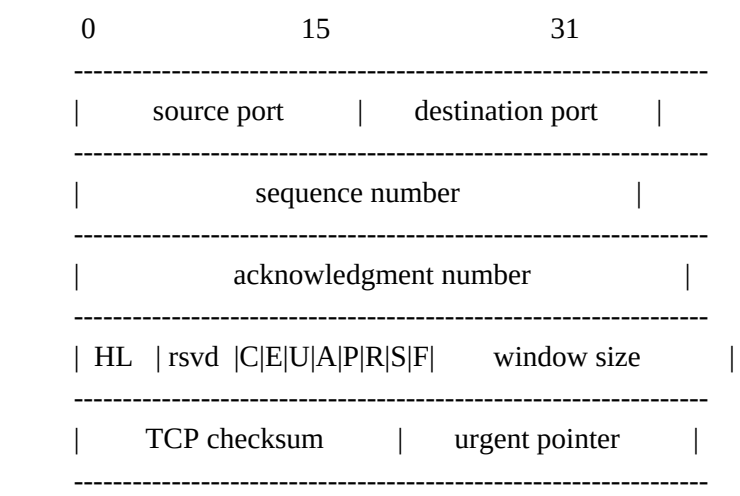
CWR | *ECE* | *URG* | *ACK* | *PSH* | *RST* | *SYN* | *FIN*

Let’s assume that we want to watch packets used in establishing a TCP connection. Recall that TCP uses a 3-way handshake protocol when it initializes a new connection; the connection sequence with regard to the TCP control bits is

1) Caller sends SYN
2) Recipient responds with SYN, ACK
3) Caller sends ACK

Now we’re interested in capturing packets that have only the SYN bit set (Step 1). Note that we don’t want packets from step 2 (SYN-ACK), just a plain initial SYN. What we need is a correct filter expression for *tcpdump*.

Recall the structure of a TCP header without options:



A TCP header usually holds 20 octets of data, unless options are present. The first line of the graph contains octets 0 - 3, the second line shows octets 4 - 7 etc.

Starting to count with 0, the relevant TCP control bits are contained in octet 13:

```
 0           7|           15|           23|           31
----------------|---------------|---------------|----------------
|  HL   | rsvd  |C|E|U|A|P|R|S|F|       window size           |
----------------|---------------|---------------|----------------
|             |  13th octet   |             |             |
```

Let’s have a closer look at octet no. 13:

```
                |             |
                |---------------|
                |C|E|U|A|P|R|S|F|
                |---------------|
                |7   5   3     0|
```

These are the TCP control bits we are interested in. We have numbered the bits in this octet from 0 to 7, right to left, so the PSH bit is bit number 3, while the URG bit is number 5.

Recall that we want to capture packets with only SYN set. Let’s see what happens to octet 13 if a TCP datagram arrives with the SYN bit set in its header:

```
                |C|E|U|A|P|R|S|F|
                |---------------|
                |0 0 0 0 0 0 1 0|
                |---------------|
                |7 6 5 4 3 2 1 0|
```

Looking at the control bits section we see that only bit number 1 (SYN) is set.

Assuming that octet number 13 is an 8-bit unsigned integer in network byte order, the binary value of this octet is

```
     00000010
```

and its decimal representation is

```
  7     6     5     4     3     2     1     0
0*2 + 0*2 + 0*2 + 0*2 + 0*2 + 0*2 + 1*2 + 0*2  =  2
```

We’re almost done, because now we know that if only SYN is set, the value of the 13th octet in the TCP header, when interpreted as a 8-bit unsigned integer in network byte order, must be exactly 2.

This relationship can be expressed as

tcp[13] == 2

We can use this expression as the filter for *tcpdump* in order to watch packets which have only SYN set:

tcpdump -i xl0 tcp[13] == 2

The expression says "let the 13th octet of a TCP datagram have the decimal value 2", which is exactly what we want.

Now, let’s assume that we need to capture SYN packets, but we don’t care if ACK or any other TCP control bit is set at the same time. Let’s see what happens to octet 13 when a TCP datagram with SYN-ACK set arrives:

```
|C|E|U|A|P|R|S|F|
|---------------|
|0 0 0 1 0 0 1 0|
|---------------|
|7 6 5 4 3 2 1 0|
```

Now bits 1 and 4 are set in the 13th octet. The binary value of octet 13 is

00010010

which translates to decimal

```
  7    6    5    4    3    2    1    0
0*2 + 0*2 + 0*2 + 1*2 + 0*2 + 0*2 + 1*2 + 0*2   = 18
```

Now we can’t just use ’tcp[13] == 18’ in the *tcpdump* filter expression, because that would select only those packets that have SYN-ACK set, but not those with only SYN set. Remember that we don’t care if ACK or any other control bit is set as long as SYN is set.

In order to achieve our goal, we need to logically AND the binary value of octet 13 with some other value to preserve the SYN bit. We know that we want SYN to be set in any case, so we’ll logically AND the value in the 13th octet with the binary value of a SYN:

```
    00010010 SYN-ACK              00000010 SYN
AND 00000010 (we want SYN)   AND  00000010 (we want SYN)
```

```
        --------              --------
   =    00000010         =    00000010
```

We see that this AND operation delivers the same result regardless whether ACK or another TCP control bit is set. The decimal representation of the AND value as well as the result of this operation is 2 (binary 00000010), so we know that for packets with SYN set the following relation must hold true:

((value of octet 13) AND (2)) == (2)

This points us to the *tcpdump* filter expression

tcpdump -i xl0 'tcp[13] & 2 == 2'

Some offsets and field values may be expressed as names rather than as numeric values. For example tcp[13] may be replaced with tcp[tcpflags]. The following TCP flag field values are also available: tcp-fin, tcp-syn, tcp-rst, tcp-push, tcp-ack, tcp-urg, tcp-ece and tcp-cwr.

This can be demonstrated as:

tcpdump -i xl0 'tcp[tcpflags] & tcp-push != 0'

Note that you should use single quotes or a backslash in the expression to hide the AND ('&') special character from the shell.

UDP Packets

UDP format is illustrated by this rwho packet:

actinide.who > broadcast.who: udp 84

This says that port *who* on host *actinide* sent a UDP datagram to port *who* on host *broadcast*, the Internet broadcast address. The packet contained 84 bytes of user data.

Some UDP services are recognized (from the source or destination port number) and the higher level protocol information printed. In particular, Domain Name service requests (RFC 1034/1035) and Sun RPC calls (RFC 1050) to NFS.

TCP or UDP Name Server Requests

(N.B.:The following description assumes familiarity with the Domain Service protocol described in RFC 1035. If you are not familiar with the protocol, the following description will appear to be written in Greek.)

Name server requests are formatted as

src > dst: id op? flags qtype qclass name (len)
h2opolo.1538 > helios.domain: 3+ A? ucbvax.berkeley.edu. (37)

Host *h2opolo* asked the domain server on *helios* for an address record (qtype=A) associated with the name *ucbvax.berkeley.edu.* The query id was '3'. The '+' indicates the *recursion desired* flag was set. The query length was 37 bytes, excluding the TCP or UDP and IP protocol headers. The query operation was the normal one, *Query*, so the op field was omitted. If the op had been anything else, it would have been printed between the '3' and the '+'. Similarly, the qclass was the normal one, *C_IN*, and omitted. Any other qclass would have been printed immediately after the 'A'.

A few anomalies are checked and may result in extra fields enclosed in square brackets: If a query contains an answer, authority records or additional records section, *ancount*, *nscount*, or *arcount* are printed as '[*n*a]', '[*n*n]' or '[*n*au]' where *n* is the appropriate count. If any of the response bits are set (AA, RA or rcode) or any of the 'must be zero' bits are set in bytes two and three, '[b2&3=*x*]' is printed, where *x* is the hex value of header bytes two and three.

TCP or UDP Name Server Responses

Name server responses are formatted as

src > dst: id op rcode flags a/n/au type class data (len)
helios.domain > h2opolo.1538: 3 3/3/7 A 128.32.137.3 (273)
helios.domain > h2opolo.1537: 2 NXDomain* 0/1/0 (97)

In the first example, *helios* responds to query id 3 from *h2opolo* with 3 answer records, 3 name server records and 7 additional records. The first answer record is type A (address) and its data is internet address 128.32.137.3. The total size of the response was 273 bytes, excluding TCP or UDP and IP headers. The op (Query) and response code (NoError) were omitted, as was the class (C_IN) of the A record.

In the second example, *helios* responds to query 2 with a response code of non-existent domain (NXDomain) with no answers, one name server and no authority records. The '*' indicates that the *authoritative answer* bit was set. Since there were no answers, no type, class or data were printed.

Other flag characters that might appear are '-' (recursion available, RA, *not* set) and '|' (truncated message, TC, set). If the 'question' section doesn't contain exactly one entry, '[*n*q]' is printed.

SMB/CIFS decoding

tcpdump now includes fairly extensive SMB/CIFS/NBT decoding for data on UDP/137, UDP/138 and TCP/139. Some primitive decoding of IPX and NetBEUI SMB data is also done.

By default a fairly minimal decode is done, with a much more detailed decode done if -v is used. Be

warned that with -v a single SMB packet may take up a page or more, so only use -v if you really want all the gory details.

For information on SMB packet formats and what all the fields mean see https://download.samba.org/pub/samba/specs/ and other online resources. The SMB patches were written by Andrew Tridgell (tridge@samba.org).

NFS Requests and Replies

Sun NFS (Network File System) requests and replies are printed as:

src.sport > dst.nfs: NFS request xid xid len op args
src.nfs > dst.dport: NFS reply xid xid reply stat len op results

```
sushi.1023 > wrl.nfs: NFS request xid 26377
        112 readlink fh 21,24/10.73165
wrl.nfs > sushi.1023: NFS reply xid 26377
        reply ok 40 readlink "../var"
sushi.1022 > wrl.nfs: NFS request xid 8219
        144 lookup fh 9,74/4096.6878 "xcolors"
wrl.nfs > sushi.1022: NFS reply xid 8219
        reply ok 128 lookup fh 9,74/4134.3150
```

In the first line, host *sushi* sends a transaction with id *26377* to *wrl*. The request was 112 bytes, excluding the UDP and IP headers. The operation was a *readlink* (read symbolic link) on file handle (*fh*) 21,24/10.731657119. (If one is lucky, as in this case, the file handle can be interpreted as a major,minor device number pair, followed by the inode number and generation number.) In the second line, *wrl* replies ‘ok’ with the same transaction id and the contents of the link.

In the third line, *sushi* asks (using a new transaction id) *wrl* to lookup the name ‘*xcolors*’ in directory file 9,74/4096.6878. In the fourth line, *wrl* sends a reply with the respective transaction id.

Note that the data printed depends on the operation type. The format is intended to be self explanatory if read in conjunction with an NFS protocol spec. Also note that older versions of tcpdump printed NFS packets in a slightly different format: the transaction id (xid) would be printed instead of the non-NFS port number of the packet.

If the -v (verbose) flag is given, additional information is printed. For example:

```
sushi.1023 > wrl.nfs: NFS request xid 79658
        148 read fh 21,11/12.195 8192 bytes @ 24576
```

```
wrl.nfs > sushi.1023: NFS reply xid 79658
        reply ok 1472 read REG 100664 ids 417/0 sz 29388
```

(-v also prints the IP header TTL, ID, length, and fragmentation fields, which have been omitted from this example.) In the first line, *sushi* asks *wrl* to read 8192 bytes from file 21,11/12.195, at byte offset 24576. *Wrl* replies ‘ok’; the packet shown on the second line is the first fragment of the reply, and hence is only 1472 bytes long (the other bytes will follow in subsequent fragments, but these fragments do not have NFS or even UDP headers and so might not be printed, depending on the filter expression used). Because the -v flag is given, some of the file attributes (which are returned in addition to the file data) are printed: the file type (‘‘REG’’, for regular file), the file mode (in octal), the UID and GID, and the file size.

If the -v flag is given more than once, even more details are printed.

NFS reply packets do not explicitly identify the RPC operation. Instead, *tcpdump* keeps track of ‘‘recent’’ requests, and matches them to the replies using the transaction ID. If a reply does not closely follow the corresponding request, it might not be parsable.

AFS Requests and Replies

Transarc AFS (Andrew File System) requests and replies are printed as:

```
src.sport > dst.dport: rx packet-type
src.sport > dst.dport: rx packet-type service call call-name args
src.sport > dst.dport: rx packet-type service reply call-name args

elvis.7001 > pike.afsfs:
        rx data fs call rename old fid 536876964/1/1 ".newsrc.new"
        new fid 536876964/1/1 ".newsrc"
pike.afsfs > elvis.7001: rx data fs reply rename
```

In the first line, host elvis sends a RX packet to pike. This was a RX data packet to the fs (fileserver) service, and is the start of an RPC call. The RPC call was a rename, with the old directory file id of 536876964/1/1 and an old filename of ‘.newsrc.new’, and a new directory file id of 536876964/1/1 and a new filename of ‘.newsrc’. The host pike responds with a RPC reply to the rename call (which was successful, because it was a data packet and not an abort packet).

In general, all AFS RPCs are decoded at least by RPC call name. Most AFS RPCs have at least some of the arguments decoded (generally only the ‘interesting’ arguments, for some definition of interesting).

The format is intended to be self-describing, but it will probably not be useful to people who are not familiar with the workings of AFS and RX.

If the -v (verbose) flag is given twice, acknowledgement packets and additional header information is printed, such as the RX call ID, call number, sequence number, serial number, and the RX packet flags.

If the -v flag is given twice, additional information is printed, such as the RX call ID, serial number, and the RX packet flags. The MTU negotiation information is also printed from RX ack packets.

If the -v flag is given three times, the security index and service id are printed.

Error codes are printed for abort packets, with the exception of Ubik beacon packets (because abort packets are used to signify a yes vote for the Ubik protocol).

AFS reply packets do not explicitly identify the RPC operation. Instead, *tcpdump* keeps track of ‘‘recent’’ requests, and matches them to the replies using the call number and service ID. If a reply does not closely follow the corresponding request, it might not be parsable.

KIP AppleTalk (DDP in UDP)

AppleTalk DDP packets encapsulated in UDP datagrams are de-encapsulated and dumped as DDP packets (i.e., all the UDP header information is discarded). The file */etc/atalk.names* is used to translate AppleTalk net and node numbers to names. Lines in this file have the form

number name

```
1.254       ether
16.1        icsd-net
1.254.110   ace
```

The first two lines give the names of AppleTalk networks. The third line gives the name of a particular host (a host is distinguished from a net by the 3rd octet in the number - a net number *must* have two octets and a host number *must* have three octets.) The number and name should be separated by whitespace (blanks or tabs). The */etc/atalk.names* file may contain blank lines or comment lines (lines starting with a ‘#’).

AppleTalk addresses are printed in the form

net.host.port

```
144.1.209.2 > icsd-net.112.220
office.2 > icsd-net.112.220
```

jssmag.149.235 > icsd-net.2

(If the */etc/atalk.names* doesn't exist or doesn't contain an entry for some AppleTalk host/net number, addresses are printed in numeric form.) In the first example, NBP (DDP port 2) on net 144.1 node 209 is sending to whatever is listening on port 220 of net icsd node 112. The second line is the same except the full name of the source node is known ('office'). The third line is a send from port 235 on net jssmag node 149 to broadcast on the icsd-net NBP port (note that the broadcast address (255) is indicated by a net name with no host number - for this reason it's a good idea to keep node names and net names distinct in /etc/atalk.names).

NBP (name binding protocol) and ATP (AppleTalk transaction protocol) packets have their contents interpreted. Other protocols just dump the protocol name (or number if no name is registered for the protocol) and packet size.

NBP packets are formatted like the following examples:

```
icsd-net.112.220 > jssmag.2: nbp-lkup 190: "=:LaserWriter@*"
jssmag.209.2 > icsd-net.112.220: nbp-reply 190: "RM1140:LaserWriter@*" 250
techpit.2 > icsd-net.112.220: nbp-reply 190: "techpit:LaserWriter@*" 186
```

The first line is a name lookup request for laserwriters sent by net icsd host 112 and broadcast on net jssmag. The nbp id for the lookup is 190. The second line shows a reply for this request (note that it has the same id) from host jssmag.209 saying that it has a laserwriter resource named "RM1140" registered on port 250. The third line is another reply to the same request saying host techpit has laserwriter "techpit" registered on port 186.

ATP packet formatting is demonstrated by the following example:

```
jssmag.209.165 > helios.132: atp-req  12266<0-7> 0xae030001
helios.132 > jssmag.209.165: atp-resp 12266:0 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:1 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:2 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:3 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:4 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:5 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:6 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp*12266:7 (512) 0xae040000
jssmag.209.165 > helios.132: atp-req  12266<3,5> 0xae030001
helios.132 > jssmag.209.165: atp-resp 12266:3 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:5 (512) 0xae040000
jssmag.209.165 > helios.132: atp-rel  12266<0-7> 0xae030001
jssmag.209.133 > helios.132: atp-req* 12267<0-7> 0xae030002
```

Jssmag.209 initiates transaction id 12266 with host helios by requesting up to 8 packets (the '<0-7>'). The hex number at the end of the line is the value of the 'userdata' field in the request.

Helios responds with 8 512-byte packets. The ‘:digit’ following the transaction id gives the packet sequence number in the transaction and the number in parens is the amount of data in the packet, excluding the ATP header. The ‘*’ on packet 7 indicates that the EOM bit was set.

Jssmag.209 then requests that packets 3 & 5 be retransmitted. Helios resends them then jssmag.209 releases the transaction. Finally, jssmag.209 initiates the next request. The ‘*’ on the request indicates that XO (‘exactly once’) was *not* set.

BACKWARD COMPATIBILITY

The TCP flag names **tcp-ece** and **tcp-cwr** became available when linking with libpcap 1.9.0 or later.

SEE ALSO

stty(1), **pcap**(3PCAP), **bpf**(4), **nit**(4P), **pcap-savefile**(5), **pcap-filter**(7), **pcap-tstamp**(7)

https://www.iana.org/assignments/media-types/application/vnd.tcpdump.pcap

AUTHORS

The original authors are:

Van Jacobson, Craig Leres and Steven McCanne, all of the Lawrence Berkeley National Laboratory, University of California, Berkeley, CA.

It is currently maintained by The Tcpdump Group.

The current version is available via HTTPS:

https://www.tcpdump.org/

The original distribution is available via anonymous ftp:

ftp://ftp.ee.lbl.gov/old/tcpdump.tar.Z

IPv6/IPsec support is added by WIDE/KAME project. This program uses OpenSSL/LibreSSL, under specific configurations.

BUGS

To report a security issue please send an e-mail to security@tcpdump.org.

To report bugs and other problems, contribute patches, request a feature, provide generic feedback etc. please see the file *CONTRIBUTING.md* in the tcpdump source tree root.

NIT doesn't let you watch your own outbound traffic, BPF will. We recommend that you use the latter.

On Linux systems with 2.0[.x] kernels:

packets on the loopback device will be seen twice;

packet filtering cannot be done in the kernel, so that all packets must be copied from the kernel in order to be filtered in user mode;

all of a packet, not just the part that's within the snapshot length, will be copied from the kernel (the 2.0[.x] packet capture mechanism, if asked to copy only part of a packet to userspace, will not report the true length of the packet; this would cause most IP packets to get an error from **tcpdump**);

capturing on some PPP devices won't work correctly.

We recommend that you upgrade to a 2.2 or later kernel.

Some attempt should be made to reassemble IP fragments or, at least to compute the right length for the higher level protocol.

Name server inverse queries are not dumped correctly: the (empty) question section is printed rather than real query in the answer section. Some believe that inverse queries are themselves a bug and prefer to fix the program generating them rather than *tcpdump*.

A packet trace that crosses a daylight savings time change will give skewed time stamps (the time change is ignored).

Filter expressions on fields other than those in Token Ring headers will not correctly handle source-routed Token Ring packets.

Filter expressions on fields other than those in 802.11 headers will not correctly handle 802.11 data packets with both To DS and From DS set.

ip6 proto should chase header chain, but at this moment it does not. **ip6 protochain** is supplied for this behavior.

Arithmetic expression against transport layer headers, like **tcp[0]**, does not work against IPv6 packets. It only looks at IPv4 packets.