MyWebUniversity
python
Rocky Linux
Manual Pages
FPDF Library
PDF generator
FPDF
Full credit is given to the above companies including the OS that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'pcap-filter.7'

$ man pcap-filter.7

```
PCAP-FILTER(7)        Miscellaneous Information Manual        PCAP-FILTER(7)

NAME

       pcap-filter - packet filter syntax

DESCRIPTION

       pcap_compile()  is used to compile a string into a filter program.  The

       resulting filter program can then be applied to some stream of  packets

       to  determine  which  packets  will  be  supplied  to pcap_loop(3PCAP),

       pcap_dispatch(3PCAP), pcap_next(3PCAP), or pcap_next_ex(3PCAP).

       The filter expression consists of one or more  primitives.   Primitives

       usually consist of an id (name or number) preceded by one or more qual?

       ifiers.  There are three different kinds of qualifier:

       type   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    dir qualifiers specify a particular transfer direction to and/or

              from id.  Possible directions are src, dst, src or dst, src  and

              dst,  ra,  ta, 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 ra, ta, addr1, addr2,
addr3, and addr4 qualifiers are only valid for IEEE 802.11 Wire?
less LAN link layers.

proto 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 identi?
cally 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 ex?
plicitly 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 wire?
less 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 (or equivalently: `&&', `||' and `!' respectively) 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 key?

    words, 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 IPv4 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 numerical MAC address of the

    form "xx:xx:xx:xx:xx:xx", "xx.xx.xx.xx.xx.xx", "xx-xx-xx-xx-xx-

    xx", "xxxx.xxxx.xxxx", "xxxxxxxxxxxx", or various mixes of ':',

    '.', and '-', where each "x" is a hex digit (0-9, a-f, or A-F).

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 resolu?

tion mechanisms (host name file, DNS, NIS, etc.) and by the ma?
chine'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 net?
work 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 IPv4 TCP, IPv4 UDP, IPv6 TCP or IPv6 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 IPv4 TCP, IPv4 UDP, IPv6 TCP or IPv6 UDP and has a destination port value between port1 and port2. port1 and port2 are interpreted in the same fashion as the port param?eter 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 (\).   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.

       proto protocol

              True  if  the  packet is an IPv4 or IPv6 packet of protocol type

              protocol.  Note that this primitive does not chase the  protocol

              header chain.

       tcp, udp, icmp

              Abbreviations for:

                   proto \protocol

              where protocol is one of the above protocols.

       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, authentica?

              tion header, routing header, or hop-by-hop  option  header,  be?

              tween  IPv6 header and TCP header.  The BPF code emitted by this

              primitive is complex and cannot be optimized by  the  BPF  opti?
```

mizer code, and is not supported by filter engines in the ker?
nel, so this can be somewhat slow, and may cause more packets to
be dropped.

ip protochain protocol

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

protochain protocol

True if the packet is an IPv4 or IPv6 packet of protocol type
protocol. Note that this primitive chases the protocol header
chain.

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 be?
ing 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 aarp, arp, atalk, decnet, ip, ip6,
ipx, iso, lat, loopback, mopdl, moprc, netbeui, rarp, sca or

stp. Note these identifiers (except loopback) are also keywords

and must be escaped via backslash (\).

[In the case of FDDI (e.g., `fddi proto \arp'), Token Ring

(e.g., `tr proto \arp'), and IEEE 802.11 wireless LANs (e.g.,

`wlan proto \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 Eth?

ernet 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.

ip, ip6, arp, rarp, atalk, aarp, decnet, iso, stp, ipx, netbeui

Abbreviations for:

ether proto \protocol

where protocol is one of the above protocols.

lat, moprc, mopdl

Abbreviations for:

ether proto \protocol

where protocol is one of the above protocols. Note that not all applications using pcap(3PCAP) currently know how to parse these protocols.

decnet src host

True if the DECnet source address is host, which may be an ad?dress 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.

llc   True if the packet has an 802.2 LLC header. This includes:

Ethernet packets with a length field rather than a type field that aren't raw NetWare-over-802.3 packets;

IEEE 802.11 data packets;

Token Ring packets (no check is done for LLC frames);

FDDI packets (no check is done for LLC frames);

LLC-encapsulated ATM packets, for SunATM on Solaris.

llc type

True if the packet has an 802.2 LLC header and has the specified type. type can be one of:

i   Information (I) PDUs

s	Supervisory (S) PDUs
u	Unnumbered (U) PDUs
rr	Receiver Ready (RR) S PDUs
rnr	Receiver Not Ready (RNR) S PDUs
rej	Reject (REJ) S PDUs
ui	Unnumbered Information (UI) U PDUs
ua	Unnumbered Acknowledgment (UA) U PDUs
disc	Disconnect (DISC) U PDUs
sabme	Set Asynchronous Balanced Mode Extended (SABME) U PDUs
test	Test (TEST) U PDUs
xid	Exchange Identification (XID) U PDUs
frmr	Frame Reject (FRMR) U PDUs

inbound

Packet was received by the host performing the capture rather
than being sent by that host. This is only supported for cer?
tain link-layer types, such as SLIP and the ``cooked'' Linux
capture mode used for the ``any'' device and for some other de?
vice types.

outbound

Packet was sent by the host performing the capture rather than
being received by that host. This is only supported for certain
link-layer types, such as SLIP and the ``cooked'' Linux capture
mode used for the ``any'' device and for some other device
types.

ifname interface

True if the packet was logged as coming from the specified in?
terface (applies only to packets logged by OpenBSD's or Free?
BSD'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, normal?

ize, 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 rule?

set name of an anchored ruleset (applies only to packets logged

by OpenBSD's or FreeBSD's pf(4)).

ruleset name

Synonymous 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

Synonymous 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 ra ehost

True if the IEEE 802.11 RA is ehost. The RA field is used in

all frames except for management frames.

wlan ta ehost

True if the IEEE 802.11 TA is ehost. The TA field is used in

all frames except for management frames and CTS (Clear To Send)

and ACK (Acknowledgment) control frames.

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 Distri? bution System) frames.

type wlan_type

True if the IEEE 802.11 frame type matches the specified wlan_type. Valid wlan_types 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_subtypes 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_subtypes are: ps-poll, rts, cts, ack, cf-end and cf-end-ack.

If the specified wlan_type is data, then valid wlan_subtypes 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 nu? meric value.

vlan [vlan_id]

True if the packet is an IEEE 802.1Q VLAN packet. If the op? tional vlan_id is specified, only true if the packet has the specified vlan_id. Note that the first vlan keyword encountered in an expression changes the decoding offsets for the remainder of the expression on the assumption that the packet is a VLAN packet. The `vlan [vlan_id]` keyword may be used more than once, to filter on VLAN hierarchies. Each use of that keyword 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 protocol encapsulated in VLAN 300 encapsulated within any higher order VLAN.

mpls [label_num]

True if the packet is an MPLS packet. If the optional label_num is specified, only true if the packet has the specified la? bel_num. Note that the first mpls keyword encountered in an ex? pression changes the decoding offsets for the remainder of the expression on the assumption that the packet is a MPLS-encapsu? lated IP packet. The `mpls [label_num]` keyword may be used more than once, to filter on MPLS hierarchies. Each use of that keyword 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 (Eth?

ernet type 0x8863).

pppoes [session_id]

True if the packet is a PPP-over-Ethernet Session packet (Ether?

net type 0x8864). If the optional session_id is specified, only

true if the packet has the specified session_id. Note that the

first pppoes keyword encountered in an expression changes the

decoding offsets for the remainder of the expression on the as?

sumption that the packet is a PPPoE session packet.

For example:

```
pppoes 0x27 && ip
```

filters IPv4 protocol encapsulated in PPPoE session id 0x27.

geneve [vni]

True if the packet is a Geneve packet (UDP port 6081). If the

optional vni is specified, only true if the packet has the spec?

ified vni. Note that when the geneve keyword is encountered in

an expression, it changes the decoding offsets for the remainder

of the expression on the assumption that the packet is a Geneve

packet.

For example:

```
geneve 0xb && ip
```

filters IPv4 protocol encapsulated in Geneve with VNI 0xb. This

will match both IPv4 directly encapsulated in Geneve as well as

IPv4 contained inside an Ethernet frame.

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 \protocol
```

where protocol 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 encoun?
tered in an expression changes the tests done in the remainder
of the 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.

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 Proceed?
ing, 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 Pro?
ceeding, 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.

The % and ^ operators are currently only supported for filtering
in the kernel on Linux with 3.7 and later kernels; on all other
systems, if those operators are used, filtering will be done in
user mode, which will increase the overhead of capturing packets
and may cause more packets to be dropped.

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, rela?
tive to the indicated protocol layer, is given by expr. Size is
optional and indicates the number of bytes in the field of in?
terest; 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 un?

fragmented IPv4 datagrams and frag zero of fragmented IPv4 data?

grams. This check is implicitly applied to the tcp and udp in?

dex operations. For instance, tcp[0] always means the first

byte of the TCP header, and never means the first byte of an in?

tervening 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), icmp6type

(ICMPv6 type field), icmpcode (ICMP code field), icmp6code

(ICMPv6 code field) and tcpflags (TCP flags field).

The following ICMP type field values are available: icmp-echore?

ply, icmp-unreach, icmp-sourcequench, icmp-redirect, icmp-echo,

icmp-routeradvert, icmp-routersolicit, icmp-timxceed, icmp-

paramprob, icmp-tstamp, icmp-tstampreply, icmp-ireq, icmp-ire?

qreply, icmp-maskreq, icmp-maskreply.

The following ICMPv6 type fields are available: icmp6-destina?

tionrunreach, icmp6-packettoobig, icmp6-timeexceeded, icmp6-pa?

rameterproblem, icmp6-echo, icmp6-echoreply, icmp6-multicastlis?

tenerquery, icmp6-multicastlistenerreportv1, icmp6-multicastlis?

tenerdone, icmp6-routersolicit, icmp6-routeradvert, icmp6-neigh?

borsolicit, icmp6-neighboradvert, icmp6-redirect, icmp6-router?

renum, icmp6-nodeinformationquery, icmp6-nodeinformationre?

sponse, icmp6-ineighbordiscoverysolicit, icmp6-ineighbordiscov?

eryadvert, icmp6-multicastlistenerreportv2, icmp6-homeagentdis?

coveryrequest, icmp6-homeagentdiscoveryreply, icmp6-mobilepre?

fixsolicit, icmp6-mobileprefixadvert, icmp6-certpathsolicit,

icmp6-certpathadvert, icmp6-multicastrouteradvert, icmp6-multi?

castroutersolicit, icmp6-multicastrouterterm.

The following TCP flags field values are available: tcp-fin, tcp-syn, tcp-rst, tcp-push, tcp-ack, tcp-urg, tcp-ece, tcp-cwr.

Primitives may be combined using:

A parenthesized group of primitives and operators.

Negation (`!' or `not').

Concatenation (`&&' or `and').

Alternation (`||' or `or').

Negation has the 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 IPv4 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 IPv4 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 the TCP packets with flags RST and ACK both set.   (i.e. se?
lect only the RST and ACK flags in the flags field, and if the result
is "RST and ACK both set", match)

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

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 IPv4 packets longer than 576 bytes sent through gateway
`snup':

```
gateway snup and ip[2:2] > 576
```

To select IPv4 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
icmp6[icmp6type] != icmp6-echo and icmp6[icmp6type] != icmp6-echoreply
```

SEE ALSO

pcap(3PCAP)

BUGS

To report a security issue please send an e-mail to security@tcp?
dump.org.

To report bugs and other problems, contribute patches, request a fea?
ture, provide generic feedback etc please see the file CONTRIBUTING.md
in the libpcap source tree root.

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. For example, to match

IPv6 fragments: `ip6 protochain 44'

Arithmetic expression against transport layer headers, like tcp[0],

does not work against IPv6 packets. It only looks at IPv4 packets.