


| | |
|---|--|
|  <div> <div>Universidad</div> <div>Carlos III</div> <div>de Madrid</div> </div> | <div>Wireshark traffic capture software help guide</div> |
| Redes y Servicios de Comunicaciones | 2020-2021 |

1. USING WIRESHARK

For the use of the program, you can refer to the system manual pages or the online documentation (Wireshark Documentation), available at <http://www.wireshark.org/docs/>.

The `wireshark` capture tool is available for multiple operating systems and can be used to analyze previously saved captures. Alternatively, you may find it interesting to use the `cloudshark` tool (<http://www.cloudshark.org/>) to analyze saved captures without having `wireshark` installed on your PC.

1.1 Capturing traffic

To start capturing the traffic being transferred over the network, use the "Capture->Start" menu (or the corresponding shortcut button in the main window). But, before you start capturing, you must first specify the network interface(s) where you want to capture the traffic, which can be done in the main window when you start the program, or in the "Capture->Options" menu. Once a network interface is selected, all captures are done on that interface until it is changed.

When the traffic capture starts, in the main window you can see the packets being captured and a summary of packet information).

To stop capturing, click 'Stop' in the capture process information window (or "Capture->Stop").

You can save the capture to a file (for later analysis using `wireshark` or `cloudshark`).

1.2 Examining frames

The list of captured frames appears at the top of the main `wireshark` screen, indicating source, destination, protocol, etc. When you select a frame, you can see the frame detail at the bottom, including the data in each frame header. `Wireshark` offers two views of the frame content, one decoded and the other with the content in hexadecimal. In the decoded one, the information is divided by header, so it is easy to go to the specific header you want to analyze. For example, in a frame we can see Ethernet, IP, TCP and HTTP headers. By scrolling the information of a header you can see the different fields of that header. Clicking on any field highlights that field in the hexadecimal sequence that represents the frame in the other window. The fields in square brackets, which appear in the decoded frame, are not data included in the frame but additional information that `wireshark` has been able to deduce from the frame. For example, in an Echo request there is a [Response frame:] that indicates the packet number in the `wireshark` capture in which the corresponding Echo reply can be found (if it does not appear it means that `wireshark` has not been able to determine it or that the Echo reply frame has not been captured). And in the Echo reply there is a [Request frame:]

indicating the packet number in the wireshark capture in which to find the corresponding Echo request.

1.3 Filters

In wireshark there are two types of filters: capture and display, which also have a different syntax.

Capture filters in wireshark are nothing more than a text that follows the format of the 'tcpdump' command. In appendix I we describe slightly the features of this filter definition language. This type of filters are used to discriminate which types of packets are captured (and saved in the capture file) and which are not. These filters are normally used when traffic is going to be captured over a long period of time, and the size of what is going to be captured can become a problem so it is better to discriminate before capturing.

When making small captures, it is usually better to capture all the packets and set a display filter to see only those that interest you. This has the advantage of being able to change the filter and view other packets if we realize that our initial selection was not accurate (which we cannot do with capture filters because we have not captured those other packets). In principle during the practices we will use display filters. The display filter is written in a "Filter" window at the top of the Wireshark window. The syntax of display filters (which is different from capture filters) is described in the following link:

https://www.wireshark.org/docs/wsug_html_chunked/ChWorkDisplayFilterSection.html

It is important to read that page and the next two pages linked from it to understand how to work with display filters. Also in the wireshark program itself, in Help->Manual Pages, you can access information about wireshark filters. A more detailed reference of the display filters that we can do in wireshark is: <https://www.wireshark.org/docs/dfref/> (which is given as a reference, although it is not necessary to see it in detail for practical purposes).

Wireshark also provides a dialog box to help create display filters. Next to the display filter window there is an "Expression" button and clicking it brings up a dialog window (see Figure 1). The window is basically a list of protocols, each protocol can be extended to view fields within the protocol:

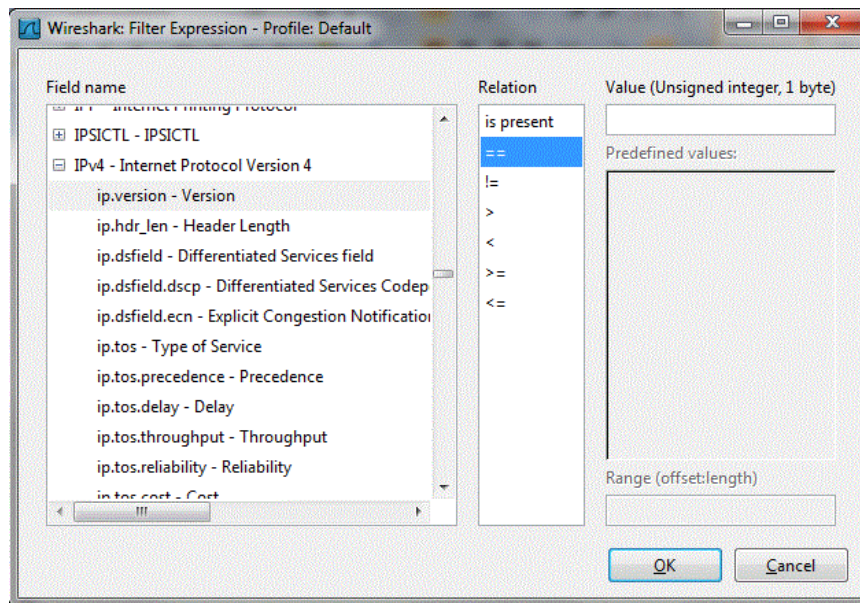


Figure 1: Window for creating display filters

You can type the first letter of a protocol name to move forward and find the protocol you are looking for faster. You can also use the search box to search by the full name of a protocol. Once in the searched protocol, you can check the value of a field against a certain value. When you click OK the expression will appear in the display filter window. Thus we can create any expression to include in a filter. Then we can combine it with other expressions using the operators described in the help pages linked above (and, or, not, etc.). Therefore, this help is very useful to learn how to create filters, even if later with practice we usually write the filter directly without the help of the "Filter Expression" window.

APPENDIX I: Capture filters in Wireshark

Capture filters follow the syntax of 'tcpdump' and can be done per protocol, per host, per port, etc. Filters can be combined using boolean operators ('and', 'or' and 'not'). To eliminate operator precedence problems, parentheses can be used.

An expression in a filter consists of one or more primitives. Primitives usually consist of an identifier (name) preceded by one or more qualifiers that parameterize the operation of the primitive.

The possible qualifiers are of one of the following types:

1- Type: identifies whether we are talking about a host address, a network address or a port number.

2- Dir: Specifies a transfer address from or to the identifier. Possible addresses are 'src' (source), 'dst' (destination), 'src or dst' (either source or destination), 'src and dest' (source and destination).

3- Proto: Restrict the capture to a particular protocol. Possible values are 'ether', 'fddi', 'ip', 'rarp', 'tcp', 'udp', etc.

The main valid primitives are:

1- dst host <host>: true if the destination IP address of the packet is the specified host, which can be a name or an IP address. E.g.: 'dst host it015' or 'dst host 163.117.244.212'.

2- src host <host>: analogous to the previous one but with the source IP address of the packet.

3- host <host>: true if the source or destination IP address of the packet is host.

4- ether dst <host>, ether src <host>, ether host <host>: analogous to the previous three but with ethernet addresses.

5- dst net <net>, src net <net>, net <net>: analogous to the first three primitives but with network addresses instead of host addresses.

6- dst port <port>, src port <port>, port <port>: analogous to the first three primitives but with port numbers instead of host addresses.

7- ip proto <protocol>: true if the packet is an ip packet with the higher level protocol protocol above it. Protocol can be icmp, igmp, udp or tcp. Since tcp, udp and icmp are

also identifiers, they must be escaped with '\'. For example, to keep the ip packets that use tcp above, we would write '(ip proto \tcp)'.

8- ether broadcast or ip broadcast: true if the packet is an ethernet broadcast or ip broadcast, respectively.

9- ether multicast or ip multicast: true if the packet is an ethernet multicast or ip multicast, respectively.

10- ether proto <protocol>: true if the packet is an ethernet packet with the higher level protocol protocol above it. Protocol can be ip, arp or rarp. Since ip, arp and rarp are also identifiers, they must be escaped with 'escaped'. For example, to keep ethernet packets that use ip above, we would write '(ether proto \ip)'.

11- ip, arp, rarp: abbreviations for 'ether proto <protocol>', where <protocol> is ip, arp or rarp.

12- tcp, udp, icmp: abbreviations for 'ip proto <protocol>', where <protocol> is tcp, udp or icmp.

If, for example, we want to keep only the tcp packets with source it012 and destination it015, we would set '(src host it012) and (dest host it015) and (ip proto \tcp)'.

Once a filter is activated, if we perform a capture, only the traffic that meets the filtering conditions will be captured. We can make it act on the traffic of a file already loaded by simply using the 'Reload' option of the 'File' menu.

On the other hand we have the display filters, **which do not influence which packets are captured**, but which packets are displayed in the main wireshark window. Their syntax is not identical to the one used for capture filters (you can refer to the Wireshark help: Wireshark Filter for more information). In general, capture filters are important when capturing traffic over long periods of time, so you want to avoid having to store a lot of unneeded data. But if you make a mistake with the filter, you may find that you don't have the traffic you need. Display filters are more practical when storage of the capture is not an issue (such as the simple time-sensitive captures of this practice) because we have all the traffic and can change the filter if we are not seeing what we expected.