Filtering expression syntax
[WinPcap user's manual]

Note: this document has been drawn from the tcpdump man page. The original version can be found at www.tcpdump.org.

wpcap filters are based on a declarative predicate syntax. A filter is an ASCII string containing a filtering *expression*. pcap_compile() takes the expression and translates it in a program for the kernel-level packet filter.

The expression selects which packets will be dumped. If no expression is given, all packets on the net will be accepted by the kernel-level filtering engine. Otherwise, only packets for which *expression* is `true' will be accepted.

The *expression* consists of one or more *primitives.* Primitives usually consist of an *id* (name or number) 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 protos 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.

More complex filter expressions are built up by using the words **and**, **or** and **not** to combine primitives. 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 ftp-data or domain' is exactly the same as `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.) and by the machine's host-name-to-Ethernet-address resolution mechanism (/etc/ethers, 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 *net*.

net *net*
: True if either the IPv4/v6 source or destination address of the packet has a network number of *net*.

net *net* **mask** *netmask*
: True if the IP address matches *net* with the specific *netmask*. May be qualified with **src** or **dst**. Note that this syntax is not valid for IPv6 *net*.

net *net*/*len*
: True if the IPv4/v6 address matches *net* with a netmask *len* bits wide. May be qualified with **src** or **dst**.

dst port *port*
: True if the packet is ip/tcp, ip/udp, ip6/tcp or ip6/udp and has a destination port value of *port*. The *port* can be a number or a name used in /etc/services (see *tcp*(4P) and *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., **dst port 513** will print both tcp/login traffic and udp/who traffic, and **port domain** will print both tcp/domain and udp/domain traffic).

src port *port*
: True if the packet has a source port value of *port*.

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

```
tcp src port port
```

which matches only tcp packets whose source port is *port*.

less *length*
: True if the packet has a length less than or equal to *length*. This is equivalent to:

```
len <= length.
```

greater *length*
: True if the packet has a length greater than or equal to *length*. This is equivalent to:

```
len >= length.
```

ip proto *protocol*
: True if the packet is an IP packet (see *ip*(4P)) of protocol type *protocol*. *Protocol* can be a number or one of the names *icmp*, *icmp6*, *igmp*, *igrp*, *pim*, *ah*, *esp*, *vrrp*, *udp*, or *tcp*. Note that the identifiers *tcp*, *udp*, and *icmp* are also keywords and must be escaped via backslash (\), which is \\ in the C-shell. Note that this primitive does not chase the protocol header chain.

ip6 proto *protocol*
: True if the packet is an IPv6 packet of protocol type *protocol*. Note that this primitive does not chase the protocol header chain.

ip6 protochain *protocol*
: True if the packet is IPv6 packet, and contains protocol header with type *protocol* in its protocol header chain. For example,

```
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.

ip protochain *protocol*
Equivalent to **ip6 protochain** *protocol*, but this is for IPv4.

ether broadcast
True if the packet is an ethernet broadcast packet. The *ether* keyword is optional.

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

ether multicast
True if the packet is an ethernet multicast packet. The *ether* keyword is optional. This is shorthand for `**ether[0] & 1 != 0**'.

ip multicast
True if the packet is an IP multicast packet.

ip6 multicast
True if the packet is an IPv6 multicast packet.

ether proto *protocol*
True if the packet is of ether type *protocol*. *Protocol* can be a number or one of the names *ip*, *ip6*, *arp*, *rarp*, *atalk*, *aarp*, *decnet*, *sca*, *lat*, *mopdl*, *moprc*, *iso*, *stp*, *ipx*, or *netbeui*. Note these identifiers are also keywords and must be escaped via backslash (\).
[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*, where it checks the DSAP of the LLC header, and *atalk*, where it checks for a SNAP-format 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 IPX DSAP in the LLC header, 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 true is the packet has the specified *vlan_id*. Note that the first **vlan** keyword encountered in *expression* 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] & 0xf != 5**' catches all IP packets with options. The expression \`**ip[6:2] & 0x1fff = 0**' catches only unfragmented datagrams and frag zero of fragmented datagrams. This check is implicitly applied to the **tcp** and **udp** index operations. For instance, **tcp[0]** 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: **icmptype** (ICMP type field), **icmpcode** (ICMP code field), and **tcpflags** (TCP flags field).

The following ICMP type field values are available: **icmp-echoreply**, **icmp-unreach**, **icmp-sourcequench**, **icmp-redirect**, **icmp-echo**, **icmp-routeradvert**, **icmp-routersolicit**, **icmp-timxceed**, **icmp-paramprob**, **icmp-tstamp**, **icmp-tstampreply**, **icmp-ireq**, **icmp-ireqreply**, **icmp-maskreq**, **icmp-maskreply**.

The following TCP flags field values are available: **tcp-fin**, **tcp-syn**, **tcp-rst**, **tcp-push**, **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 easier to pass it as a single, quoted argument. Multiple arguments are concatenated with spaces before being parsed.