Filter rules of wireshark (pcap filtering)

출처: http://www.tcpdump.org/manpages/pcap-filter.7.html

필터 표현식은 하나 이상의 기본원소들(primitives)로 이루어 집니다. 기본원소들(primitives)은 주로 id나 이름 혹은 숫자로 구성되는 1개나 여러 개의 수식어구(qualifiers)가 선행되어야 합니다.

3가지 종류의 수식어구(qualifiers)가 있습니다:

type

type 수식어구(qualifiers)는 ID나 이름이나 숫자가 의미하는 것을 말합니다. 가능한 타입은 **host**, **net**, **port**, **portrange**가 있습니다. 예를 들어, `host foo', `net 128.3', `port 20', `portrange 6000-6008' 만약 타입 수식어구가 없다면, **host**는 추정됩니다.

dir

dir 수식어구(qualifiers)는 id에서 특정 전송 방향을 지정합니다. 가능한 방향은 **src,dst, src** 혹은 **dst, src** 그리고 **dst, ra, ta, addr1, addr2, addr3** 그리고 **addr4** 입니다. 예를 들어, `src foo', `dst net 128.3', `src or dst port ftp-data'가 있습니다. 만약 dir 수식어구가 없는 경우, **src**나 **dst**는 수정됩니다. **ra, ta, addr1, addr2, addr3, addr4** 수식어구는IEEE 802.11 무선 랜 링크 층에서만 유효합니다. SLIP 및 "cooked" Linux 캡처 모드와 같은 일부 링크 계층의 경우 "모든" 장치 및 일부 다른 장치 유형에 사용되는 인바운드 및 아웃바운드 수식어구를 사용하여 원하는 방향을 지정할 수 있습니다.

proto

양성자 한정자는 특정 프로토콜로 매치를 제한합니다. 가능한 양성자는 ether, fdi, tr,wlan, ip6, ip6, rp, decnet, tcp, enet src foo이다. 예: src foo는 '(ip 또는 arp) src foo'(src foo)를 의미하며, net bar'는 'ip 또는 arp'를 의미합니다.

[fddi]는 실제로 'ether'의 별칭으로, 파서는 이들을 ``특정 네트워크 인터페이스에 사용되는 데이터 링크 수준"이라는 의미로 취급한다.'' FDDI 헤더는 이더넷과 유사한 소스 및 대상 주소를 포함하며 종종 이더넷과 유사한 패킷 유형을 포함하므로, 아날로그 이더넷 필드와 마찬가지로 이러한 FDDI 필드를 필터링할 수 있습니다. FDDI 헤더에는 다른 필드도 포함되어 있지만 필터 식에서 명시적으로 이름을 지정할 수는 없습니다.

마찬가지로, 'tr'과 'wlan'은 'ether'의 별칭이다. 이전 단락의 FDDI 헤더에 대한 진술은 토큰 링과 802.11 무선 LAN 헤더에도 적용된다. 802.11 헤더의 경우 대상 주소는 DA 필드이고 원본 주소는 SA 필드입니다. BSSID, RA 및 TA 필드는 테스트되지 않습니다.]

위 외에도, '게이트웨이', '방송', 덜, 더 큰, 산수 표현 등 패턴을 따르지 않는 특별한 '일차적' 키워드가 있다. 이러한 모든 사항은 아래에 설명되어 있습니다.

보다 복잡한 필터 표현식은 단어 및 기본 표현식을 조합하지 않고 단어로 구성됩니다. 예: '포트 foo 및 포트 ftp not port ftp and data'. 입력을 저장하려면 동일한 한정자 목록을 생략할 수 있습니다. 예: 'tcp dst port ftp 또는 ftp-data 또는 domain'은 'tcp dst port ftp 또는 tcp dst port'와 동일합니다.

허용되는 기본 요소는 다음과 같습니다:

**dst host** host

패킷의 IPv4/v6 대상 필드가 호스트(주소 또는 이름일 수 있음)인 경우 True입니다.

**src host** host

패킷의 IPv4/v6 소스 필드가 호스트인 경우 True입니다.

**host** host

패킷의 IPv4/v6 소스 또는 대상이 호스트인 경우 True입니다. 위의 호스트 식 중 어떤 것이든 다음과 같이 키워드, **IP**, **arp**, **rarp** 또는 **ip6**으로 앞에 붙일 수 있습니다:

**ip host** *host*

이는 다음과 같습니다.

**ether proto** *\ip* **and host** *host*

호스트가 IP 주소가 여러 개인 이름인 경우 각 주소가 일치하는지 확인합니다.

**ether dst** ehost

이더넷 대상 주소가 ehost인 경우 True입니다. Ehost는 /etc/ethers의 이름이거나 "xx:xx:xx:xx:xx" 또는 "xx.xx.xx" 형식의 숫자 MAC 주소일 수 있습니다.

**ether src** ehost

이더넷 소스 주소가 ehost인 경우 True입니다.

**ether src** ehost

이더넷 소스 또는 대상 주소가 ehost인 경우 True입니다.

**gateway** host

패킷이 호스트를 게이트웨이로 사용하는 경우 True입니다. 즉, 이더넷 소스 또는 대상 주소가 호스트였지만 IP 소스와 IP 대상 모두 호스트되지 않았습니다. 호스트는 이름이어야 하며 시스템의 호스트 이름 대 IP 주소 확인 메커니즘(호스트 이름 파일, DNS, NIS 등)과 시스템의 호스트 이름 대 이더넷 주소 확인 메커니즘(/etc/ethers, etc)에 의해 검색되어야 합니다. (동등한 표현은

**ether host** *ehost* **and not host** *host*

호스트/ehost의 이름 또는 숫자와 함께 사용할 수 있습니다.) 현재 IPv6 사용 구성에서 이 구문이 작동하지 않습니다.

**dst net** *net*

패킷의 IPv4/v6 대상 주소에 네트워크 번호가 있는 경우 True입니다. Net은 네트워크 데이터베이스(/etc/networks 등)의 이름이나 네트워크 번호일 수 있습니다. IPv4 네트워크 번호는 점 쿼드(예: 192.168.1.0), 점 3배(예: 192.15.268.1), 점 쌍(예: 172.16) 또는 단일 번호를 의미합니다. 번호 IPv6 네트워크 번호는 완전히 작성되어야 합니다. 넷마스크는 ff:ff:ff:ff:ff:ff:ff:ff:ff:ff. 따라서 IPv6 "네트워크" 일치는 실제로 호스트 일치하고 네트워크 길이 일치합니다.

**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 (\). 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** *p*

where *p* 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, 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, and is not supported by filter engines in the kernel, 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 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 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 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.

**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](http://www.tcpdump.org/manpages/pcap.3pcap.html)**(3PCAP) currently know how to parse these protocols.

**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*.

**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** Fitype

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

**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*

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 Distribution System) frames.

**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***[session\_id]*

True if the packet is a PPP-over-Ethernet Session packet (Ethernet type 0x8864). If *[session\_id]* is specified, only true if the packet has the specified *session\_id*. 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 0x27 && ip**

filters IPv4 protocols encapsulated in PPPoE session id 0x27.

**geneve***[vni]*

True if the packet is a Geneve packet (UDP port 6081). If *[vni]* is specified, only true if the packet has the specified *vni*. Note that when the **geneve** keyword is encountered in *expression*, it changes the decoding offsets for the remainder of *expression* on the assumption that the packet is a Geneve packet.

For example:

**geneve 0xb && ip**

filters IPv4 protocols encapsulated in Geneve with VNI 0xb. This will match both IP directly encapsulated in Geneve as well as IP 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** *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.

**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.

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, 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), **icmp6type (ICMP v6 type field) icmpcode** (ICMP code field), **icmp6code** (ICMP v6 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 ICMPv6 type fields are available: **icmp6-echo**, **icmp6-echoreply**, **icmp6-multicastlistenerquery**, **icmp6-multicastlistenerreportv1**, **icmp6-multicastlistenerdone**,**icmp6-routersolicit**, **icmp6-routeradvert**, **icmp6-neighborsolicit**, **icmp6-neighboradvert**, **icmp6-redirect**, **icmp6-routerrenum**, **icmp6-nodeinformationquery**, **icmp6-nodeinformationresponse**, **icmp6-ineighbordiscoverysolicit**, **icmp6-ineighbordiscoveryadvert**, **icmp6-multicastlistenerreportv2**, **icmp6-homeagentdiscoveryrequest**, **icmp6-homeagentdiscoveryreply**, **icmp6-mobileprefixsolicit**, **icmp6-mobileprefixadvert**, **icmp6-certpathsolicit**, **icmp6-certpathadvert**, **icmp6-multicastrouteradvert**, **icmp6-multicastroutersolicit**, **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 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 )**