NAME

pcap-filter - packet filter syntax

DESCRIPTION

The *filter 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** , **port** and **portrange**. E.g., `host foo', `net 128.3', `port 20', `portrange 6000-6008'. 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**, **src and dst**, **addr1**, **addr2**, **addr3**, and **addr4**. 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. The **addr1**, **addr2**, **addr3**, and **addr4** qualifiers are only valid for IEEE 802.11 Wireless LAN link layers. For some link layers, such as SLIP and the ``cooked'' Linux capture mode used for the ``any'' device and for some other device types, 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**, **wlan**, **ip**, **ip6**, **arp**, **rarp**, **decnet**, **tcp** and **udp**. E.g., `ether src foo', `arp net 128.3', `tcp port 21', `udp portrange 7000-7009', `wlan addr2 0:2:3:4:5:6'. 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' and `wlan' are aliases for `ether'; the previous paragraph's statements about FDDI headers also apply to Token Ring and 802.11 wireless LAN headers. For 802.11 headers, the destination address is the DA field and the source address is the SA field; the BSSID, RA, and TA fields aren't tested.]

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 the networks database (/etc/networks, etc.) or a network number. An IPv4 network number can be written as a dotted quad (e.g., 192.168.1.0), dotted triple (e.g., 192.168.1), dotted pair (e.g, 172.16), or single number (e.g., 10); the netmask is 255.255.255.255 for a dotted quad (which means that it's really a host match), 255.255.255.0 for a dotted triple, 255.255.0.0 for a dotted pair, or 255.0.0.0 for a single number. An IPv6 network number must be written out fully; the netmask is ff:ff:ff:ff:ff:ff:ff:ff, so IPv6 "network" matches are really always host matches, and a network match requires a netmask length.

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 IPv4 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*.

dst portrange *port1-port2*
True if the packet is ip/tcp, ip/udp, ip6/tcp or ip6/udp and has a destination port value between *port1* and *port2*. *port1* and *port2* are interpreted in the same fashion as the *port* parameter for **port**.

src portrange *port1-port2*
True if the packet has a source port value between *port1* and *port2*.

portrange *port1-port2*
True if either the source or destination port of the packet is between *port1* and *port2*.

Any of the above port or port range 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 IPv4 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 the BPF optimizer code, 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 IPv4 broadcast packet. It checks for both the all-zeroes and all-ones broadcast conventions, and looks up the subnet mask on the interface on which the capture is being done.

If the subnet mask of the interface on which the capture is being done is not available, either because the interface on which capture is being done has no netmask or because the capture is being done on the Linux "any" interface, which can capture on more than one interface, this check will not work correctly.

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 IPv4 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**'), Token Ring (e.g., `**tr protocol arp**'), and IEEE 802.11 wireless LANS (e.g., `**wlan 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, Token Ring, or 802.11 header.

When filtering for most protocol identifiers on FDDI, Token Ring, or 802.11, the filter 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 the filter checks the DSAP (Destination Service Access Point) and SSAP (Source Service Access Point) fields of the LLC header;

stp and **netbeui**
the filter checks the DSAP of the LLC header;

atalk the filter checks for a SNAP-format packet with an OUI of 0x080007 and the AppleTalk etype.

In the case of Ethernet, the filter checks the Ethernet type field for most of those protocols. The exceptions are:

iso, **stp**, and **netbeui**
the filter checks for an 802.3 frame and then checks the LLC header as it does for FDDI, Token Ring, and 802.11;

atalk the filter checks both for the AppleTalk etype in an Ethernet frame and for a SNAP-format packet as it does for FDDI, Token Ring, and 802.11;

aarp the filter checks for the AppleTalk ARP etype in either an Ethernet frame or an 802.2 SNAP frame with an OUI of 0x000000;

ipx the filter 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*.

ifname *interface*
True if the packet was logged as coming from the specified interface (applies only to packets logged by OpenBSD's or FreeBSD's **pf(4)**).

on *interface*
Synonymous with the **ifname** modifier.

rnr *num*
True if the packet was logged as matching the specified PF rule number (applies only to packets logged by OpenBSD's or FreeBSD's **pf(4)**).

rulenum *num*
Synonymous with the **rnr** modifier.

reason *code*
True if the packet was logged with the specified PF reason code. The known codes are: **match**, **bad-offset**, **fragment**, **short**, **normalize**, and **memory** (applies only to packets logged by OpenBSD's or FreeBSD's **pf(4)**).

rset *name*
True if the packet was logged as matching the specified PF ruleset name of an anchored ruleset (applies only to packets logged by OpenBSD's or FreeBSD's **pf(4)**).

ruleset *name*
Synonomous with the **rset** modifier.

srnr *num*
True if the packet was logged as matching the specified PF rule number of an anchored ruleset (applies only to packets logged by OpenBSD's or FreeBSD's **pf(4)**).

subrulenum *num*
Synonomous with the **srnr** modifier.

action *act*
True if PF took the specified action when the packet was logged. Known actions are: **pass** and **block** and, with later versions of **pf(4)**), **nat**, **rdr**, **binat** and **scrub** (applies only to packets logged by OpenBSD's or FreeBSD's **pf(4)**).

wlan addr1 *ehost*
True if the first IEEE 802.11 address is *ehost*.

wlan addr2 *ehost*
True if the second IEEE 802.11 address, if present, is *ehost*. The second address field is used in all frames except for CTS (Clear To Send) and ACK (Acknowledgment) control frames.

wlan addr3 *ehost*
True if the third IEEE 802.11 address, if present, is *ehost*. The third address field is used in management and data frames, but not in control frames.

wlan addr4 *ehost*
True if the fourth IEEE 802.11 address, if present, is *ehost*. The fourth address field is only used for WDS (Wireless Distribution System) frames.

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 not all applications using **pcap(3)** currently know how to parse these protocols.

type *wlan_type*
True if the IEEE 802.11 frame type matches the specified *wlan_type*. Valid *wlan_type*s are: **mgt**, **ctl** and **data**.

type *wlan_type* **subtype** *wlan_subtype*
True if the IEEE 802.11 frame type matches the specified *wlan_type* and frame subtype matches the specified *wlan_subtype*.

If the specified *wlan_type* is **mgt**, then valid *wlan_subtype*s are: **assoc-req**, **assoc-resp**, **reassoc-req**, **reassoc-resp**, **probe-req**, **probe-resp**, **beacon**, **atim**, **disassoc**, **auth** and **deauth**.

If the specified *wlan_type* is **ctl**, then valid *wlan_subtype*s are: **ps-poll**, **rts**, **cts**, **ack**, **cf-end** and **cf-end-ack**.

If the specified *wlan_type* is **data**, then valid *wlan_subtype*s are: **data**, **data-cf-ack**, **data-cf-poll**, **data-cf-ack-poll**, **null**, **cf-ack**, **cf-poll**, **cf-ack-poll**, **qos-data**, **qos-data-cf-ack**, **qos-data-cf-poll**, **qos-data-cf-ack-poll**, **qos**, **qos-cf-poll** and **qos-cf-ack-poll**.

subtype *wlan_subtype*
True if the IEEE 802.11 frame subtype matches the specified *wlan_subtype* and frame has the type to which the specified *wlan_subtype* belongs.

dir *dir*
True if the IEEE 802.11 frame direction matches the specified *dir*. Valid directions are: **nods**, **tods**, **fromds**, **dstods**, or a

numeric value.

vlan *[vlan_id]*
True if the packet is an IEEE 802.1Q VLAN packet. If *[vlan_id]* is specified, only true if 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. The **vlan** *[vlan_id]* expression may be used more than once, to filter on VLAN hierarchies. Each use of that expression increments the filter offsets by 4.

For example:
vlan 100 && vlan 200
filters on VLAN 200 encapsulated within VLAN 100, and
vlan && vlan 300 && ip
filters IPv4 protocols encapsulated in VLAN 300 encapsulated within any higher order VLAN.

mpls *[label_num]*
True if the packet is an MPLS packet. If *[label_num]* is specified, only true is the packet has the specified *label_num*. Note that the first **mpls** keyword encountered in *expression* changes the decoding offsets for the remainder of *expression* on the assumption that the packet is a MPLS-encapsulated IP packet. The **mpls** *[label_num]* expression may be used more than once, to filter on MPLS hierarchies. Each use of that expression increments the filter offsets by 4.

For example:
mpls 100000 && mpls 1024
filters packets with an outer label of 100000 and an inner label of 1024, and
mpls && mpls 1024 && host 192.9.200.1
filters packets to or from 192.9.200.1 with an inner label of 1024 and any outer label.

pppoed True if the packet is a PPP-over-Ethernet Discovery packet (Ethernet type 0x8863).

pppoes True if the packet is a PPP-over-Ethernet Session packet (Ethernet type 0x8864). Note that the first **pppoes** keyword encountered in *expression* changes the decoding offsets for the remainder of *expression* on the assumption that the packet is a PPPoE session packet.

For example:
pppoes && ip
filters IPv4 protocols encapsulated in PPPoE.

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.

l1, l2, iih, lsp, snp, csnp, psnp
Abbreviations for IS-IS PDU types.

vpi *n* True if the packet is an ATM packet, for SunATM on Solaris, with a virtual path identifier of *n*.

vci *n* True if the packet is an ATM packet, for SunATM on Solaris, with a virtual channel identifier of *n*.

lane True if the packet is an ATM packet, for SunATM on Solaris, and is an ATM LANE packet. Note that the first **lane** keyword encountered in *expression* changes the tests done in the remainder of *expression* on the assumption that the packet is either a LANE emulated Ethernet packet or a LANE LE Control packet. If **lane** isn't specified, the tests are done under the assumption that the packet is an LLC-encapsulated packet.

llc True if the packet is an ATM packet, for SunATM on Solaris, and is an LLC-encapsulated packet.

oamf4s True if the packet is an ATM packet, for SunATM on Solaris, and is a segment OAM F4 flow cell (VPI=0 & VCI=3).

oamf4e True if the packet is an ATM packet, for SunATM on Solaris, and is an end-to-end OAM F4 flow cell (VPI=0 & VCI=4).

oamf4 True if the packet is an ATM packet, for SunATM on Solaris, and is a segment or end-to-end OAM F4 flow cell (VPI=0 & (VCI=3 | VCI=4)).

oam True if the packet is an ATM packet, for SunATM on Solaris, and is a segment or end-to-end OAM F4 flow cell (VPI=0 & (VCI=3 | VCI=4)).

metac True if the packet is an ATM packet, for SunATM on Solaris, and is on a meta signaling circuit (VPI=0 & VCI=1).

bcc True if the packet is an ATM packet, for SunATM on Solaris, and is on a broadcast signaling circuit (VPI=0 & VCI=2).

sc True if the packet is an ATM packet, for SunATM on Solaris, and is on a signaling circuit (VPI=0 & VCI=5).

ilmic True if the packet is an ATM packet, for SunATM on Solaris, and is on an ILMI circuit (VPI=0 & VCI=16).

connectmsg
True if the packet is an ATM packet, for SunATM on Solaris, and is on a signaling circuit and is a Q.2931 Setup, Call Proceeding, Connect, Connect Ack, Release, or Release Done message.

metaconnect
True if the packet is an ATM packet, for SunATM on Solaris, and is on a meta signaling circuit and is a Q.2931 Setup, Call Proceeding, Connect, Release, or Release Done message.

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. Note that all comparisons are unsigned, so that, for example, 0x80000000 and 0xffffffff are > 0. To access data inside the packet, use the following syntax:

proto **[** *expr* **:** *size* **]**

Proto is one of **ether**, **fddi**, **tr**, **wlan**, **ppp**, **slip**, **link**, **ip**, **arp**, **rarp**, **tcp**, **udp**, **icmp**, **ip6** or **radio**, and indicates the protocol layer for the index operation. (**ether**, **fddi**, **wlan**, **tr**, **ppp**, **slip** and **link** all refer to the link layer. **radio** refers to the "radio header" added to some 802.11 captures.) 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 IPv4 packets with options. The expression `**ip[6:2] & 0x1fff = 0**' catches only unfragmented IPv4 datagrams and frag zero of fragmented IPv4 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-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)

EXAMPLES

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

host sundown

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

host helios and \(hot or ace \)

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

ip host ace and not helios

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

net ucb-ether

To select all ftp traffic through internet gateway *snup*:

gateway snup and (port ftp or ftp-data)

To select 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).

ip and not net *localnet*

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

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

To select 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.)

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

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

gateway snup and ip[2:2] > 576

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

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

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

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