WinPcap BPF 过滤规则

[WinPcap 用户手册]

注释：此文档是在 TcpDump（ps：知名嗅探工具）上的文档基础上改写而成，源文档可以在 www.tcpdump.org 上找到（ps：http://www.tcpdump.org/tcpdump_man.html）

Wpcap 过滤器是基于描述性而制定的语法规则，过滤规则是一个包含过滤表达式的 ASCII 码字符串，此规则表达式在程序中作为 pcap_compile()函数的参数被编译，并用于内核级的包过滤机制使用.（ps：表达式的形成后面会讲到，个人感觉表达式应该是由各种过滤关键字形成的）

此表达式决定哪种包将被捕获（ps：dumped？感觉逆着翻译更好些），如果表达式为空，所有网卡捕获到的数据包都会提交给内核过滤引擎，否则，只有满足表达式中关键字的数据包才会

表达式是由一个或者多个原语（ps：原语也太专业了，个人感觉：翻译成关键字或者基本类型）组成，而关键字通常是被一个或多个修饰词修饰的标识符（名称或序号）（ps：id 貌似是数据包头里的各种类型，参考 ip_header 数据结构即可明白），下面是三种不同类型的修饰词：

type

此类修饰词决定哪种类型的标识符被引用，常用的修饰词有 **host**, **net** and **port**. 例如, `host foo', `net 128.3', `port 20'.在没有此类修饰符的情况下，默认使用 **host**.(ps：foo 这个词很有意思，参加 RFC3092..嘿嘿)

dir

此修饰符类型从标识符指定了一个详尽的数据传输方向(使用 and 和 or)，常用的方向指示修饰符有 **src**, **dst**, **src or dst** and **src and dst**.等.例如， `src foo', `dst net 128.3', `src or dst port ftp-data'.在没有指定此类修饰符时，默认使用 **src or dst**.如果没有链路层（像 slip 这种点对点类型协议）可以使用 inbound 和 outbound 来具体描述一方向.

proto

此类修饰符的作用是过滤指定协议数据包，常见的协议有：**ether,fddi,tr,ip,ip6,arp,rarp,decent,tcp** and **udp,**例如， `ether src foo', `arp net 128.3', `tcp port 21'.如果没有指定协议类型修饰符，默认为使用所有与标识符匹配的协议. src foo' 等价于 `(ip or arp or rarp) src foo' (除非 src 不合法), `net bar'等价于 `(ip or arp or rarp) net bar' ， `port 53' 等价于 `(tcp or udp) port 53'.

['fddi' 实际上只是`ether'的一个别名，解析器都把它们理解为“数据链路层运行在详尽的网路接口上” FDDI 数据头包含了源和目的以太网地址，而且很可能包含以太网数据类型（ps：看下 ether_header 数据结构就晓得了），所以 FDDI 和 Ethernet 数据结构是可以相互转换的，FDDI 头还包含其它信息，可惜现有的过滤表达式无法清楚的描述出它们.

同上, `tr' 也是`ether'的别名，上段关于 FDDI 头信息的声明同样适用于令牌网数据头.]

除了以上内容，还有一些特殊类型关键字不符合以上修饰：饰
gateway, **broadcast**, **less**, **greater** 和算术表达式，下面来详细讨论：

我们使用 **and**, **or** 和 **not** 连接这些基本类型来实现强大的混合多功能表达式，例如，`host foo and not port ftp and not port ftp-data'.为减少输入，同一修饰符可以省略.例如，`tcp dst port ftp or ftp-data or domain' 完全等价于 `tcp dst port ftp or tcp dst port ftp-data or tcp dst port domain'.

可允许的基本类型（原语）有：

dst host *host*
当 IPv4/v6 数据包的 destination 域是一个 *host* 类型（地址、名称都可）时成立.(ps:destination 域其实就是数据报头的结构体里的一个变量)

src host *host*
当 IPv4/v6 数据包的 source 域是一个 *host* 类型（地址、名称都可）时成立.

host *host*
当 IPv4/v6 数据包的 source 域或者 destination 域是一个 *host* 类型（地址、名称都可）时成立.以上三种关于 **host** 的表达式都可以被 **ip**, **arp**, **rarp**, 或者 **ip6** 这些前缀关键字修饰，例如：
`ip host` *host*
等价于
`ether proto \`*ip* `and host` *host*
如果 **host** 是一个包含多 IP 的名称，每一个 IP 地址都会做一次匹配检查.

ether dst *ehost*
如果目标以太网地址是 ehost 类型是成立，Ehost 可以是一个从/etc/ethers 得到的名称，或者是一个数字（标识）（参见 ethers（3N）数字格式化）（ps：貌似是类似 dword 类型的 IP 通过格式化就得到正常 IP 类型那个意思）

ether src *ehost*
当源以太网卡地址是 ehost 类型时成立.

ether host *ehost*
当目标以太网卡地址是 ehost 类型时成立.

gateway *host*
当数据包的 host 域为网关地址时成立,例如,以太网源地址或者目标地址是 host 类型，但是源 IP 和目标 IP 却不是（不同时是）host 类型（ps：我汗，初中学过的 neither nor 的翻译，我记不清了！悔不该当初啊..），Host 必须是一个名称，而且必须在 host-IP 映射关系途径（host 镜像文件，DNS,NIS，等）和 Host-Ethernet 关系途径（/etc/ethers 等）定义过，（一个等价表达式：
`ether host` *ehost* `and not host` *host*
其中的 host 或者 ehost 可以是名称和标识号）目前这种语法在 IPv6 安装的情况下不可用.

dst net *net*

当数据包的 IPv4/6 目标地址是一个固定静态 IP 时成立，Net 既可以是一个来自 /etc/networks 的名称，也可以是一个网络地址号（详见 network（4））;

src net *net*

当数据包的 IPv4/6 源地址是一个固定静态 IP 时成立.

net *net*

当数据包的 IPv4/6 源地址或者目标地址是一个固定静态 IP 时成立.

(ps:net 是嘛玩意，清楚的告诉我..我就先当固定静态 IP 翻译了..)

net *net* **mask** *netmask*

当 IP 地址经过指定的网络掩码处理后与 Net 匹配时成立，可以用 src 和 dst 修饰词进行限制，需要注意的是：此语法在 IPv6 网络环境下不可用.

net *net*/*len*

当 IPv4/6 地址经过网络掩码长度字节宽度（瞎翻的，据 Ferris Xu 的翻译是：掩码那个 DWORD 里的连续为 1 的长度）与 net 匹配处理匹配时成立，可以使用 src 或者 dst 修饰词修饰。

dst port *port*

当数据包是 tcp 或者 udp 类型，并且目标端口相匹配时成立，这个端口可以是一个数字也可以是一个/etc/servces 里定义协议服务名，（详见 tcp（4p） 和 udp（4p）），如果使用的是名称，则端口的号码和协议都要进行匹配检查，如果是使用端口号或者是一个不明确的协议名称，则仅仅对数字进行匹配（例如，dst port 513 将得到 tcp/login 的数据流和 udp/who 的数据流，port domain 将得到 tcp/udp 的 domain 协议数据流）.

src port *port*

当数据包的源端口与指定端口匹配时成立.

port *port*

当数据包的源端口或者目的端口与指定端口匹配时成立.以上关于端口的表达式都可以被 **tcp** 和 **udp** 作为前缀修饰符修饰（ps：后面会提到 tcp 和 udp 即是修饰符又是关键字，我怎么觉着应该是修饰呢..），例如：

`tcp src port` *port*

此式只有在 tcp 协议包的源端口与指定的端口匹配时成立.

less *length*

当数据包的长度小于或等于指定长度时成立，如下式：

`len <=` *length*.

greater *length*

当数据包的长度大于或等于指定长度时成立，如下式：

`len >=` *length*.

ip proto *protocol*

当数据包是一个属于 IP 类型协议的数据包（详见 ip（4p））时成立（不被过滤掉），*Protocol* 可以是一个标识号，也可以是以下协议名称中的一种：

icmp, *icmp6*, *igmp*, *igrp*, *pim*, *ah*, *esp*, *vrrp*, *udp*, or *tcp*.

需要注意的是，tcp，udp，icmp 这些标识符同时也是关键字，所以必须当转义字符处理，类似 C 风格的\\，还有要注意的是:此原语不捕获协议头链

ip6 proto *protocol*

当数据包是一个属于IPv6类型的协议的数据包时成立.注意这个原语是不能捕获协议报头链的

ip6 protochain *protocol*

当数据包为 IPv6 类型，并且协议头链里包含协议类型域时成立，例如，

ip6 protochain 6

匹配所有协议头链里是 TCP 协议头的 IPv6 协议数据包，这个数据包可能包含例如：头确认，路由头，或者点对点头，Tcpdump 在处理 IPv6 报头和 TCP 报头时，由于原语是很复杂的，所以很难被 BPF 代码所有效的处理，最终导致了使用此规则表达式使 Tcpdump 可能出现性能问题..

ip protochain *protocol*

与上面的 **ip6 protochain** *protocol* 功能一致，但是作用在 IPv4 情况下.

ether broadcast

当数据包为以太网广播报文时成立，关键字 ether 是可选择的.

ip broadcast

当数据包是一个 IP 广播报文时成立.它检查所有的全 0 或者全 1 的广播模式，然后再寻找本地子网掩码.

ether multicast

当数据包为以太网多播报文时成立.其中关键字 ether 是可选择的，算是对`**ether[0] & 1 != 0**' 这种形式的简写和助记.

ip multicast

当数据包是 IP 多播报文时成立.

ip6 multicast

当数据包是一个 IPv6 广播报文时成立.

ether proto *protocol*

当数据包属于以太网协议类型时成立. *Protocol* 可以是一个数字也可是 *ip*, *ip6*, *arp*, *rarp*, *atalk*, *aarp*, *decnet*, *sca*, *lat*, *mopdl*, *moprc*, *iso*, *stp*, *ipx*, or*netbeui* 这些协议名称中的一种,注意这些标识符也属于关键字，所以必须通过转义字符(\)的处理.

[对于 FDDI(例如，`**fddi protocol arp**')和令牌网（例如，`**tr protocol arp**'），对于大多数此种协议，这种协议标识符来自比 FDDI 和令牌类型网络更高层的 802.2 LLC 层（逻辑链路层）

当过滤 FDDI 网络或者令牌网的协议的时候，Tcpdump 只检查用 0x000000 的 OUI 格式处理（被称为 SNAP 格式化）LLC 头的 protocol ID 域，而对于以太网，Tcpdump 则没有检查此域.

当然，存在一些特殊情况：

iso 协议检查 LLC 头的 DSAP(目标服务接入点)域和 SSAP（源服务接入点）域，stp 和 netbeui 协议则检查 LLC 头的 DSAP（目标服务接入点）域，还有 atalk 协议，它会检查用 0x080007 进行 OUI 处理的 SNAP 格式数据包（包括 Appletalk 类型）.

对于以太网类型，tcpdump 检查大部分协议的以太网类型域，以下协议存在特殊性：

iso，sap，netbeui 先作一个 802.3 框架协议的匹配，然后就和 FDDI 和令牌网下的情形一样检查 LLC 头信息，对于 atalk 协议，它将进行以太网下的 Appletalk 协议匹

配和 FDDI、令牌网下的 SNAP 格式数据包匹配双重检查，aarp 协议则是进行以太网类型和用 0x000000 进行 OUI 处理的 802.2 SNAP 类型的双重检查，还有 ipx 协议，它要进行很多地方的匹配检查：以太网类型的 IPX 类型匹配，LLC 头信息的 IPX 协议的 DSAP（目标服务接入点）匹配，802.3 下没有被 LLC 头信息封装的 IPX 检查，和 SNAP 格式的 IPX 类型检查.]

decnet src *host*

当 DECNET 源地址为 host 类型时成立（ps：DECNET 为一种协议类型），其中 host 可以是“10.123”这种格式的地址，或者是一个 DECNET host 名称.[DECNET host 名称只在已经配置并运行 DECNET 的 Ultrix 系统上可用.]

decnet dst *host*

当 DECNET 协议的目标地址是 host 类型时成立.

decnet host *host*

当 DECNET 的源地址或者目标地址是 host 类型时成立.

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

原型是：

ether proto *p*

其中 p 是上面 11 种协议中的任一种.

lat, **moprc**, **mopdl**

原型是：

ether proto *p*

其中 p 指代以上三种协议中的任一种，但是需要注意的是 tcpdump 目前并不能准确的区分这些协议类型.

vlan *[vlan_id]*

当数据包是一个 IEEE 802.1Q VLAN 类型的协议包时成立，如果指定了[vlan_id],则当数据包也包含此指定 vlan_id 时才成立，注意在假设这个包是 VLAN 类型的时候，第一个 vlan 关键字遇到不能解析的表达式位置时将改变编码偏移直接解析表达式的剩余部分.

tcp, **udp**, **icmp**

原型：

ip proto *p* or ip6 proto *p*

其中 p 是 tcp，udp，icmp 中的任一种.

iso proto *protocol*

当数据包是符合 OSI 数据包协议类型时成立.其中 Protocol 可以是一个数字或者是 clnp，esis，或者 isis 三种类型中的一种.

clnp, **esis**, **isis**

原型:

iso proto *p*

其中 p 是以上协议中的任一种，需要注意的是 tcpdump 还没有完全的搞定这些协议的解析工作.

expr relop expr

当表达式的关系运算正确时成立，其中 relop 是下面运算符之一：

>, <, >=, <=, =, !=

expr 是一个算术表达式（使用标 C 语法即可），可以使用正整数、常规的位运算符[+, -, *, /, &, |]、长度运算符（ps：len）、和特殊的数据位置（指包信息里的某个位），如果想得到数据包里的某个数据,可以使用如下语法：

proto [*expr* : *size*]

Proto 是 **ether, fddi, tr, ip, arp, rarp, tcp, udp, icmp**,**ip6** 中的任一种，用来明确这些索引操作所处的协议级别.注意 tcp，udp 和其它一些上层协议类型之提供了 IPv4 版本,不包含 IPv6(这个我们会在以后解决)(ps:需不需要我的帮忙？嘿嘿..)字节偏移，当确定了协议级别后，这个偏移就来自 expr.Size 属性是可以选择的，根据你的兴趣（用途）来确定.它的取值可以是 1，2，4，默认为 1.还有那个长度操作符，是使用 len 关键字定义的，用来提供数据包长度限制.

例如，`**ether[0] & 1 != 0**'此表达式可以获得所有的多播数据流，`**ip[0] & 0xf != 5**'则抓取所有可选择的 IP 包. `**ip[6:2] & 0x1fff = 0**'只抓取整个从源到目的的包，或者全 0 处理（参考 Ferris Xu 的翻译，应为：段偏移量是 0）的源到端（src2dst）的包.这个检查其实是默认的应用在了 tcp 和 udp 索引操作上，例如，tcp[0]一般代表 TCP 头的第一个字节，绝不会代表一个间接传输（非 src-dst）的碎片数据包的第一个字节.

一些偏移后得到的域数值可以用名称表示而不是仅仅只可以用数字.下面一些协议头的域偏移是可行的：

icmptype (ICMP 类型域),**icmpcode** (ICMP 编码域), and **tcpflags** (TCP 标志域).

下面是一些可用的 ICMP 类型域:

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**.

(ps:具体指的是那个偏移位置，以及如何使用..大家自己测试即可.反正我不清楚..)

下面是一些可用的 TCP 标志域:

tcp-fin, **tcp-syn**, **tcp-rst**, **tcp-push**, **tcp-push**, **tcp-ack**, **tcp-urg**.

(ps:这个大家相对熟悉些了吧，这些标志《TCP/IP 详解》里都有解释)

原语（基本类型）可以用按以下格式组合：

原语加上括号或者用使用运算符（圆括号对于表达式来说是敏感的，所以必须避免（转义））.

非(`**!**' or `**not**').
与 (`**&&**' or `**and**').
或 (`**||**' or `**or**').

非操作具有最高优先级，与操作和或操作则是相同级别且符合从左到右优先级关系，注意当一个明显的 and 使用时，不是用来合并，而是用来连接一系列的原语.

如果标识符前没有使用关键字，则假设其关键字为最近的那个(就近原则).例如,

not host vs and ace
原型:(谁是谁的缩写)
not host vs and host ace
千万表理解成下面这种：
not (host vs or ace)

(ps:上面这段根本就是 C 风格，学过 C 的都晓得撒)

关于过滤表达式的参数：可以以单独的参数或者是多个参数的方式传给 tcpdump，方便优先，一般的，如果表达式包含转义特征，按单一参数比较简单些，关于参数，多个参数在被解析前会先用空格给合并起来.

Translated by weety（MSN:0x0o@Live.Cn），仅作为 LanStat 项目研究所用.

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.
