https://www.wireshark.org/docs/wsug\_html\_chunked/ChWorkBuildDisplayFilterSection.html

https://wiki.wireshark.org/CaptureFilters#Capture\_filter\_is\_not\_a\_display\_filter

Filter-

http.request.uri == <https://www.wireshark.org/>

udp contains 81:60:03

The example above match packets that contains the 3-byte sequence 0x81, 0x60, 0x03 anywhere in the UDP header or payload.

http.host matches "acme\.(org|com|net)"

The example above match HTTP packets where the HOST header contains acme.org or acme.com or acme.net. Comparisons are case-insensitive. Note: Wireshark needs to be built with libpcre in order to be able to use the matches resp. ~ operator.

eth.src[0:3] == 00:00:83

The example above uses the n:m format to specify a single range. In this case n is the beginning offset and m is the length of the range being specified.

eth.src[1-2] == 00:83

The example above uses the n-m format to specify a single range. In this case n is the beginning offset and m is the ending offset.

eth.src[0:3,1-2,:4,4:,2] ==

00:00:83:00:83:00:00:83:00:20:20:83

Wireshark allows you to string together single ranges in a comma separated list to form compound ranges as shown above.

**Membership Operator.**

Wireshark allows you to test a field for membership in a set of values or fields. After the field name, use the in operator followed by the set items surrounded by braces {}.

tcp.port in {80 443 8080}

This can be considered a shortcut operator, as the previous expression could have been expressed as:

tcp.port == 80 || tcp.port == 443 || tcp.port == 8080

### A Common Mistake

Using the != operator on combined expressions like eth.addr, ip.addr, tcp.port, and udp.port will probably not work as expected.

Often people use a filter string to display something like ip.addr == 1.2.3.4 which will display all packets containing the IP address 1.2.3.4.

Then they use ip.addr != 1.2.3.4 to see all packets not containing the IP address 1.2.3.4 in it. Unfortunately, this does not do the expected.

Instead, that expression will even be true for packets where either source or destination IP address equals 1.2.3.4. The reason for this, is that the expression ip.addr != 1.2.3.4 must be read as “the packet contains a field named ip.addr with a value different from 1.2.3.4”. As an IP datagram contains both a source and a destination address, the expression will evaluate to true whenever at least one of the two addresses differs from 1.2.3.4.

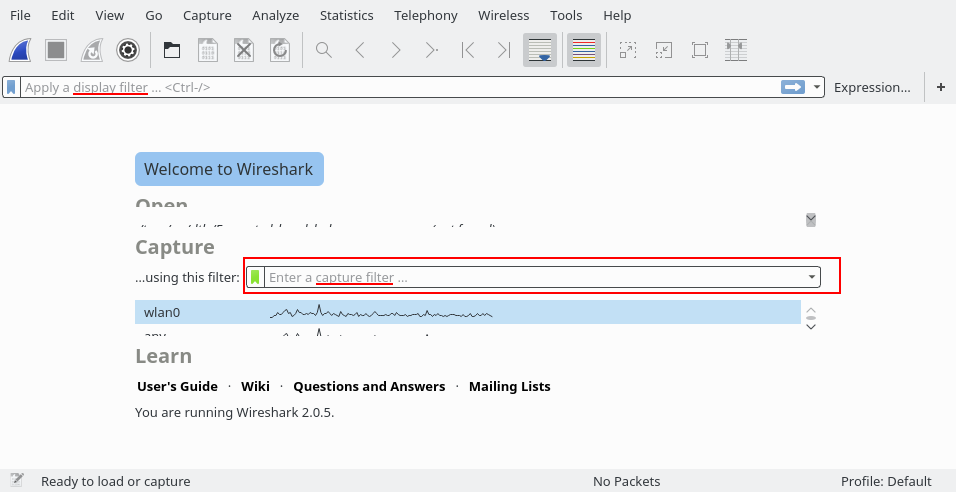
If you want to filter out all packets containing IP datagrams to or from IP address 1.2.3.4, then the correct filter is !(ip.addr == 1.2.3.4) as it reads “show me all the packets for which it is not true that a field named ip.addr exists with a value of 1.2.3.4”, or in other words, “filter out all packets for which there are no occurrences of a field named ip.addr with the value 1.2.3.4”.

## Capture filter is not a display filter

Capture filters (like tcp port 80) are not to be confused with display filters (like tcp.port == 80). The former are much more limited and are used to reduce the size of a raw packet capture. The latter are used to hide some packets from the packet list.

Capture filters are set before starting a packet capture and cannot be modified during the capture. Display filters on the other hand do not have this limitation and you can change them on the fly.

In the main window, one can find the capture filter just above the interfaces list and in the interfaces dialog. The display filter can be changed above the packet list as can be seen in this picture:



## Gotchas

Some filter fields match against multiple protocol fields. For example, "ip.addr" matches against both the [IP](https://wiki.wireshark.org/IP) source and destination addresses in the IP header. The same is true for "tcp.port", "udp.port", "eth.addr", and others. It's important to note that

* ip.addr == 10.43.54.65

is equivalent to

ip.src == 10.43.54.65 or ip.dst == 10.43.54.65

This can be counterintuitive in some cases. Suppose we want to filter out any traffic to or from 10.43.54.65. We might try the following:

* ip.addr != 10.43.54.65

which is equivalent to

ip.src != 10.43.54.65 or ip.dst != 10.43.54.65

This translates to "pass all traffic except for traffic with a source IPv4 address of 10.43.54.65 **and** a destination IPv4 address of 10.43.54.65", which isn't what we wanted.

Instead we need to negate the expression, like so:

* ! ( ip.addr == 10.43.54.65 )

which is equivalent to

! (ip.src == 10.43.54.65 or ip.dst == 10.43.54.65)

This translates to "pass any traffic except with a source IPv4 address of 10.43.54.65 **or** a destination IPv4 address of 10.43.54.65", which is what we wanted.

This can also happen if, for example, you have tunneled protocols, so that you might have two separate IPv4 or IPv6 layers and two separate IPv4 or IPv6 headers, or if you have multiple instances of a field for other reasons, such as multiple IPv6 "next header" fields.

If you have a filter expression of the form name op value, where name is the name of a field, op is a comparison operator such as == or != or <or..., and value is a value against which you're comparing, it should be thought of as meaning "match a packet if there is at least one instance of the field named name whose value is (equal to, not equal to, less than, ...) value". The negation of that is "match a packet if there are noinstances of the field named name whose value is (equal to, not equal to, less than, ...) value"; simply negating op, e.g. replacing == with != or <with >=, give you another "if there is at least one" check, which is not the negation of the original check.

1. ip.addr==…
2. ip.addr==192.168.43.44 and frame.len>400
3. tcp
4. icmp and frame.number

<https://serverfault.com/questions/220775/what-does-the-in-mean-in-a-zone-file>

<https://blog.dnsimple.com/2015/03/whats-in-a-dns-response/>