TCPDUMP filters

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.

ber) preceded by one or more qualifiers. There are three different kinds of qualifier:

type  qualifiers say what kind of thing the id name or number refers to. Possible types are host, net and port. E.g., 'host foo', 'net 128.3', 'port 20'. If there is no type qualifier, host is assumed.

dir  qualifiers specify a particular transfer direction to and/or from id. Possible directions are src, dst, src or dst and src and dst. E.g., 'src foo', 'dst net 128.3', 'src or dst port ftp-data'. If there is no dir qualifier, src or dst is assumed. For 'null' link layers (i.e. point to point protocols such as slip) the inbound and outbound qualifiers can be used to specify a desired direction.

proto  qualifiers restrict the match to a particular protocol. Possible protocols are: ether, fddi, tr, ip, ip6, arp, rarp, decnet, tcp and udp. E.g., 'ether src foo', 'arp net 128.3', 'tcp port 21'. If there is no proto qualifier, all protocols consistent with the type are assumed. E.g., 'src foo' means '(ip or arp or rarp) src foo' (except the latter is not legal syntax), 'net bar' means '(ip or arp or rarp) net bar' and 'port 53' means '(tcp or udp) port 53'.

['fddi' is actually an alias for 'ether'; the parser treats them identically as meaning "the data link level used on the specified network interface." FDDI headers contain Ethernet-like source and destination addresses, and often contain Ethernet-like packet types, so you can filter on these FDDI fields just as with the analogous Ethernet fields. FDDI headers also contain other fields, but you cannot name them explicitly in a filter expression.

Similarly, 'tr' is an alias for 'ether'; the previous paragraph's statements about FDDI headers also apply to Token Ring headers.]

In addition to the above, there are some special 'primitive' keywords that don't follow the pattern: gateway, broadcast, less, greater and arithmetic expressions. All of these are described below.

tives. E.g., 'host foo and not port ftp and not port ftp-data'. To save typing, identical qualifier lists can be omitted. E.g., 'tcp dst port ftp or tcp dst port ftp-data' or 'tcp dst port domain'.

Allowable primitives are:

dst host host
True if the IPv4/v6 destination field of the packet is host, which may be either an address or a name.

src host host
True if the IPv4/v6 source field of the packet is host.

host host
True if either the IPv4/v6 source or destination of the packet is host. Any of the above host expressions can be prepended with the keywords, ip, arp, rarp, or ip6 as in:

ip host host
which is equivalent to:

ether proto ip and host host
If host is a name with multiple IP addresses, each address will be checked for a match.

ether dst ehost
True if the ethernet destination address is ehost. Ehost may be either a name from /etc/ethers or a number (see ethers(3N) for numeric format).

ether src ehost
True if the ethernet source address is ehost.

ether host ehost
True if either the ethernet source or destination address is ehost.

gateway host
True if the packet used host as a gateway. I.e., the ethernet source or destination address was host but neither the IP source nor the IP destination was host. Host must be a name and must be found both by the machine's host-name-to-IP-address resolution mechanisms (host name file, DNS, NIS, etc.) etc.) (An equivalent expression is ether host ehost and not host host which can be used with either names or numbers for host / ehost.) This syntax does not work in IPv6-enabled configuration at this moment.

dst net net
True if the IPv4/v6 destination address of the packet has a network number of net. Net may be either a name from /etc/networks or a network number (see networks(4) for details).

src net net
True if the IPv4/v6 source address of the
packet has a network number of \textit{net}.

\textbf{net net}
True if either the IPv4/v6 source or destination address of the packet has a network number of \textit{net}.

\textbf{net net mask netmask}
True if the IP address matches \textit{net} with the specific \textit{netmask}. May be qualified with \textit{src} or \textit{dst}. Note that this syntax is not valid for IPv6 \textit{net}.

\textbf{net net/len}
True if the IPv4/v6 address matches \textit{net} with a \textit{netmask} \textit{len} bits wide. May be qualified with \textit{src} or \textit{dst}.

\textbf{dst port port}
True if the packet is \textit{ip}/\textit{tcp}, \textit{ip}/\textit{udp}, \textit{ip6}/\textit{tcp} or \textit{ip6}/\textit{udp} and has a destination port value of \textit{port}. The \textit{port} can be a number or a name used in /etc/services (see \textit{tcp}(4P) and \textit{udp}(4P)). If a name is used, both the port number and protocol are checked. If a number or ambiguous name is used, only the port number is checked (e.g., \textbf{dst port 513} will print both tcp/login traffic and udp/who traffic, and \textbf{port domain} will print both tcp/domain and udp/domain traffic).

\textbf{src port port}
True if the packet has a source port value of \textit{port}.

True if either the source or destination port of the packet is \textit{port}. Any of the above port expressions can be prepended with the keywords, \textit{tcp} or \textit{udp}, as in:

\textbf{tcp src src port port}
which matches only tcp packets whose source port is \textit{port}.

\textbf{less length}
True if the packet has a length less than or equal to \textit{length}. This is equivalent to:
\begin{itemize}
  \item \textit{len} <= \textit{length}.
\end{itemize}

\textbf{greater length}
True if the packet has a length greater than or equal to \textit{length}. This is equivalent to:
\begin{itemize}
  \item \textit{len} >= \textit{length}.
\end{itemize}

\textbf{ip proto protocol}
True if the packet is an IPv4 packet of protocol \textit{protocol}. \textit{Protocol} can be a number or one of the names \textit{icmp}, \textit{icmp6}, \textit{igmp}, \textit{igmpv6}, \textit{pim}, \textit{ah}, \textit{esp}, \textit{vmp}, \textit{udp}, or \textit{tcp}. Note that the identifiers \textit{tcp}, \textit{udp}, and \textit{icmp} are also keywords and must be escaped via backslash (\textbackslash{}), which is \textbackslash{} in the C-shell. Note that this primitive does not chase the protocol header chain.

\textbf{ip6 proto protocol}
True if the packet is an IPv6 packet of protocol \textit{protocol}. Note that this primitive does not chase the protocol header chain.

\textbf{ip6 protochain protocol}
True if the packet is IPv6 packet, and contains protocol header with type \textit{protocol} in its protocol header chain. For example, \textbf{ip6 protochain 6} matches any IPv6 packet with TCP protocol header in the protocol header chain. The packet may contain, for example, authentication header, routing header, or hop-by-hop option header, between IPv6 header and TCP header. The BPF code emitted by this primitive is complex and cannot be optimized by BPF optimizer code in tcpdump, so this can be somewhat slow.

\textbf{ip protochain protocol}
Equivalent to \textbf{ip6 protochain protocol}, but True if the packet is an ethernet broadcast packet. The \textit{ether} keyword is optional.

\textbf{ip broadcast}
True if the packet is an IP broadcast packet. It checks for both the all-zeros and all-ones broadcast conventions, and looks up the local subnet mask.

\textbf{ether multicast}
True if the packet is an ethernet multicast packet. The \textit{ether} keyword is optional. This is shorthand for \textit{'ether[0] & 1 != 0'}.

\textbf{ip multicast}
True if the packet is an IP multicast packet.

\textbf{ip6 multicast}
True if the packet is an IPv6 multicast packet.

\textbf{ether proto protocol}
True if the packet is of ether type \textit{protocol}. \textit{Protocol} can be a number or one of the names \textit{ip6}, \textit{ipp}, \textit{rarp}, \textit{rar}, \textit{auth}, \textit{authp}, \textit{dec net}, \textit{sca}, \textit{lat}, \textit{sapd}, \textit{map}, \textit{iso}, \textit{stp}, \textit{ipx}, or \textit{netbeui}. Note these identifiers are also keywords and must be escaped via backslash (\textbackslash{}).

In the case of FDDI (e.g., `fddi protocol arp`) and Token Ring (e.g., `tr protocol arp`), for most of those protocols, the protocol identification comes from the 802.2 Logical Link Control (LLC) header, which is usually layered on top of the FDDI or Token Ring header.

When filtering for most protocol identifiers on FDDI or Token Ring, tcpdump checks only the protocol ID field of an LLC header in
so-called SNAP format with an Organizational Unit Identifier (OUI) of 0x000000, for encapsulated Ethernet; it doesn’t check whether the packet is in SNAP format with an OUI of 0x000000.

The exceptions are *iso*, for which it checks the DSAP (Destination Service Access Point) and SSAP (Source Service Access Point) fields of the LLC header, *stp* and *netbeui*, packet with an OUI of 0x080007 and the Appletalk etype.

In the case of Ethernet, *tcpdump* checks the Ethernet type field for most of those protocols; the exceptions are *iso*, *sap*, and *netbeui*, for which it checks for an 802.3 frame and then checks the LLC header as it does for FDDI and Token Ring, *atalk*, where it checks both for the Appletalk etype in an Ethernet frame and for a SNAP-format packet as it does for FDDI and Token Ring, *aarp*, where it checks for the Appletalk ARP etype in either an Ethernet frame or an 802.2 SNAP frame with an OUI of 0x000000, and *ipx*, where it checks for the IPX etype in an Ethernet frame, the 802.3 with no LLC header encapsulation of IPX, and the IPX etype in a SNAP frame.]

decnet src host
True if the DECNET source address is *host*, which may be an address of the form "10.123", or a DECNET host name. [DECNET host name support is only available on Ultrix systems that are configured to run DECNET.]

decnet dst host
True if the DECNET destination address is *host*.

decnet host host
True if either the DECNET source or destination address is *host*.

ip, ip6, arp, rarp, atalk, aarp, decnet, iso, stp, ipx, netbeui
Abbreviations for:
ether proto *p*
where *p* is one of the above protocols.

lat, moprc, mopdl
Abbreviations for:
ether proto *p*
where *p* is one of the above protocols. Note that *tcpdump* does not currently know how to parse these protocols.

vlan [vlan_id]
True if the packet is an IEEE 802.1Q VLAN packet. If [vlan_id] is specified, only encountered in expressions changes the decoding offsets for the remainder of expression on the assumption that the packet is a VLAN packet.

tcp, udp, icmp
Abbreviations for:
ip proto *p* or ip6 proto *p*
where *p* is one of the above protocols.

iso proto protocol
True if the packet is an OSI packet of protocol type *protocol*. *Protocol* can be a number or one of the names *clnp*, *esis*, or *isis*.

clnp, esis, isis
Abbreviations for:
iso proto *p*
where *p* is one of the above protocols. Note that *tcpdump* does an incomplete job of parsing these protocols.

expr relop expr
True if the relation holds, where *relop* is one of >, <, >=, <=, =, !=, and *expr* is an arithmetic expression composed of integer constants (expressed in standard C syntax), the normal binary operators [/+, -, *, /, &, &], a length operator, and special packet data accessors. To access data inside the packet, use the following syntax:
**proto [ expr : size ]**
*Proto* is one of *ether*, *fddi*, *tr*, *ip*, *arp*, *rarp*, *tcp*, *udp*, *icmp* or *ip6*, and indicates the protocol layer for the index operation. Note that *tcp*, *udp* and other upper-layer protocol types only apply to IPv4, not IPv6 (this will be fixed in the future). The byte offset, relative to the indicated protocol layer, is given by *expr*. Size is optional and indicates the number of bytes in the field of interest; it can be either one, two, or four, and defaults to one. The length operator, indicated by the keyword *len*, gives the length of the packet.

For example, `ether[0] & 1 != 0` catches all multicast traffic. The expression `ip[0] & 0xff != 0` catches all IP packets with options. The expression `ip[6:2] & 0x1fff == 0` catches unfragmented datagrams and fragment zero of fragmented datagrams. This always means the first byte of the TCP header, and never means the first byte of an intervening fragment.

Some offsets and field values may be expressed as names rather than as numeric values. The following protocol header field offsets are available: *icmp-type* (ICMP type field), *icmp-code* (ICMP code field), and *tcphopflags* (TCP flags field).

The following ICMP type field values are available: *icmp-echoreply*, *icmp-unreach*, *icmp-sourcequench*, *icmp-redirect*, *icmp-echo*, *icmp-routeradv*, *icmp-router solicit*, *icmp-timex*,
The following TCP flags field values are available: tcp-fin, tcp-syn, tcp-rst, tcp-push, tcp-ack, tcp-urg.

Primitives may be combined using:

- A parenthesized group of primitives and operators (parentheses are special to the Shell and must be escaped).
- Negation (‘!’ or `not`).
- Concatenation (‘&&’ or `and`).
- Alternation (‘||’ or `or`).

Negation has highest precedence. Alternation and concatenation have equal precedence and associate left to right. Note that explicit and tokens, not juxtaposition, are now required for concatenation.

If an identifier is given without a keyword, the most recent keyword is assumed. For example, `not host vs and ace` is short for `not host vs and host ace` which should not be confused with `not ( host vs or ace )`.

Expression arguments can be passed to `tcpdump` as either a single argument or as multiple arguments, whichever is more convenient. Generally, if the expression contains Shell metacharacters, it is before being parsed.

### EXAMPLES

To print all packets arriving at or departing from `sun` down:
```
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 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'
```