# Table of Contents

About this guide ............................................................................................................................................ v  
Scope......................................................................................................................................................... v 
Audience ................................................................................................................................................... v  
Software version note ............................................................................................................................... v  
Related information .................................................................................................................................. v 
NetShark filter syntax ................................................................................................................................... 1  
  How the filters work ................................................................................................................................. 1  
  Filter syntax .............................................................................................................................................. 1  
Short list of useful filters............................................................................................................................... 5  
  Generic filters............................................................................................................................................ 5  
  Ethernet filters........................................................................................................................................... 6  
  IP filters..................................................................................................................................................... 8  
  TCP/UDP filters...................................................................................................................................... 11  
  HTTP filters............................................................................................................................................ 13  
  802.11 filters ........................................................................................................................................... 14  
  VoIP filters.............................................................................................................................................. 16  
Configuration files ...................................................................................................................................... 19  
  Port numbers file..................................................................................................................................... 19  
  Protocol groups file................................................................................................................................. 19  
Manufacturers file...................................................................................................................................... 20  
  Multiple address ranges ....................................................................................................................... 20  
  Multiple manufacturer names ............................................................................................................. 20  
  Contract manufacturers ....................................................................................................................... 21  
  Location of the file .............................................................................................................................. 21  
Modifying configuration files .................................................................................................................. 21  
  Location of the files on the local system ............................................................................................. 21  
  Configuration files on NetShark appliances ....................................................................................... 22  
Appendix: Migrating old filters .................................................................................................................. 23  
  arp:: ......................................................................................................................................................... 24  
  cws:: ........................................................................................................................................................ 25  
  dns:: ........................................................................................................................................................ 25  
  fix:: .......................................................................................................................................................... 26  
  generic:: .................................................................................................................................................. 28  
  http:: ........................................................................................................................................................ 31  
  icmp:: ...................................................................................................................................................... 33
Filters Guide

ieee80211:: .............................................................................................................................................. 34
ip:: ........................................................................................................................................................... 38
ipres:: ...................................................................................................................................................... 41
mac:: ........................................................................................................................................................ 42
multi_segment:: ....................................................................................................................................... 43
pseudo:: ................................................................................................................................................... 44
rios:: ........................................................................................................................................................ 50
rtp:: .......................................................................................................................................................... 51
sip:: .......................................................................................................................................................... 51
sql:: .......................................................................................................................................................... 52
tcp:: ......................................................................................................................................................... 54
tcp_state:: ................................................................................................................................................ 56
udp:: ........................................................................................................................................................ 59
voip:: ....................................................................................................................................................... 60
ws:: .......................................................................................................................................................... 63
About this guide

Scope
This guide explains the structure and syntax of Riverbed® SteelCentral™ NetShark filters (formerly called Cascade Pilot filters), which are used by Riverbed® SteelCentral™ Packet Analyzer (formerly Cascade Pilot). It describes several commonly used filters, and provides information on using all of the SteelCentral NetShark filters—hundreds of them.

Audience
This guide assumes that you are an experienced network administrator and have a basic familiarity with the SteelCentral Packet Analyzer.

Software version note
The NetShark filter syntax and many of the filter names changed when version 10.0 of the NetShark and Packet Analyzer was released. In this guide the information on filters and filter syntax describes the version 10.0-and-later filters. An appendix near the back of this guide provides information on updating version 9.x-and-earlier filters to version 10.0-and-later filters.

Related information
For more information on the use of NetShark filters in Packet Analyzer, refer to the “Filtering” section of the SteelCentral Packet Analyzer Reference Manual.
NetShark filter syntax

NetShark filters, as used in Packet Analyzer, allow you to filter out extraneous data from samples of network traffic and concentrate on the data of interest.

How the filters work

For example, if you had this view of network traffic in Packet Analyzer…

…and you wanted to look only at email traffic, you could apply the filter

```
generic.application = "Email"
```

to the data set. This would limit the view to email packets, and the result would be:

NetShark filtering works by evaluating each packet in the data set against the filter. If the result for a packet is true, the packet is retained for display in the filtered view; if it is false, the packet is discarded from the data points that make up the view. In the example above, discarding the non-email packets and rescaling the remaining data make the pattern of email activity stand out much more clearly.

(For details on how to apply filters in Packet Analyzer, see the "Filtering" chapter in the *SteelCentral Packet Analyzer Reference Manual*.)

Filter syntax

*Note: The information in this section describes the filter syntax for version 10.0 and later of NetShark and Packet Analyzer products. Version 9.6-and-earlier filters do not work on version 10.0-and-later NetShark and Packet Analyzer products. For information on converting version 9.6-and-earlier filters to version 10.0 syntax, refer to Appendix: Migrating old filters at the back of this guide.*

NetShark filters are composed of expressions containing one or more comparisons combined or modified with Boolean operators.

A *comparison* has a syntax of:

```
field OPERATOR "value"
```

where:

- **field** is case sensitive and indicates the attribute or characteristic of the data that is being compared. The Filters panel in Packet Analyzer shows all the available fields.

- **OPERATOR** can be:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>contains</td>
<td>contains</td>
</tr>
</tbody>
</table>

- **value** specifies the data value for the comparison. The value must be enclosed in quotation marks, and no space is allowed between the quotation mark and the value.
An **expression** may be a single **comparison** or may group multiple comparisons as:

\[
\text{comparison BOOLEAN comparison}
\]

where **BOOLEAN** can be:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td>&amp;</td>
<td>and</td>
</tr>
</tbody>
</table>

A comparison may be negated using the `!` operator, and comparisons can be grouped using parentheses.

**Example filters:**

```
generic.application = "Web"

(ip.src = "63.147.82.80") & (ip.dst = "192.168.77.10")

http.uri contains "google.com" & !(tcp.server_port = "80" | tcp.server_port = "8080")
```
Short list of useful filters

The next several sections list commonly used Packet Analyzer filters, grouped as follows:

- **Generic**
- **Ethernet**
- **IP**
- **TCP / UDP**
- **HTTP**
- **802.11**
- **VoIP**

A brief explanation and an example accompany each filter.

**Generic filters**

**generic.application**

This expression depends on a set of customizable parameters that associate a list of ports/protocols to a common name such as “Web” or “Email” for frequently used filters. This allows filters to be more flexible and compact. The associations are contained in the proto-groups configuration file, described in the "Protocol groups file" section below.

For instance, the proto-groups file associates the common name "Web" with TCP ports 80 (HTTP), 8080 (HTTP), and 443 (HTTPS). Thus, using the generic.application filter with a value of "Web" matches all packets communicating on any of those ports.

Note that you should use caution if you want to match specific numbered ports, as the common names such as “Web” or “Email” are likely to combine multiple port numbers to make a broad match.

*Example:*

```
generic.application = “Web”
```
Filters Guide

**Ethernet filters**

**mac.address**

(mac.src)

(mac.dst)

This expression allows you to filter on Ethernet host (MAC) addresses. Using `mac.address` selects the packets if either the source or the destination matches the value. Replace the field `mac.address` with the expressions in parentheses if you are interested only in packets coming from a specific source MAC address or going to a specific destination MAC address.

*Example:*

`mac.src = "00:1d:6a:b8:a6:3f"`

**mac.vendor**

(mac.src_vendor)

(mac.dst_vendor)

This expression allows you to filter on a vendor name, which can be useful if a vendor is associated with more than one MAC address range. Refer to the “Manufacturers file” section for more information.

Using `mac.vendor` selects the packets if either the source or the destination matches the value. Replace `mac.vendor` with one of the expressions in parentheses if you are interested only in packets coming from a specific source vendor or going to a specific destination vendor.

*Allowed values:* Allowed vendor names are stored in a file called `manuf` in the directory `\server\configuration` of the Packet Analyzer installation. Refer to the “Manufacturers file” section for more information.

*Example:*

`mac.src_vendor = "Cisco-Link"`
**mac.vendor_with_mac**

(mac.src_vendor_with_mac)  
(mac.dst_vendor_with_mac)  

This expression allows you to filter on a string defined as a vendor name, combined with the last 3 bytes of the MAC address. Using `mac.vendor_with_mac` selects the packets if either the source or the destination matches the value. Replace the field `mac.vendor_with_mac` with one of the expressions in parentheses if you are interested only in packets coming from a specific source MAC address or going to a specific destination MAC address.

*Allowed values:* Allowed vendor names are stored in a file called `manuf` in the directory `\server\configuration` of the Packet Analyzer installation. Refer to the “Manufacturers file” section for more information. The vendor name represents the first 3 bytes of the MAC address; you add the last 3 bytes, using the format “<vendor_name>_xx:xx:xx”, where xx represents a byte in a MAC address.

*Example:*

mac.src_vendor_with_mac = "Cisco-Link_0c:08:78"

**mac.protocol_type_name**

This expression allows you to filter on the specified protocol at the network layer.

*Allowed values:* Unknown, IP, IPv6, ARP, RARP, XEROX, DLOG, X.75, NBS, ECMA, Chaosnet, X.25, AARP, EAPS, IPX, SNMP, MPCP, PPP, GSMP, MPLS, MPLS, PPPoE, EAPOL, AoE, LWAPP, LLDP, WSMP

*Example:*

mac.protocol_type_name = "IP"

**mac.dst_delivery_type**

(mac.src_delivery_type)  

This filter selects the type of delivery used for the MAC layer transmission. Destination or source can be specified.

*Allowed values:* Broadcast, Multicast, Unicast

*Example:*

mac.dst_delivery_type = "Multicast"
Filters Guide

**mac.vlan_id**

This expression allows you to filter on the VLAN Identifier.

*Example:*

```plaintext
mac.vlan_id = "1"
```

**IP filters**

**ip.address**

(ip.src)

(ip.dst)

This expression allows you to filter on a host IP address or name. Replace `ip.address` with one of the expressions in parentheses if you are interested only in an IP source or destination address or name.

*Example:*

```plaintext
ip.src = "74.125.155.103"
```

**ip.dst_delivery_type**

This expression allows you to filter on IP "Unicast", "Broadcast", and "Multicast".

Allowed values: Broadcast, Multicast, Unicast

*Example:*

```plaintext
ip.dst_delivery_type = "Unicast"
```
**ip.protocol_name**

This expression allows you to filter on the specified protocol at the transport layer contained in the IP protocol.


*Example:*

(ip.protocol_name = "TCP") or (ip.protocol_name = "UDP")

**ip.c_net**

(ip:src_c_net)
(ip:dst_c_net)

This filter allows you to filter on traffic coming or going to an IP Class C source or destination subnet. The expressions in parentheses allow you to filter only source subnets or only destination subnets.

*Example:*

ip.c_net = "192.168.77.0"
Filters Guide

**ip.domain**
(ip.src_domain)
(ip.dst_domain)

This filter allows you to filter on traffic coming from or going to a selected Internet Domain. It is possible to specify only source or destination with using one of the expressions in parentheses.

*Example:*

ip.domain = "le100.net"

**ip.country**
(ip.src_country)
(ip.dst_country)

This filter selects the source and destination country based on a GeoIP lookup. Use one of the expressions in parentheses to select only source or destination.

*Example:*

ip.dst_country = "Russian Federation"

**ip.src_internal**
(ip.dst_internal)

This filter allows specifying the IP address of the source (destination) interface if the host is in the internal net. To get all the traffic coming from (or going to) an external host use the expression "Remote".

*Example:*

ip.src_internal = "Remote"

**ip.is_fragmented_str**

This expression allows selecting between "Fragmented" and "Not Fragmented" traffic.

*Example:*

ip.is_fragmented_str = "Fragmented"
ip.time_to_live

This filter specifies the maximum time (in seconds) that a datagram is allowed to survive.

Example:

ip.time_to_live = "53"

TCP/UDP filters

tcp.port_pair

This expression allows you to filter on a TCP port number.

Example:

tcp.port_pair = "80"

tcp.identification_port_name

This expression allows you to use strings such as "pop3s" instead of port numbers to filter on TCP ports.

Allowed values are contained in the port-numbers file, described above in the “Port numbers file” section.

Example:

tcp.identification_port_name = "pop3s"

tcp.flags

This filter allows filtering packets according to TCP flags.

*Allowed values:* SYN, FIN, RST, PSH, ACK, URG, No Flags or any combination, e.g. SYN-ACK, PSH-ACK

Example:

tcp.flags = "PSH-ACK"
Filters Guide

**tcp.error_type**

This filter allows selecting packets according to TCP errors.

*Allowed values:* Retransmissions, Timeouts, Out of Order, Lost Segments, Duplicate Acks, Zero Windows, Resets

*Example:*

tcp.error_type = "Resets"

**tcp.server_ip**

This filter allows selecting packets specifying the IP address of the hosts that receive TCP connections.

*Example:*

tcp.server_ip = "87.255.33.136"

**tcp.client_ip**

This filter allows selecting packets specifying the IP address of the hosts that start TCP connections.

*Example:*

tcp.client_ip = "192.168.77.115"

**udp.port_pair**

This expression allows you to filter on UDP port numbers.

*Example:*

udp.port_pair = "19543"
udp.identification_port_name

This expression allows you to use strings such as "DNS" instead of port numbers to filter on UDP ports.

Example:
udp.identification_port_name = "DNS"

HTTP filters

http.uri

This expression allows you to filter on all or part of the URI.

Example:
http.uri contains "1A8928AF6E4E4255BBECE04056B00DA038/TC2.pdb"

http.host

This expression allows you to filter on the Host name in the http header.

Example:
http.host contains "youtube"

http.path

This expression allows you to filter on the HTTP resource path and name.

Example:
http.path contains "/books?id=Vi05"

http.method

This expression allows you to filter on the HTTP request type.

Allowed values: GET, POST, HEAD, PUT, DELETE, TRACE, OPTIONS, CONNECT

Example:
http.method = "GET"
**http.content_type**

This expression allows you to filter on the HTTP content type.

*Allowed values:* Any of the http mime types. See [http://www.iana.org/assignments/media-types](http://www.iana.org/assignments/media-types).

*Example:*

http.content_type contains "image"

**http.status_code**


*Example:*

http.status_code = "200"

---

**802.11 filters**

**wlan_link.channel**

This expression allows you to filter on packets using 802.11 channel representation strings such as BG 001, BG 002 ...

*Allowed values:* `<BG | A | N | Nhigh | NLow> space <3 digits channel number>`

*Example:*

wlan_link.channel = "BG 002"

**wlan_link.channel_frequency**

This expression allows you to filter on packets using 802.11 channel frequency in MHz (2412, 2417 ...)

*Example:*

wlan_link.channel_frequency = "2447"
wlan.bssid.essid

This expression allows you to filter on packets using the Extended Service Set IDentifier (ESSID) string.

Example:
wlan.bssid.essid = "Riverbed_WIFI"

wlan.frame_control.src_type
(wlan.frame_control.dst_type)

This expression allows you to filter on source (destination) wireless nodes according to their function as access points (AP) or stations (STA).

Allowed values: AP, STA

Example:
wlan.frame_control.src_type = "AP"

wlan_link.channel_designator_per_station

This expression allows you to filter on the string of the channel type designator.

Allowed values: For PPI valid values are A, B, G, N; for Radiotap valid values are A, B, G.

Example:
wlan_link.channel_designator_per_station = "B"

wlan.frame_control.protection_type_ap

This filter allows you to select the type of encryption used based on the AP to which the client is associated.

Allowed values: Unknown, WEP, WPA [TKIP], WPA2 [CCMP], None

Example:
wlan.frame_control.protection_type_ap = "WPA [TKIP]"
**Filters Guide**

**wlan.frame_control.type_name**

This expression allows you to filter on the string of the frame type.

Allowed values: Management, Control, Data, Reserved

*Example:*

```
wlan.frame_control.type_name = "Data"
```

**wlan.frame_control.type_subtype_name**

This expression allows you to filter on the string of the frame type/subtype.

Allowed values: Association request, Association response, Reassociation request, Reassociation response, Probe request, Probe response, Beacon, ATIM, Disassociation, Authentication, Deauthentication, Action, Action No Ack, Control Wrapper, Block Ack Request (BlockAckReq), Block Ack (BlockAck), PS-Poll, RTS, CTS, ACK, CF-End, CF-End + CF-Ack, Data, Data + CF-Ack, Data + CF-Poll, Data + CF-Ack + CF-Poll, Null (no data), CF-Ack (no data), CF-Poll (no data), CF-Ack + CF-Poll (no data), QoS Data, QoS Data + CF-Ack, QoS Data + CF-Poll, QoS Data + CF-Ack + CF-Poll, QoS Null (no data), QoS CF-Poll (no data), QoS CF-Ack + CF-Poll (no data)

*Example:*

```
wlan.frame_control.type_subtype_name = "ACK"
```

**VoIP filters**

**voip.user_number**

*(voip.caller_number)*

*(voip.receiver_number)*

This expression allows filtering on the phone number of the caller or the receiver of the VoIP call. `voip.user_number` is used to filter if either the caller OR the receiver matches the specified phone number. Use one of the expressions in parentheses to select the caller or the receiver separately.

*Example:*

```
voip.user_number = "15023591801"
```
**voip.user_ip**

(voip.caller_ip)

(voip.receiver_ip)

This expression allows selecting caller or receiver IP address. Use one of the expressions in parentheses to select caller or receiver separately.

*Example:*

voip.caller_ip = "192.168.77.27"

**voip.call_id**

This expression can be used to filter the Call-ID of a call.

*Example:*

voip.call_id = "7603a6824759d0f8366970ae6ba3c4c9@192.168.77.27"

**voip.final_status**

This expression can be used to filter on the state of a terminated call.

*Allowed values:* Canceled, Rejected, Completed, TimeOut

*Example:*

voip.final_status = "Completed"

**voip.protocol**

This expression can be used to filter on the protocol used during the call (SIP or H.323).

*Allowed values:* SIP, H.323

*Example:*

voip.protocol = "SIP"
Configuration files

There are a few configuration files that the Packet Analyzer and NetShark software may use when applying filters:

- port numbers file—associates a TCP or UDP port number with a well known protocol name
- protocol groups file—groups related protocols together
- manufacturers file—associates the first half of a MAC address with a device's manufacturer

Port numbers file

The port numbers file associates TCP/UDP ports with well-known protocol names. This lets you create more meaningful expressions in Packet Analyzer filters:

*Typical entry in port numbers file:*

```plaintext
ftp-data  20/tcp    File Transfer [Default Data]
ftp-data  20/udp    File Transfer [Default Data]
```

*Typical filter:*

```plaintext
tcp.identification_port_name = "ftp-data"
```

Protocol groups file

The protocol groups file groups together different ports/protocols and associates them with a single value, allowing you to use a simple expression to filter more than one item at a time. For example, the file defines the Email group as:

```plaintext
# Email
Email   25/tcp    SMTP
Email   465/tcp   Secure SMTP
Email   587/tcp   SMTP
Email   110/tcp   POP3
Email   995/tcp   POP3 over SSL
Email   143/tcp   IMAP
Email   585/tcp   Secure IMAP
Email   993/tcp   IMAP over SSL
Email   119/tcp   NNTP
```

This definition allows you to filter all of the above protocols with a single string. For example:

```plaintext
generic.application = “Email”
```

finds all packets of any of the Email protocols.
Filters Guide

Manufacturers file

This file lists ranges of MAC addresses and tells what manufacturer is associated with each range. A typical section of the file looks like this:

<table>
<thead>
<tr>
<th>Manufacturer's code (first three octets of MAC address)</th>
<th>Manufacturer name</th>
<th>Expanded manufacturer name (comment field)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:00:07 Xerox</td>
<td># XEROX CORPORATION</td>
<td></td>
</tr>
<tr>
<td>00:00:08 Xerox</td>
<td># XEROX CORPORATION</td>
<td></td>
</tr>
<tr>
<td>00:00:09 Xerox</td>
<td># XEROX CORPORATION</td>
<td></td>
</tr>
<tr>
<td>00:00:0A OmronTatei</td>
<td># OMRON TATEI ELECTRONICS CO.</td>
<td></td>
</tr>
<tr>
<td>00:00:0B Matrix</td>
<td># MATRIX CORPORATION</td>
<td></td>
</tr>
<tr>
<td>00:00:0C Cisco</td>
<td># CISCO SYSTEMS, INC.</td>
<td></td>
</tr>
<tr>
<td>00:00:0D Fibronics</td>
<td># FIBRONICS LTD.</td>
<td></td>
</tr>
<tr>
<td>00:00:0E Fujitsu</td>
<td># FUJITSU LIMITED</td>
<td></td>
</tr>
<tr>
<td>00:00:10 Hughes</td>
<td># NEXT, INC.</td>
<td></td>
</tr>
<tr>
<td>00:00:11 Tektronix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00:00:12 Information</td>
<td># INFORMATION TECHNOLOGY LIMITED</td>
<td></td>
</tr>
<tr>
<td>00:00:13 Canex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00:00:14 Metronix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00:00:15 Datapoint</td>
<td># DATAPoint CORPORATION</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This lets you make a more readable filter by using the manufacturer name instead of the first three MAC octets.

Example:

Since the entry for Riverbed Technology in the manufacturers file is:

00:0E:B6 RiverbedTechnology # Riverbed Technology, Inc.

you can use the filter:

```
mac.vendor = "RiverbedTechnology"
```

to find all packets coming from or going to equipment manufactured by Riverbed Technology, Inc. (that is, packets with 00:0E:B6 as the first 3 octets of the MAC address).

The manufacturer name must be spelled exactly the same as it appears in the file. For instance, if the filter in the example above used a value of “Riverbed” it would not find any packets.

Multiple address ranges

If the same manufacturer name is associated with multiple MAC address ranges, the filters find packets with addresses in all those ranges. For example, a manufacturer name of Xerox matches manufacturer codes of 00:00:07, 00:00:08, and 00:00:09 (as indicated in the listing above), and several others.

Multiple manufacturer names

Filters use the information exactly as it appears in the file. For example, the listings in the file for equipment made by Intel Corporation may carry a manufacturer name of either Intel or IntelCorpo. A filter using “Intel” as the manufacturer name would find only packets with MAC addresses associated with Intel; it would not find packets with MAC addresses associated with IntelCorpo. To find all packets associated with equipment manufactured by Intel you would have to use two filters, one with each name. (Alternatively, you could make a single filter that combined both filters with a Boolean OR.)
Contract manufacturers

Filtering with a network equipment vendor’s name may not show all traffic to or from equipment that carries that vendor’s brand. When a vendor has its equipment manufactured by a contract manufacturer, the equipment may have a MAC address associated with the contract manufacturer, not the vendor.

Location of the file

The manufacturers file is the Wireshark manuf file, which is maintained by the Wireshark organization. It is stored on your local system as:

C:\Users\<user>\AppData\Roaming\Riverbed\SteelCentral Packet Analyzer\<version>\server\configuration\manuf

where <user> is the name of the system user who installed Packet Analyzer and <version> is the version number of the software.

Modifying configuration files

You can modify the port numbers and protocol groups files described above to suit your purposes. Generally you will not need to modify the port numbers file, as it is based on standard assignments of port numbers used throughout the networking industry.

If you modify the protocol groups file, note that any port/protocol can belong to only one group. If a port/protocol is assigned to more than one group, only the first assignment in the file will be valid.

Location of the files on the local system

If you wish to modify the configuration files, use the port numbers and protocol groups files that are stored on your local system (the system that runs Packet Analyzer) as:

- port-numbers
- proto-groups

in the folder:

C:\Users\<user>\AppData\Roaming\Riverbed\SteelCentral Packet Analyzer\<version>\server\configuration

where <user> is the name of the system user who installed Packet Analyzer and <version> is the version number of the software. You can find the correct version number by opening Packet Analyzer and clicking the green “i” button in the upper right corner of the screen; the version number will be listed as the “internal build” on the About dialog.

Note that copies of the port-numbers and proto-groups files were stored under the C:\Program Files (x86) file structure during installation. Those files are used to restore the defaults; do not modify them.
Filters Guide

**Configuration files on NetShark appliances**

The configuration files on the local system, described above, affect only the filtering that is performed on the local system—that is, on:

- devices that are in or connected directly to the local system
- files that are stored on the local system or on networked drives mapped to the local system

The configuration files on the local system do not affect filtering on remote NetShark or NetShark virtual edition products—that is, on:

- capture jobs running on those appliances
- capture files stored on those appliances

Remote appliances run their own copies of the Packet Analyzer server software, and use their own copies of the configuration files when performing filtering operations.

**Important:** If you modify the configuration files, make sure that you modify the files on each of the remote appliances as well as the files on your local system. On the appliances, the configuration files are located as follows:

- port numbers file—In the appliance’s web interface, go to **Settings > Port/Protocol Names** (**Port Definitions** in version 10.5 and greater).
- protocol groups file—In the appliance’s web interface, go to **Settings > Port/Protocol Groups** (**Port Group Definitions** in version 10.5 and greater).
- manufacturers file—Not modifiable on the system.
Appendix: Migrating old filters

The 10.0 release of the Packet Analyzer and NetShark software introduced changes to the filters:

- The filter syntax changed (the extractor was incorporated into the field):

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Old:</strong> extractor::id OPERATOR &quot;value&quot;</td>
<td>ip::ip.str = &quot;10.5.3.17&quot;</td>
</tr>
<tr>
<td><strong>New:</strong> field OPERATOR &quot;value&quot;</td>
<td>ip.address = &quot;10.5.3.17&quot;</td>
</tr>
</tbody>
</table>

- Many of the filter names changed.
- The name of the filtering scheme changed from "Cascade Pilot filters" to "NetShark filters".

As a result, old (9.x-and-earlier) filters do not work with new (10.0-and-later) software, and vice versa.

If you have custom filters that you want migrate from the old form to the new form, the information in the sections below will help. These sections are arranged alphabetically by extractor name, and within each section the field IDs are alphabetized. Each entry shows:

```
old_filter -> new_filter
description of filter
```

(If the value for new_filter is "N/A", there is no new equivalent for the old filter.)

Substitute the new filter for the old filter, and the filter is ready to use with 10.0-and-later versions of Packet Analyzer software and NetShark appliances.

Click one of these links to go to the section corresponding to your filter's extractor:

- [arp::](#)
- [cws::](#)
- [dns::](#)
- [fix::](#)
- [generic::](#)
- [http::](#)
- [icmp::](#)
- [ieee80211::](#)
- [ip::](#)
Filters Guide

- ipres:
- mac:
- multi-segment:
- pseudo:
- rios:
- rtp:
- sip:
- sql:
- tcp:
- tcp_state:
- udp:
- voip:
- ws:

arp::

arp::bits -> arp.bits
    Bit count of ARP packets

arp::bytes -> arp.bytes
    Byte count of ARP packets

arp::class.str -> arp.class_name
    ARP packet class (ARP, RARP, or INARP)

arp::destination_hardware_address -> arp.src_hw_address.delivery_type
    Delivery type used for the hardware layer transmission

arp::destination_hardware_address.delivery_type.str ->
arp.dst_hw_address.delivery_type
    Delivery type used for the hardware layer transmission

arp::destination_hardware_address.vendor.str ->
arp.dst_hw_address
    Hardware address of the receiving host

arp::destination_hardware_address.vendor_with_mac.str ->
arp.dst_hw_address.vendor
    Hardware address vendor of the receiving host

arp::gratuitous_arp -> arp.is_gratuitous
    Indicates if a particular ARP request is gratuitous, meaning the source and destination protocol addresses are the same.

arp::gratuitous_arp.str -> arp.is_gratuitous_str
    Indicates if a particular ARP request is gratuitous, meaning the source and destination protocol addresses are the same.

arp::hardware_address_len -> arp.hardware_address_len
    Length of the hardware address
Appendix: Migrating old filters

**arp::**

- **hardware_type** -> **arp.hardware_type**
  Hardware type code (e.g. 1)

- **hardware_type.str** -> **arp.hardware_type_name**
  Hardware type string (e.g. Ethernet)

- **packets** -> **arp.packets**
  Packet count of ARP packets

- **protocol_address_len** -> **arp.protocol_address_len**
  Length of the protocol address

- **protocol_type** -> **arp.protocol_type**
  Protocol type code (e.g. 0x0800)

- **protocol_type.str** -> **arp.protocol_type_name**
  Protocol type string (e.g. IP)

- **source_hardware_address** -> **arp.src_hw_address**
  Hardware address of the transmitting host

- **source_hardware_address.delivery_type.str** -> **arp.dst_hw_address.vendor_with_mac**
  Hardware address and vendor of the receiving host

- **source_hardware_address.vendor** -> **arp.src_hw_address.vendor**
  Hardware address vendor of the transmitting host

- **source_hardware_address.vendor_with_mac** -> **arp.src_hw_address.vendor_with_mac**
  Hardware address and vendor of the transmitting host

- **type** -> **arp.type**
  ARP packet type code

- **type.str** -> **arp.type_name**
  ARP packet type string

**CWS::**

- **cws::dhcp.relayed.str** -> **dhcp.is_relayed_str**
  Description of the DHCP traffic (Relayed or Not Relayed)

**dns::**

- **is_authenticated_data** -> **dns.is_authenticated_data**
  Indication of whether the data in the response has been verified or otherwise meets the local security policy of the issuing server

- **is_authoritative** -> **dns.is_authoritative**
  Indication of whether the sending server is an authority for the domain name requested

- **is_query** -> **dns.is_query**
  Indication of whether the packet is a query

- **is_recursion_available** -> **dns.is_recursion_available**
  Indication of whether the sending server supports recursive queries
**Filters Guide**

**dns::is_recursion_desired** -> **dns.is_recursion_requested**
Indicates the sending client supports recursion and desires it.

**dns::is_response** -> **dns.is_response**
Indication of whether the packet is a response

**dns::is_truncated** -> **dns.is_truncated**
Indication of whether only the first 512 bytes of the response was returned

**dns::number_of_additional_rrs** -> **dns.response.additional_rrs**
The number of additional Resource Records (RRs) present in a packet.

**dns::number_of_answer_rrs** -> **dns.response.answer_rrs**
The number of answer Resource Records (RRs) present in a packet.

**dns::number_of_authority_rrs** -> **dns.response.authority_rrs**
The number of authority Resource Records (RRs) present in a packet.

**dns::number_of_queries** -> **dns.query.count**
The number of queries present in a packet.

**dns::opcode** -> **dns.opcode**
Type of DNS packet

**dns::opcode.str** -> **dns.opcode_name**
Description of DNS packet type

**dns::response_time** -> **dns.response_time**
Time that elapsed from when the request was issued to when the response is received

**dns::response_time.category.str** -> **dns.response_time_range**
Time range that elapsed from when the request was issued to when the response is received

**dns::return_code** -> **dns.status_code**
The return code for the DNS Query/Response.

**dns::return_code.str** -> **dns.status_code_name**
The return code string for the DNS Query/Response.

**dns::return_code.tfs** -> **dns.is_success**
Indication of whether the return code for the DNS Query/Response is success

**dns::return_code.tfs.str** -> **dns.is_success_str**
Description of the return code for the DNS Query/Response (Success or Failure)

**dns::transaction_id** -> **dns.transaction_id**
The session identifier for this packet

---

**fix::**

**fix::alloc_account** -> **fix.alloc_account**
Sub-account mnemonic

**fix::alloc_status** -> **fix.alloc_status**
Identifies status of allocation (e.g. '3')

**fix::alloc_status.name** -> **fix.alloc_status_name**
Identifies status of allocation (e.g. 'received')
fix::alloc_type -> fix.alloc_type
   Allocation type or purpose of an allocation message (e.g. '9')

fix::alloc_type.name -> fix.alloc_type_name
   Allocation type or purpose of an allocation message (e.g. 'Accept')

fix::bw -> fix.message_bits
   Number of bits used by the current message

fix::bw.bytes -> fix.message_bytes
   Number of bytes used by the current message

fix::cl_ord_id -> fix.cl_ord_id
   Order ID generated locally by the client, not necessarily unique in a multi-client environment

fix::error.count -> fix.type_errors
   Number of FIX errors of a given type.

fix::error.count.total -> fix.errors
   Number of FIX errors

fix::error.type.str -> fix.error_type
   Type of the FIX error as a string

fix::exec_type -> fix.exec_type
   Describes the specific Execution Report (e.g. '4 - Canceled')

fix::is_fix -> fix.is_fix
   Indication of whether the current packet contains FIX traffic

fix::message.category -> fix.message_category
   FIX Message type (e.g. ProgramTrading)

fix::message.count -> fix.messages
   FIX Message count

fix::message.type -> fix.message_type
   FIX Message type (e.g. NewOrderList)

fix::network.rtt -> fix.round_trip_time
   Client to server round trip time

fix::normalized.srt -> fix.service_response_time
   Delay between the order placement and execution report

fix::ord_id -> fix.order_id
   Order ID

fix::ord_id.grid -> fix.order_id
   Order ID (Used for grid, set also empty value)

fix::ord_qty -> fix.order_qty
   Order quantity

fix::ord_status -> fix.ord_status
   Order status code (e.g. 0)

fix::ord_status.name -> fix.ord_status_name
   Order status returned only for the response (e.g. New)

fix::ord_type -> fix.ord_type
   Order type code (e.g. '2 - Limit')
Filters Guide

**fix::packets** -> **fix.packets**
Number of packets carrying FIX traffic

**fix::price** -> **fix.price**
Price per unit of quantity (e.g. per share)

**fix::sender.compid** -> **fix.sender_compid**
Sender Firm ID

**fix::sender.locationid** -> **fix.sender_locationid**
Sender Location ID

**fix::sender.subid** -> **fix.sender_subid**
Specific Sender ID

**fix::seqnum** -> **fix.seqnum**
Message sequence number

**fix::seqnum_error_count** -> **fix.seqnum_errors**
Amount of messages with a sequence number that is lower than expected

**fix::seqnum_gap_count** -> **fix.seqnum_gaps**
Amount of messages with a sequence number that is higher than expected

**fix::seqnum_reset_count** -> **fix.seqnum_resets**
Amount of sequence reset messages

**fix::side** -> **fix.side**
Side code of order (e.g. '2 - Sell')

**fix::symbol** -> **fix.symbol**
Common, 'human understood' representation of the security

**fix::target.compid** -> **fix.target_compid**
Target Firm ID

**fix::target.locationid** -> **fix.target_locationid**
Target Location ID

**fix::target.subid** -> **fix.target_subid**
Target Specific ID

**fix::transaction.time** -> **fix.transaction_time**
SRT or Network RTT, depending on the value of **fix.transaction.time.type**

**fix::transaction.time.type** -> **fix.transaction_time_category**
String that distinguishes between the Network RTT and the SRT

**fix::unfilled_order.count** -> **fix.unfulfilled_orders**
Number of unfulfilled orders.

**generic::**

**generic::absolute_pktnum** -> **generic.absolute_pktnum**
Absolute packet number including those dropped by filters (but not including those dropped by BPF filters)

**generic::absolute_pktnum.every_10** -> N/A
The absolute packet number, regardless of how many packets were dropped by filters. Note: packets dropped with BPF filters are not counted in this value
Appendix: Migrating old filters

generic::absolute_pktnum.every_100 -> N/A
The absolute packet number, regardless of how many packets were dropped by filters. Note: packets dropped with BPF filters are not counted in this value.

generic::absolute_pktnum.every_1000 -> N/A
The absolute packet number, regardless of how many packets were dropped by filters. Note: packets dropped with BPF filters are not counted in this value.

generic::absolute_pktnum.every_10000 -> N/A
The absolute packet number, regardless of how many packets were dropped by filters. Note: packets dropped with BPF filters are not counted in this value.

generic::absolute_pktnum.every_100000 -> N/A
The absolute packet number, regardless of how many packets were dropped by filters. Note: packets dropped with BPF filters are not counted in this value.

generic::absolute_pktnum.every_1000000 -> N/A
The absolute packet number, regardless of how many packets were dropped by filters. Note: packets dropped with BPF filters are not counted in this value.

generic::application.str -> generic.application
Traffic type classification (e.g. 'Email' or 'Web')

generic::bits -> generic.bits
Bit Count

generic::buffer.arrival.time -> N/A
Estimated arrival time of buffer to the receiving host.

generic::buffer.departure.time -> N/A
Estimated departure time of buffer from the sending host.

generic::bytes -> generic.bytes
Byte Count

generic::bytes_normalized -> generic.bytes_normalized
Byte count normalized with a minimum length of 60

generic::counter.double -> generic.packets_float
Packet Count as a floating point number (used for time averaging)

generic::empty_string.str -> generic.empty_string.str
empty constant string that can be used to add a dimension in cube configurations that have only values

generic::end.time.hints -> message.end_time
Last packet timestamp of the stream.

generic::end.time.unprocessed -> message.end_time_estimate
Estimated arrival time of packet to the receiving host

generic::fcs_error -> generic.fcs_error
FCS error code

generic::fcs_error.str -> generic.fcs_error_description
FCS error description

generic::hour -> N/A
Numeric value that ranges from 0 to 24.

generic::ignored.str -> generic.empty_string.str
This field can be used to fill a dimension with a value that will be ignored by the chart.
Filters Guide

generic::max_microburst_bits_100ms -> generic.max_microburst_100ms.bits
   Maximum bit count in a 100ms interval

generic::max_microburst_bits_10ms -> generic.max_microburst_10ms.bits
   Maximum bit count in a 10ms interval

generic::max_microburst_bits_1ms -> generic.max_microburst_1ms.bits
   Maximum bit count in a 1ms interval

generic::max_microburst_bytes_100ms -> generic.max_microburst_100ms.bytes
   Maximum byte count in a 100ms interval

generic::max_microburst_bytes_10ms -> generic.max_microburst_10ms.bytes
   Maximum byte count in a 10ms interval

generic::max_microburst_bytes_1ms -> generic.max_microburst_1ms.bytes
   Maximum byte count in a 1ms interval

generic::max_microburst_packets_100ms -> generic.max_microburst_100ms.packets
   Maximum packet count in a 100ms interval

generic::max_microburst_packets_10ms -> generic.max_microburst_10ms.packets
   Maximum packet count in a 10ms interval

generic::max_microburst_packets_1ms -> generic.max_microburst_1ms.packets
   Maximum packet count in a 1ms interval

generic::minute -> N/A
   Numeric value that ranges from 0 to 59.

generic::month -> N/A
   Numeric value that ranges from 1 to 12.

generic::packet_length_category -> generic.packet_length_range
   Packet length range

generic::packets -> generic.packets
   Packet Count

generic::protocol.str -> generic.protocol
   Protocol classification based on the IP protocol, TCP port or UDP port

generic::relative_pktnum -> generic.relative_pktnum
   Packet number ignoring those dropped by filters

generic::segment.end.time -> message.segment.end.time
   Estimated arrival time of the segment if the destination is an endpoint, capture time if the destination is a capture point

generic::segment.start.time -> message.segment.start.time
   Estimated departure time of the segment if the source is an endpoint, capture time if the source is a capture point

generic::start.time.hints -> message.start_time
   First packet timestamp of the stream

generic::start.time.unprocessed -> message.start_time_estimate
   Estimated departure time of packet from the sending host

generic::start_time_long -> generic.start_time
   Initial capture time expressed as a 64-bit value in nano seconds.
Appendix: Migrating old filters

generic::test.delay -> generic.test_delay
This field can be used to slow down the view processing by adding a sleep during the view processing.

generic::time_from_first -> generic.relative_time
Delta from the first packet in the capture.

generic::time_long -> generic.absolute_time
Packet time expressed as a 64-bit value in nano seconds.

generic::time_sec -> generic.absolute_time_seconds
Packet UTC time expressed in seconds from January 1st 1970 as a 32-bit value.

generic::total_bits -> generic.total_bits
Bit count including any pseudo header such as Radiotap or PPI.

generic::total_bytes -> generic.total_bytes
Byte count including any pseudo header such as Radiotap or PPI.

generic::total_packet_length_category -> N/A
Packet length including any pseudo header such as Radiotap or PPI.

generic::week_of_the_year -> N/A
Numeric value that ranges from 0 to 53.

generic::weekday -> N/A
Text value that ranges from Monday to Sunday.

generic::wire_bits -> generic.wire_bits
Bit count including Ethernet overhead (i.e. preamble + packet + CRC + inter frame gap)

generic::wire_bytes -> generic.wire_bytes
Byte count including Ethernet overhead (i.e. preamble + packet + CRC + inter frame gap)

generic::wire_overhead_bits -> generic.wire_overhead_bits
Bit count of the overhead used by the Ethernet protocol (i.e. preamble + CRC + inter frame gap)

generic::wire_overhead_bytes -> generic.wire_overhead_bytes
Byte count of the overhead used by the Ethernet protocol (i.e. preamble + CRC + inter frame gap)

http::

http::answered.request.count -> http.answered_requests
The number of HTTP requests that received a well-formed answer from the server.

http::bot.name.str -> http.bot_name
Client browser model and version, only if the client is a bot.

http::browser.str -> http.browser
Client browser model and version.

http::content.length -> http.content_length
HTTP content length.

http::content.type -> http.content_type
HTTP content type.

http::encryption.type.str -> http.scheme
Type of web traffic (http, https...).
Filters Guide

http::full.object.time -> http.object_transfer_time
  Time to transfer an object (html page, image...), from the beginning of the request to the end of a response

http::host -> http.host
  Host name in the http header

http::method -> http.method
  HTTP request type (GET, POST, etc.)

http::object.download.rate -> http.object_transfer_rate
  The rate at which an object (html page, image...), has been downloaded, dividing the object size by the transaction time (time of the request beginning to time of the response end)

http::object.len -> http.object_length
  Length of the http object, in bytes. This is different from the HTTP Content Length because this length is calculate by observing the packets and not retrieved from the http header. Therefore, it returns the size of chunked encoded objects as well

http::parameters -> http.parameters
  HTTP resource parameters

http::referer -> http.referer
  Referer host name in the http request

http::request.count -> http.requests
  The number of HTTP requests.

http::request.duration.time -> http.duration
  Request duration measured from the first to the last packet seen for a request

http::request.start.absolute.time -> http.start_time
  Time of the first request packet

http::resource -> http.path
  HTTP resource path and name

http::resource.no.param -> http.path_no_param
  HTTP resource path and name without variable parameters

http::status.code -> http.status_code
  HTTP Status Code

http::status.code.str -> http.status_description
  Human readable version of the HTTP status code

http::uri -> http.uri
  HTTP Request URI

http::uri.no.param -> http.uri_no_param
  HTTP Request URI Without Variable Parameters

http::user.agent -> http.user_agent
  HTTP user agent
Appendix: Migrating old filters

**icmp::**

*icmp::bits -> N/A*
ICMP Bit Count, size in bits of the whole packet containing ICMP

*icmp::bytes -> N/A*
ICMP Byte Count, size in bytes of the whole packet containing ICMP

*icmp::checksum -> icmp.checksum*
Checksum of the ICMP header and payload

*icmp::code -> icmp.code*
Code of ICMP message

*icmp::code.str -> icmp.code_name*
Description of the code of ICMP message (e.g. Source Route Failed)

*icmp::dest_unreachable.code -> icmp.dest_unreachable.code*
Destination unreachable code of the error

*icmp::dest_unreachable.code.str -> icmp.dest_unreachable.code_description*
Description of destination unreachable code error

*icmp::dest_unreachable.next_mtu -> icmp.dest_unreachable.next_mtu*
Next-Hop MTU

*icmp::echo_reply.identifier -> icmp.echo_reply.identifier*
Echo reply identifier

*icmp::echo_reply.response_time -> icmp.echo_reply.response_time*
Time that elapsed from when the request was issued to when the response is received

*icmp::echo_reply.response_time.category.str -> icmp.echo_reply.response_time_range*
Time category that elapsed from when the request was issued to when the response is received

*icmp::echo_reply.seq_num -> icmp.echo_reply.seqnum*
Used to match echo request with the desired reply.

*icmp::echo_request.identifier -> icmp.echo_request.identifier*
Echo request identifier

*icmp::echo_request.seq_num -> icmp.echo_request.seqnum*
Used to match echo request with the desired reply.

*icmp::packets -> N/A*
ICMP packet count

*icmp::time_exceeded.code -> icmp.time_exceeded.code*
Time exceeded code of the error

*icmp::time_exceeded.code.str -> icmp.time_exceeded.code_name*
Description of time exceeded code error

*icmp::type -> icmp.type*
Type of ICMP message

*icmp::type.str -> icmp.type_name*
Description of the type of ICMP message (e.g. Echo reply)
**ieee80211::**

`ieee80211::bits.retransmitted` -> `wlan.retransmitted_bits`
Retransmitted bits

`ieee80211::bssid.essid.str` -> `wlan.bssid.essid`
ESSID string.

`ieee80211::bssid.essid.str.data` -> `N/A`
ESSID string, but only for data frames.

`ieee80211::bssid.essid_dirty.str` -> `N/A`
ESSID string only for those packets that are on the wrong channel. Unless used for debug purposes, bssid.essid should be used.

`ieee80211::bssid.essid_pure.str` -> `N/A`
ESSID string whether or not it is bound to the AP's channel.

`ieee80211::bssid.str` -> `wlan.bssid`
BSSID MAC address string.

`ieee80211::bssid.vendor.str` -> `wlan.bssid.vendor`
BSSID vendor name string.

`ieee80211::bssid.vendor_with_mac.str` -> `wlan.bssid.vendor_with_mac`
BSSID vendor name with last 3 bytes of the MAC address.

`ieee80211::bytes.retransmitted` -> `wlan.retransmitted_bytes`
Retransmitted bytes

`ieee80211::channel.is_correct` -> `wlan.is_correct_channel`
Determines if the packet was captured on the correct channel, or if it has strayed to a neighboring channel.

`ieee80211::channel.stray` -> `wlan.channel_stray`
The number of channels that a packet has strayed from the channel from which it was transmitted. A positive result indicates the receive channel was above the intended channel.

`ieee80211::channel.usage` -> `wlan.channel_usage`
The usage of a channel in percent.

`ieee80211::filter_bpf_mac_convo.str` -> `N/A`
The BPF filter for the MAC conversation.

`ieee80211::fragment_number` -> `wlan.fragment_number`
The fragment number of this 802.11 frame.

`ieee80211::frame_control.association_failed` -> `wlan.is_association_failed`
Determines if the current packet is a Failed Association Response

`ieee80211::frame_control.authentication_failed` -> `wlan.is_authentication_failed`
Determines if the current packet is a Failed Authentication

`ieee80211::frame_control.control_retry` -> `wlan.frame_control.control_retry`
Resent frame

`ieee80211::frame_control.control_retry.str` -> `wlan.frame_control.control_retry_str`
Resent frame

`ieee80211::frame_control.control_retry.text.str` ->
`wlan.frame_control.control_retry_type`
Retransmission Type
Appendix: Migrating old filters

```
ieee80211::frame_control.destination_type.str -> wlan.frame_control.dst_type
   The type, AP or STA, that the destination wireless node represents.

ieee80211::frame_control.from_ds -> wlan.frame_control.from_ds
   Frame originated from the Distribution System (DS)

ieee80211::frame_control.more_data -> wlan.frame_control.more_data
   More data

ieee80211::frame_control.more_fragments -> wlan.frame_control.more_fragments
   More fragments

ieee80211::frame_control.order_ht_control -> wlan.frame_control.order_ht_control
   Order or presence of HT Control field.

ieee80211::frame_control.power_management -> wlan.frame_control.power_management
   Power Management

ieee80211::frame_control.protected_frame -> wlan.frame_control.protected_frame
   Encrypted frame

ieee80211::frame_control.protected_frame.str -> N/A
   Encrypted frame

ieee80211::frame_control.protection_type.str -> wlan.frame_control.protection_type
   The type of encryption used

ieee80211::frame_control.protection_type_simple.str -> wlan.frame_control.protection_type_ap
   The type of encryption used based on the AP to which the client is associated.

ieee80211::frame_control.protocol_version -> wlan.frame_control.protocol_version
   Protocol Version

ieee80211::frame_control.source_type.custom.str -> wlan.frame_control.src_type_custom
   The type, AP, STA, or Probing, that the originating wireless node represents.

ieee80211::frame_control.source_type.str -> wlan.frame_control.src_type
   The type, AP or STA, that the originating wireless node represents.

ieee80211::frame_control.subtype -> wlan.frame_control.subtype
   Frame Subtype

ieee80211::frame_control.subtype.authentication.count.double -> wlan.authentication_packets
   Determines if the current packet is an authentication

ieee80211::frame_control.subtype.deauthentication.count.double -> wlan.deauthentication_packets
   Determines if the current packet is a deauthentication

ieee80211::frame_control.to_ds -> wlan.frame_control.to_ds
   Frame destined for the Distribution System (DS)

ieee80211::frame_control.to_from_ds.str -> wlan.frame_control.to_from_ds
   Frame direction description

ieee80211::frame_control.type -> wlan.frame_control.type
   Frame Type

ieee80211::frame_control.type.str -> wlan.frame_control.type_name
   Frame Type String
```
Filters Guide

```
ieee80211::frame_control.type_subtype.is_association_request -> wlan.association_packets
   Determines if the current packet is an Association Request

ieee80211::frame_control.type_subtype.is_authentication -> wlan.is_authentication
   Determines if the current packet is an authentication

ieee80211::frame_control.type_subtype.is_deauthentication -> wlan.is_deauthentication
   Determines if the current packet is a deauthentication

ieee80211::frame_control.type_subtype.is_disassociation -> wlan.disassociation_packets
   Determines if the current packet is a Disassociation

ieee80211::frame_control.type_subtype.is_disassociation.str -> wlan.is_disassociation_str
   Determines if the current packet is a Disassociation

ieee80211::frame_control.type_subtype.is_probe_request -> wlan.is_probe_request
   Determines if the current packet is a Probe Request

ieee80211::frame_control.type_subtype.is_probe_request.str -> wlan.is_probe_request_str
   Determines if the current packet is a Probe Request

ieee80211::frame_control.type_subtype.is_reassociation_request -> wlan.reassociation_packets
   Determines if the current packet is a Reassociation Request

ieee80211::frame_control.type_subtype.str -> wlan.frame_control.type_subtype_name
   Frame Type/Subtype String

ieee80211::info_fields.channel -> wlan.info_fields.channel
   Channel

ieee80211::info_fields.extended_supported_rates.str -> wlan.info_fields.extended_supported_rates
   Extended Supported Rates

ieee80211::info_fields.ht_capabilities.ampdu_parameters.max_length -> wlan.info_fields.ht_capabilities.ampdu_max_length
   A-MPDU Parameters - Maximum Length

ieee80211::info_fields.ht_capabilities.ampdu_parameters.mpdu_density.str -> wlan.info_fields.ht_capabilities.ampdu_mpdu_density
   A-MPDU Parameters - MPDU Density

ieee80211::info_fields.ht_capabilities.info.40mhz.str -> wlan.info_fields.ht_capabilities.40mhz
   Capabilities Information - 40MHz channels allowed

ieee80211::info_fields.ht_capabilities.info.amsdu_length.str -> wlan.info_fields.ht_capabilities.amsdu_length
   Capabilities Information - Maximum A-MSDU Length

ieee80211::info_fields.ht_capabilities.info.delayed_block_ack.str -> wlan.info_fields.ht_capabilities.delayed_block_ack
   Capabilities Information - Delayed Block Ack supported
```
Appendix: Migrating old filters

`ieee80211::info_fields.ht_capabilities.info.greenfield.str` -> `wlan.info_fields.ht_capabilities.greenfield`
Capabilities Information - Greenfield Support

`ieee80211::info_fields.ht_capabilities.info.psmstr` -> `wlan.info_fields.ht_capabilities.psm`
Capabilities Information - Power Save Multi-Poll (PSMP) supported

`ieee80211::info_fields.ht_capabilities.info.sgi_20mhz.str` -> `wlan.info_fields.ht_capabilities.sgi_20mhz`
Capabilities Information - Short GI at 20MHz supported

`ieee80211::info_fields.ht_capabilities.info.sgi_40mhz.str` -> `wlan.info_fields.ht_capabilities.sgi_40mhz`
Capabilities Information - Short GI at 40MHz supported

`ieee80211::info_fields.ht_capabilities.info.supported_width.str` -> `wlan.info_fields.ht_capabilities.supported_width`
Capabilities Information - Supported Channel Width

`ieee80211::info_fields.ssid.str` -> `wlan.info_fields.ssid`
SSID

`ieee80211::info_fields.supported_rates.str` -> `wlan.info_fields.supported_rates`
Supported Rates

`ieee80211::packets.retransmitted` -> `wlan.retransmitted_packets`
Retransmitted packets

`ieee80211::roaming.end.bssid.str` -> `wlan.roaming.end_bssid`
BSSID of the AP that the roaming station joined.

`ieee80211::roaming.end.channel.str` -> `wlan.roaming.end_channel`
The channel where a station was when it finished a roaming operation.

`ieee80211::roaming.end.pktnum` -> `wlan.roaming.end_pktnum`
The packet number in the original trace file where a station ends a roaming operation.

`ieee80211::roaming.end.time` -> `wlan.roaming.end_time`
The time when a station ends a roaming operation.

`ieee80211::roaming.is_first_roaming_pkt` -> `wlan.roaming.is_first_roaming_pkt`
Determines if the packet is the first packet of a roaming operation.

`ieee80211::roaming.start.bssid.str` -> `wlan.roaming.start_bssid`
BSSID of the AP that the roaming station left.

`ieee80211::roaming.start.channel.str` -> `wlan.roaming.start_channel`
The channel where a station was when it started a roaming operation.

`ieee80211::roaming.start.pktnum` -> `wlan.roaming.start_pktnum`
The packet number in the original trace file where a station starts a roaming operation.

`ieee80211::roaming.start.time` -> `wlan.roaming.start_time`
The time when a station starts a roaming operation.

`ieee80211::roaming.time` -> `wlan.roaming.time`
The time a station takes to roam.

`ieee80211::sequence_number` -> `wlan.sequence_number`
The sequence number of this 802.11 frame.
Filters Guide

**ip::**

- **ip::a_net.str -> ip.a_net**
  IP Class A (/8) source or destination subnet
- **ip::application.clientserver.str -> ip.clientserver_application**
  TCP port converted into a traffic type string (e.g. 'Email' or 'Web'), but only for applications that are client/server (i.e.: no instant messaging)
- **ip::b_net.str -> ip.b_net**
  IP Class B (/16) source or destination subnet
- **ip::bits -> ip.bits**
  Number of bits
- **ip::bytes -> ip.bytes**
  Number of bytes
- **ip::c_net.str -> ip.c_net**
  IP Class C (/24) source or destination subnet
- **ip::data.direction.str -> ip.direction**
  Direction of the packet 'Inbound' (external sender, local receiver), 'Outbound' (local sender, external receiver) or 'Internal' (local sender, local receiver)
- **ip::destination_a_net.str -> ip.dst_a_net**
  Destination Class A (/8) Subnet
- **ip::destination_b_net.str -> ip.dst_b_net**
  Destination Class B (/16) Subnet
- **ip::destination_c_net.str -> ip.dst_c_net**
  Destination Class C (/24) Subnet
- **ip::destination_ip.country.geoip -> ip.dst_country**
  Destination Country Based on a GeoIP lookup
- **ip::destination_ip.delivery_type.str -> ip.dst_delivery_type**
  Description of the delivery type (Unicast, Broadcast, Multicast, Source-Specific Multicast or GLOP)
- **ip::destination_ip.internal.str -> ip.dst_internal**
  IP address of the destination interface if the host is in the internal net, otherwise 'Remote'
- **ip::destination_ip.is_private -> ip.dst_is_private**
  Indication of whether the IP address of the destination interface is private
- **ip::destination_ip.local.str -> ip.dst_local**
  Description of the destination IP location (Local or Remote)
- **ip::destination_ip.str -> ip.dst**
  IP address of the destination interface
- **ip::destination_mac.masked.str -> N/A**
  Destination MAC address string, masked to show 'n.a.' in case the address is not local.
- **ip::destination_mac.vendor.masked.str -> N/A**
  Destination vendor name string, masked to show 'n.a.' in case the address is not local.
- **ip::dscp -> ip.dscp**
  Differentiated service code point
Appendix: Migrating old filters

**ip::dscp.str** -> **ip.dscp_name**
Description of the differentiated service code point

**ip::flags.dont_fragment** -> **ip.flags.dont_fragment**
Indication of whether the IP Don't Fragment flag is set

**ip::flags.more_fragments** -> **ip.flags.more_fragments**
Indication of whether the IP More fragments flag is set

**ip::fragment_offset** -> **ip.flags.fragment_offset**
Position of the fragment in the total datagram measured in 64 bit units

**ip::fragmented_traffic** -> **ip.is_fragmented_str**
Description of the packet by fragmentation status (Fragmented or Not Fragmented)

**ip::header_checksum** -> **ip.header_checksum**
Checksum of just the IP header itself

**ip::header_checksum.valid** -> **ip.header_checksum_valid**
Indication of whether the checksum is valid

**ip::header_checksum.valid.str** -> N/A
The validity of the checksum.

**ip::header_length** -> **ip.header_length**
Length of the header in 32 bit words

**ip::id** -> **ip.id**
Unique identifier for this datagram

**ip::ip.country.geoip** -> **ip.country**
Source and Destination Country Based on a GeoIP lookup

**ip::ip.local.str** -> **ip.address_local**
Description of the IP addresses of the source and destination interfaces location (Local or Remote)

**ip::ip.str** -> **ip.address**
Source or Destination IP address

**ip::is.clientserver.application** -> **ip.is_clientserver_application**
Indication of whether the packet contains a client/server application traffic (e.g. email, web, database)

**ip::local.bits** -> N/A
counts IP bits that are local (i.e. both the source and the destination are inside the subnet)

**ip::local.bytes** -> N/A
counts IP bytes that are local (i.e. both the source and the destination are inside the subnet)

**ip::lower_ip.str** -> **ip.lower_ip**
IP address of the lower host

**ip::mac.masked.str** -> N/A
Source and Destination MAC address string, masked to show 'n.a.' in case the address is not local.

**ip::mac.vendor.masked.str** -> N/A
Source and Destination vendor name string, masked to show 'n.a.' in case the address is not local.

**ip::nflows** -> **ip.nflows**
Number of unique flows

**ip::packets** -> **ip.packets**
Number of packets
Filters Guide

```plaintext
ip::protocol -> ip.protocol
Protocol type

ip::protocol.str -> ip.protocol_name
Name of the protocol type (e.g. TCP)

ip::remote.bits -> N/A
counts IP bits that are with remote locations (i.e. either the source or the destination are outside the subnet)

ip::remote.bytes -> N/A
counts IP bytes that are with remote locations (i.e. either the source or the destination are outside the subnet)

ip::source_a_net.str -> ip.src_a_net
Source Class A (/8) Subnet

ip::source_b_net.str -> ip.src_b_net
Source Class B (/16) Subnet

ip::source_c_net.str -> ip.src_c_net
Source Class C (/24) Subnet

ip::source_ip.country.geoip -> ip.src_country
Source Country Based on a GeoIP lookup

ip::source_ip.internal.str -> ip.src_internal
IP address of the source host if the host is in the internal net, otherwise 'Remote'

ip::source_ip.is_private -> ip.src_is_private
Indication of whether the IP address of the source host is private

ip::source_ip.local.str -> ip.src_local
Description of the source IP location (Local or Remote)

ip::source_ip.str -> ip.src
IP address of the source host

ip::source_mac.masked.str -> N/A
Source MAC address string, masked to show 'n.a.' in case the address is not local.

ip::source_mac.vendor.masked.str -> N/A
Source vendor name string, masked to show 'n.a.' in case the address is not local.

ip::time_to_live -> ip.time_to_live
Maximum time in seconds that a datagram will be allowed to survive

ip::tos.delay -> ip.tos.delay
Indication of whether the IP TOS Minimize delay flag is set

ip::tos.delay.str -> N/A
Minimize delay

ip::tos.monetary_cost -> ip.tos.monetary_cost
Indication of whether the IP TOS Minimize monetary cost flag is set

ip::tos.monetary_cost.str -> N/A
Minimize monetary cost

ip::tos.precedence -> ip.tos.precedence
Indication of whether the IP TOS Precedence flag is set
```

40
Appendix: Migrating old filters

ip::tos.precedence.str -> N/A
Precedence specified for the datagram.

ip::tos.reliability -> ip.tos.reliability
Indication of whether the IP TOS Maximize reliability flag is set

ip::tos.reliability.str -> N/A
Maximize reliability

ip::tos_throughput -> ip.tos_throughput
Indication of whether the IP TOS Maximize throughput flag is set

ip::tos_throughput.str -> N/A
Maximize throughput

ip::total_length -> ip.total_length
Length of the datagram in bytes

ip::transport.destination_port -> ip.transport.dst_port
Transport protocol (TCP/UDP) destination port

ip::transport.port -> ip.transport.port
TCP/UDP both source and destination port

ip::transport.source_port -> ip.transport.src_port
Transport protocol (TCP/UDP) source port

ip::transport_payload_length<bits> -> ip.transport.payload_length_bits
Length of the TCP or UDP payload in bits

ip::transport_payload_length<bytes> -> ip.transport.payload_length_bytes
Length of the TCP or UDP payload in bytes

ip::upper_ip.str -> ip.upper_ip
IP address of the upper host

ip::version -> ip.version
Format of the IP header (e.g.)

ip::version.str -> ip.version_name
Format of the IP header (e.g. IP)

ipres::
ipres::country.str -> N/A
Country obtained from the internet domain.

ipres::destination_country.str -> N/A
Destination country obtained from the internet domain.

ipres::destination_domain.str -> ip.dst_domain
Destination Internet domain

ipres::domain.str -> ip.domain
Internet Domain

ipres::source_country.str -> N/A
Source country obtained from the internet domain.
Filters Guide

ipres::source_domain.str -> ip.src_domain
  Source Internet domain

mac::
mac::arp.str -> mac.is_arp_str
  Description of the traffic class (ARP, RARP or Not ARP)
mac::broadcast_bytes -> mac.broadcast_bytes
  Number of Broadcast Bytes
mac::broadcast_count -> N/A
  Number of Broadcast Packets.
mac::broadcast_packets -> mac.broadcast_packets
  Number of Broadcast Packets
mac::destination_mac.delivery_type.str -> mac.dst_delivery_type
  Type of delivery used the destination for the MAC layer transmission
mac::destination_mac.str -> mac.dst
  Destination MAC address string
mac::destination_mac.vendor.str -> mac.dst_vendor
  Destination vendor name
mac::destination_mac.vendor_with_mac.str -> mac.dst_vendor_with_mac
  Destination vendor name with last 3 bytes of the MAC address
mac::ip.version.str -> mac.ip_version
  IP Version
mac::local.str -> mac.is_local_str
  Description of the traffic location (Local -both the source and the destination are inside the subnet- or Remote -either the source or the destination are inside the subnet-)
mac::mac.str -> mac.address
  Source and Destination MAC address
mac::mac.vendor.str -> mac.vendor
  Source and Destination vendor name string
mac::mac.vendor_with_mac.str -> mac.vendor_with_mac
  Source and Destination vendor name with last 3 bytes of the MAC addresses
mac::mpls.label -> mac.mpls_label
  MPLS label
mac::mpls.str -> mac.is_mpls_str
  MPLS vs. other
mac::mpls.tc -> mac.mpls_traffic_class
  MPLS Traffic Class
mac::multicast_bytes -> mac.multicast_bytes
  Number of Multicast Bytes
mac::multicast_count -> N/A
  Number of Multicast Packets.
Appendix: Migrating old filters

mac::multicast_packets -> mac.multicast_packets
   Number of Multicast Packets

mac::protocol_type -> mac.protocol_type
   Protocol Type

mac::protocol_type.str -> mac.protocol_type_name
   Description of the protocol type

mac::source_mac.delivery_type.str -> mac.src_delivery_type
   Type of delivery used by the source for the MAC layer transmission

mac::source_mac.str -> mac.src
   Source MAC address

mac::source_mac.vendor.str -> mac.src_vendor
   Source vendor name

mac::source_mac.vendor_with_mac.str -> mac.src_vendor_with_mac
   Source vendor name with last 3 bytes of the MAC address

mac::station_number -> mac.stations
   Number of MAC transmitting or receiving endpoints.

mac::vlan.id -> mac.vlan_id
   VLAN Identifier

mac::vlan.pri -> mac.vlan_priority
   802.1p CoS (0 to 7, 0 best-effort and 7 real-time)

multi_segment::

multi_segment::capture_point -> multi_segment.capture_point
   Capture Point Index

multi_segment::capture_point.dst -> multi_segment.capture_point_dst
   Destination Capture Index

multi_segment::capture_point.src -> multi_segment.capture_point_src
   Source Capture Index

multi_segment::delay -> multi_segment.delay
   Segment delay

multi_segment::dropped -> multi_segment.dropped
   Segment dropped packets

multi_segment::dropped.grid -> multi_segment.dropped
   Segment dropped packets

multi_segment::out_of_order -> N/A
   Segment out of order packets

multi_segment::out_of_order.grid -> N/A
   Segment out of order packets

multi_segment::reference.dscp -> multi_segment.reference_dscp
   DSCP of the reference packet
multi_segment::reference.dscp.str -> multi_segment.reference_dscp_name
DSCP description of the reference packet

multi_segment::rtt.owd -> multi_segment.round_trip_time
Segment TCP Round Trip Time computed using the one way delay in the two directions

multi_segment::tcp.connection.duration.time -> multi_segment.tcp_connection.duration
TCP connection duration, considering the first packet seen among all the capture points

multi_segment::tcp.cp.count -> multi_segment.tcp_connection.capture_points
Count of the current number of CP, to have an updated value use the MAX computation in the extractor

multi_segment::tcp.first.conn.bits -> multi_segment.tcp_connection.normalized_bits
Number of bits normalized using the first CP in which the TCP connection is seen

multi_segment::tcp.first.conn.bytes -> multi_segment.tcp_connection.normalized_bytes
Number of bytes normalized using the first CP in which the TCP connection is seen

multi_segment::tcp.first.conn.packets -> multi_segment.tcp_connection.normalized_packets
Number of packets normalized using the first CP in which the TCP connection is seen

multi_segment::tcp.start.absolute.time -> multi_segment.tcp_connection.start_time
TCP connection start time, timestamp of the first packet seen among all the capture points

multi_segment::tcp.start.time -> N/A
TCP connection start time, considering the first packet seen in all the CPs

pseudo::

pseudo::80211_common.bits.invalid -> wlan_link.invalid_bits
Invalid Bits

pseudo::80211_common.bytes.invalid -> wlan_link.invalid_bytes
Invalid Bytes

pseudo::80211_common.channel.freq -> wlan_link.channel_frequency
PPI 802.11 Common / Radiotap - Channel Frequency (2412, 2417, ...)

pseudo::80211_common.channel.number -> wlan_link.channel_number
PPI 802.11 Common / Radiotap - Channel Number (001, 002, ...)

pseudo::80211_common.channel.str -> wlan_link.channel
PPI 802.11 Common / Radiotap - Channel Representation String (BG 001, BG 002, ...)

pseudo::80211_common.channel.type -> wlan_link.channel_type
PPI 802.11 Common / Radiotap - Channel Type

pseudo::80211_common.channel.type.2ghz -> wlan_link.channel_2ghz
PPI 802.11 Common / Radiotap - Channel Type - 2 GHz

pseudo::80211_common.channel.type.5ghz -> wlan_link.channel_5ghz
PPI 802.11 Common / Radiotap - Channel Type - 5 GHz

pseudo::80211_common.channel.type.cck -> wlan_link.channel_cck
PPI 802.11 Common / Radiotap - Channel Type - CCK
Appendix: Migrating old filters

pseudo::80211_common.channel.type.designator.str -> wlan_link.channel_designator
PPI 802.11 Common / Radiotap - Channel Type Designator String. For PPI valid values are 'A', 'B', 'G', 'N' for Radiotap valid values are 'A', 'B', 'G'

pseudo::80211_common.channel.type.designator_per_station.str -> wlan_link.channel_designator_per_station
PPI 802.11 Common / Radiotap - Channel Type Designator String match for each station. For PPI valid values are 'A', 'B', 'G', 'N' for Radiotap valid values are 'A', 'B', 'G'

pseudo::80211_common.channel.type.dynamic -> wlan_link.channel_dynamic
PPI 802.11 Common / Radiotap - Channel Type - Dynamic

pseudo::80211_common.channel.type.freq_band.str -> wlan_link.channel_freq_band
PPI 802.11 Common / Radiotap - Channel Type - Frequency band string

pseudo::80211_common.channel.type.gfsk -> wlan_link.channel_gfsk
PPI 802.11 Common / Radiotap - Channel Type - GFSK

pseudo::80211_common.channel.type.ofdm -> wlan_link.channel_ofdm
PPI 802.11 Common / Radiotap - Channel Type - OFDM

pseudo::80211_common.channel.type.passive -> wlan_link.channel_passive
PPI 802.11 Common / Radiotap - Channel Type - Passive

pseudo::80211_common.channel.type.str -> wlan_link.channel_type_name
PPI 802.11 Common / Radiotap - Channel Type Summary String

pseudo::80211_common.channel.type.turbo -> wlan_link.channel_turbo
PPI 802.11 Common / Radiotap - Channel Type - Turbo

pseudo::80211_common.channel_complete.str -> wlan_link.channel_description
PPI 802.11 Common / Radiotap - Complete Channel Representation String (2412 MHz [BG 001], 2417 MHz [BG 002], ...)

pseudo::80211_common.fhss.hopset -> wlan_link.fhss.hopset
PPI 802.11 Common / Radiotap - Frequency Hopping Spread Spectrum (FHSS) hopset

pseudo::80211_common.fhss.pattern -> wlan_link.fhss.pattern
PPI 802.11 Common / Radiotap - Frequency Hopping Spread Spectrum (FHSS) pattern

pseudo::80211_common.flags -> wlan_link.flags
PPI 802.11 Common - Flags

pseudo::80211_common.flags.fcs_invalid -> wlan_link.flags.fcs_invalid
PPI 802.11 Common / Radiotap - Invalid FCS Invalid Packets

pseudo::80211_common.flags.fcs_invalid.str -> N/A
PPI 802.11 Common / Radiotap - FCS Invalid [String]

pseudo::80211_common.flags.fcs_present -> wlan_link.flags.fcs_present
PPI 802.11n Common / Radiotap - FCS Present

pseudo::80211_common.flags.fcs_present.str -> N/A
PPI 802.11n Common / Radiotap - FCS Present [String]

pseudo::80211_common.flags.phy_error -> wlan_link.flags.phy_error
PPI 802.11n Common - PHY Error

pseudo::80211_common.flags.phy_error.str -> N/A
PPI 802.11n Common - PHY Error [String]
pseudo::80211_common.flags.tsf_units -> wlan_link.flags.tsf_units
PPI 802.11n Common / Radiotap - Timing Synchronization Function (TSF) Units

pseudo::80211_common.flags.tsf_units.str -> N/A
PPI 802.11n Common / Radiotap - Timing Synchronization Function (TSF) Units [String]

pseudo::80211_common.noise -> wlan_link.noise
PPI 802.11 Common / Radiotap - RF Noise

pseudo::80211_common.rate -> wlan_link.rate
PPI 802.11 Common / Radiotap - Rate

pseudo::80211_common.signal -> wlan_link.signal
PPI 802.11 Common / Radiotap - RF Signal

pseudo::80211_common.timer -> wlan_link.timer
PPI 802.11 Common / Radiotap - Timing Synchronization Function (TSF) Timer

pseudo::80211_common.timer.absolute_delta -> wlan_link.timer_absolute_delta
PPI 802.11 Common / Radiotap - Timing Synchronization Function (TSF) Timer Absolute Delta

pseudo::80211_common.timer.delta -> wlan_link.timer_delta
PPI 802.11 Common / Radiotap - Timing Synchronization Function (TSF) Timer Delta

pseudo::80211n_mac.ampdu_id -> wlan_link.80211n_mac.ampdu_id
PPI 802.11n MAC - A-MPDU ID

pseudo::80211n_mac.flags -> wlan_link.80211n_mac.flags
PPI 802.11n MAC - Flags

pseudo::80211n_mac.flags.aggregate -> wlan_link.80211n_mac.flags.aggregate
PPI 802.11n MAC - Flags - Aggregate

pseudo::80211n_mac.flags.aggregate.str -> N/A
PPI 802.11n MAC - Flags - Aggregate String

pseudo::80211n_mac.flags.dup_rx -> wlan_link.80211n_mac.flags.dup_rx
PPI 802.11n MAC - Flags - Duplicate RX

pseudo::80211n_mac.flags.dup_rx.str -> N/A
PPI 802.11n MAC - Flags - Duplicate RX String

pseudo::80211n_mac.flags.error_following -> wlan_link.80211n_mac.flags.error_following
PPI 802.11n MAC - Flags - Aggregate delimiter CRC error after this frame

pseudo::80211n_mac.flags.error_following.str -> N/A
PPI 802.11n MAC - Flags - Aggregate delimiter CRC error after this frame string

pseudo::80211n_mac.flags.greenfield -> wlan_link.80211n_mac.flags.greenfield
PPI 802.11n MAC - Flags - Greenfield

pseudo::80211n_mac.flags.greenfield.str -> N/A
PPI 802.11n MAC - Flags - Greenfield String

pseudo::80211n_mac.flags.more_aggregates -> wlan_link.80211n_mac.flags.more_aggregates
PPI 802.11n MAC - Flags - More Aggregates

pseudo::80211n_mac.flags.more_aggregates.str -> N/A
PPI 802.11n MAC - Flags - More Aggregates String
Appendix: Migrating old filters

pseudo::80211n_mac.flags.sgi -> wlan_link.80211n_mac.flags.sgi
  PPI 802.11n MAC - Flags - SGI

pseudo::80211n_mac.flags.sgi.str -> N/A
  PPI 802.11n MAC - Flags - SGI String

pseudo::80211n_mac.flags.width -> wlan_link.80211n_mac.flags.width
  PPI 802.11n MAC - Flags - Width

pseudo::80211n_mac.flags.width.str -> N/A
  PPI 802.11n MAC - Flags - Width String

pseudo::80211n_mac.num_delim -> wlan_link.80211n_mac.num_delim
  PPI 802.11n MAC - Number of Delimiters

pseudo::80211n_mac_phy.ampdu_id -> wlan_link.80211n_mac_phy.ampdu_id
  PPI 802.11n MAC+PHY - A-MPDU ID

pseudo::80211n_mac_phy.combined_rssi -> wlan_link.80211n_mac_phy.combined_rssi
  PPI 802.11n MAC+PHY - Combined RSSI

pseudo::80211n_mac_phy.ctrl.rssi.ant0 -> wlan_link.80211n_mac_phy.ctrl_rssi_ant0
  PPI 802.11n MAC+PHY - Control Channel: RSSI Antenna 0

pseudo::80211n_mac_phy.ctrl.rssi.ant1 -> wlan_link.80211n_mac_phy.ctrl_rssi_ant1
  PPI 802.11n MAC+PHY - Control Channel: RSSI Antenna 1

pseudo::80211n_mac_phy.ctrl.rssi.ant2 -> wlan_link.80211n_mac_phy.ctrl_rssi_ant2
  PPI 802.11n MAC+PHY - Control Channel: RSSI Antenna 2

pseudo::80211n_mac_phy.ctrl.rssi.ant3 -> wlan_link.80211n_mac_phy.ctrl_rssi_ant3
  PPI 802.11n MAC+PHY - Control Channel: RSSI Antenna 3

pseudo::80211n_mac_phy.evm.chain0 -> wlan_link.80211n_mac_phy.evm_chain0
  PPI 802.11n MAC+PHY - Error Vector Magnitude (EVM): Chain 0

pseudo::80211n_mac_phy.evm.chain1 -> wlan_link.80211n_mac_phy.evm_chain1
  PPI 802.11n MAC+PHY - Error Vector Magnitude (EVM): Chain 1

pseudo::80211n_mac_phy.evm.chain2 -> wlan_link.80211n_mac_phy.evm_chain2
  PPI 802.11n MAC+PHY - Error Vector Magnitude (EVM): Chain 2

pseudo::80211n_mac_phy.evm.chain3 -> wlan_link.80211n_mac_phy.evm_chain3
  PPI 802.11n MAC+PHY - Error Vector Magnitude (EVM): Chain 3

pseudo::80211n_mac_phy.ext.channel.freq -> wlan_link.80211n_mac_phy.ext_channel.freq
  PPI 802.11n MAC+PHY - Extension Channel Frequency

pseudo::80211n_mac_phy.ext.channel.number ->
  wlan_link.80211n_mac_phy.ext_channel.number
  PPI 802.11n MAC+PHY - Extension Channel Number

pseudo::80211n_mac_phy.ext.channel.string -> wlan_link.80211n_mac_phy.ext_channel
  PPI 802.11n MAC+PHY - Extension Channel Representation String

pseudo::80211n_mac_phy.ext.channel.type -> wlan_link.80211n_mac_phy.ext_channel.type
  PPI 802.11n MAC+PHY - Extension Channel Type

pseudo::80211n_mac_phy.ext.channel.type.2ghz ->
  wlan_link.80211n_mac_phy.ext_channel.type_2ghz
  PPI 802.11n MAC+PHY - Extension Channel Type - 2 GHz

47
Filters Guide

pseudo::80211n_mac_phy.ext.channel.type.5ghz ->
wlan_link.80211n_mac_phy.ext_channel.type_5ghz
  PPI 802.11n MAC+PHY - Extension Channel Type - 5 GHz

pseudo::80211n_mac_phy.ext.channel.type.cck ->
wlan_link.80211n_mac_phy.ext_channel.type_cck
  PPI 802.11n MAC+PHY - Extension Channel Type - CCK

pseudo::80211n_mac_phy.ext.channel.type.dynamic ->
wlan_link.80211n_mac_phy.ext_channel.type_dynamic
  PPI 802.11n MAC+PHY - Extension Channel Type - Dynamic

pseudo::80211n_mac_phy.ext.channel.type.gfsk ->
wlan_link.80211n_mac_phy.ext_channel.type_gfsk
  PPI 802.11n MAC+PHY - Extension Channel Type - GFSK

pseudo::80211n_mac_phy.ext.channel.type.ofdm ->
wlan_link.80211n_mac_phy.ext_channel.type_ofdm
  PPI 802.11n MAC+PHY - Extension Channel Type - OFDM

pseudo::80211n_mac_phy.ext.channel.type.passive ->
wlan_link.80211n_mac_phy.ext_channel.type_passive
  PPI 802.11n MAC+PHY - Extension Channel Type - Passive

pseudo::80211n_mac_phy.ext.channel.type.str ->
wlan_link.80211n_mac_phy.ext_channel.type_name
  PPI 802.11n MAC+PHY - Extension Channel Type Summary String

pseudo::80211n_mac_phy.ext.channel.type.turbo ->
wlan_link.80211n_mac_phy.ext_channel.type_turbo
  PPI 802.11n MAC+PHY - Extension Channel Type - Turbo

pseudo::80211n_mac_phy.ext.channel_complete.str ->
wlan_link.80211n_mac_phy.ext_channel_complete
  PPI 802.11n MAC+PHY - Complete Extension Channel Representation String

pseudo::80211n_mac_phy.ext.rssi.ant0 -> wlan_link.80211n_mac_phy.ext_rssi_ant0
  PPI 802.11n MAC+PHY - Extension Channel: RSSI Antenna 0

pseudo::80211n_mac_phy.ext.rssi.ant1 -> wlan_link.80211n_mac_phy.ext_rssi_ant1
  PPI 802.11n MAC+PHY - Extension Channel: RSSI Antenna 1

pseudo::80211n_mac_phy.ext.rssi.ant2 -> wlan_link.80211n_mac_phy.ext_rssi_ant2
  PPI 802.11n MAC+PHY - Extension Channel: RSSI Antenna 2

pseudo::80211n_mac_phy.ext.rssi.ant3 -> wlan_link.80211n_mac_phy.ext_rssi_ant3
  PPI 802.11n MAC+PHY - Extension Channel: RSSI Antenna 3

pseudo::80211n_mac_phy.flags -> wlan_link.80211n_mac_phy.flags
  PPI 802.11n MAC+PHY - Flags

pseudo::80211n_mac_phy.flags.aggregate -> wlan_link.80211n_macphy.flags.aggregate
  PPI 802.11n MAC+PHY - Flags - Aggregate

pseudo::80211n_mac_phy.flags.aggregate.str -> N/A
  PPI 802.11n MAC+PHY - Flags - Aggregate String

pseudo::80211n_mac_phy.flags.dup_rx -> wlan_link.80211n_macphy.flags.dup_rx
  PPI 802.11n MAC+PHY - Flags - Duplicate RX

pseudo::80211n_mac_phy.flags.dup_rx.str -> N/A
  PPI 802.11n MAC+PHY - Flags - Duplicate RX String
Appendix: Migrating old filters

pseudo::80211n_mac_phy.flags.error_following -> wlan_link.80211n_mac_phy.flags.error_following
PPI 802.11n MAC+PHY - Flags - Aggregate delimiter CRC error after this frame

pseudo::80211n_mac_phy.flags.error_following.str -> N/A
PPI 802.11n MAC+PHY - Flags - Aggregate delimiter CRC error after this frame string

pseudo::80211n_mac_phy.flags.greenfield -> wlan_link.80211n_mac_phy.flags.greenfield
PPI 802.11n MAC+PHY - Flags - Greenfield

pseudo::80211n_mac_phy.flags.greenfield.str -> N/A
PPI 802.11n MAC+PHY - Flags - Greenfield String

pseudo::80211n_mac_phy.flags.more_aggregates -> wlan_link.80211n_mac_phy.flags.more_aggregates
PPI 802.11n MAC+PHY - Flags - More Aggregates

pseudo::80211n_mac_phy.flags.more_aggregates.str -> N/A
PPI 802.11n MAC+PHY - Flags - More Aggregates String

pseudo::80211n_mac_phy.flags.sgi -> wlan_link.80211n_mac_phy.flags.sgi
PPI 802.11n MAC+PHY - Flags - SGI

pseudo::80211n_mac_phy.flags.sgi.str -> N/A
PPI 802.11n MAC+PHY - Flags - SGI String

pseudo::80211n_mac_phy.flags.width -> wlan_link.80211n_mac_phy.flags.width
PPI 802.11n MAC+PHY - Flags - Width

pseudo::80211n_mac_phy.flags.width.str -> N/A
PPI 802.11n MAC+PHY - Flags - Width String

pseudo::80211n_mac_phy.mcs -> wlan_link.80211n_mac_phy.mcs
PPI 802.11n MAC+PHY - MCS

pseudo::80211n_mac_phy.noise.ant0 -> wlan_link.80211n_mac_phy.noise_ant0
PPI 802.11n MAC+PHY - Noise: Antenna 0

pseudo::80211n_mac_phy.noise.ant1 -> wlan_link.80211n_mac_phy.noise_ant1
PPI 802.11n MAC+PHY - Noise: Antenna 1

pseudo::80211n_mac_phy.noise.ant2 -> wlan_link.80211n_mac_phy.noise_ant2
PPI 802.11n MAC+PHY - Noise: Antenna 2

pseudo::80211n_mac_phy.noise.ant3 -> wlan_link.80211n_mac_phy.noise_ant3
PPI 802.11n MAC+PHY - Noise: Antenna 3

pseudo::80211n_mac_phy.num_delim -> wlan_link.80211n_mac_phy.num_delim
PPI 802.11n MAC+PHY - Number of Delimiters

pseudo::80211n_mac_phy.num_streams -> wlan_link.80211n_mac_phy.num_streams
PPI 802.11n MAC+PHY - Number of Streams

pseudo::80211n_mac_phy.signal.ant0 -> wlan_link.80211n_mac_phy.signal_ant0
PPI 802.11n MAC+PHY - Signal: Antenna 0

pseudo::80211n_mac_phy.signal.ant1 -> wlan_link.80211n_mac_phy.signal_ant1
PPI 802.11n MAC+PHY - Signal: Antenna 1

pseudo::80211n_mac_phy.signal.ant2 -> wlan_link.80211n_mac_phy.signal_ant2
PPI 802.11n MAC+PHY - Signal: Antenna 2
Filters Guide

pseudo::80211n_mac_phy.signal.ant3 -> wlan_link.80211n_mac_phy.signal_ant3
   PPI 802.11n MAC+PHY - Signal: Antenna 3

pseudo::channel.usage -> wlan_link.channel_usage
   The amount of time the channel is actually being used.

pseudo::channel.usage.index -> wlan_link.channel_usage_index
   The percent of the channel time used.

pseudo::dlt -> wlan_link.dlt
   PPI / Radiotap DLT

pseudo::flags -> wlan_link.flags
   PPI Flags

pseudo::flags.32bit_aligned -> N/A
   PPI Flags - 32-bit Aligned

pseudo::flags.32bit_aligned.str -> N/A
   PPI Flags - 32-bit Aligned [String]

pseudo::length -> wlan_link.length
   PPI / Radiotap Header Length

pseudo::payload_length -> wlan_link.payload_length
   Length of the payload contained within PPI not including the FCS, if present

pseudo::type.str -> wlan_link.type
   Pseudo Header Type

pseudo::version -> wlan_link.version
   PPI / Radiotap Version

rios::

rios::csh_sport_id -> rios.csh_sport_id
   Steelhead internal identifier

rios::is_sh_inner -> rios.is_sh_inner
   Indication of whether the current packet contains Steelhead inner traffic

rios::outer_client_ip.str -> rios.outer_client_ip
   Client IP address for this proxied connection

rios::outer_client_port -> rios.outer_client_port
   Client TCP port for this proxied connection

rios::outer_server_ip.str -> rios.outer_server_ip
   Server IP address for this proxied connection

rios::outer_server_port -> rios.outer_server_port
   Server TCP port for this proxied connection

rios::protocol_id -> rios.protocol_id
   Steelhead internal protocol identifier
Appendix: Migrating old filters

rtp::

rtp::rtp.codec -> rtp.codec
  Description of the RTP codec

sip::

sip::call_id -> sip.call_id
  Call-ID header field, it uniquely identifies a particular invitation or all registrations of a particular client

sip::contact -> sip.contact
  Contact header field value, it provides a URI whose meaning depends on the type of request or response it is in

sip::cseq -> sip.cseq
  CSeq header field, it contains a single decimal sequence number and the request method

sip::from -> sip.from
  Source address of a SIP Packet

sip::from.display_name -> sip.from_display_name
  Display name of FROM field

sip::from.number -> sip.from_number
  Phone Number of FROM field

sip::is_request -> sip.is_request
  Indication of whether the current packet is a SIP request message

sip::is_response -> sip.is_response
  Indication of whether the current packet is a SIP response message

sip::message_type_str -> sip.message_type
  Description of the SIP message type (method for the request, status for the response)

sip::reply_to -> sip.reply_to
  Reply-To header field, it contains a logical return URI that may be different from the From header field

sip::request_method_str -> sip.request_method
  Description of the request method

sip::response_class_str -> sip.response_class
  Response class description

sip::response_status_str -> sip.response_status
  Description of the response status

sip::to -> sip.to
  To header field, it specifies the logical recipient of the request

sip::to.display_name -> sip.to_display_name
  The optional display-name is meant to be rendered by a human-user interface

sip::to.number -> sip.to_number
  Phone Number of TO field
Filters Guide

**sip::**

**sip::user_agent -> sip.user_agent**
User-Agent header field, it contains information about the User Agent Client originating the request

**sql::**

**sql::db -> sql.default_db**
Default DB Name

**sql::error_code -> sql.error_code**
Error Code

**sql::error_count -> sql.errors**
Number of error within the current message

**sql::error_desc -> sql.error_description**
Code description for MySQL or Class description for MSSQL

**sql::error_msg -> sql.error_message**
Error Message

**sql::ip.client -> sql.client_ip**
IP Address of the SQL Client

**sql::ip.server -> sql.server_ip**
IP Address of the SQL Server

**sql::is_sql -> sql.is_sql**
Indication of whether the contains SQL traffic

**sql::is_sql_end -> sql.is_message_end**
Indication of whether the current packet is the last of a SQL request/response message

**sql::is_sql_error -> sql.is_error**
Indication of whether the current packet is a SQL Error Message

**sql::overhead.str -> sql.traffic_type**
Description of the SQL traffic type (Overhead or Data)

**sql::protocol -> sql.protocol**
Protocol Name (MySQL, MSSQL)

**sql::query.count.grid -> sql.requests**
Number of query messages

**sql::query.count.res.grid -> sql.responses**
Number of query messages with received response

**sql::query.data_transfer_time -> sql.data_transfer_time**
Query Data Transfer Time request and response data

**sql::query.data_transfer_time.grid -> sql.data_transfer_time**
Query Data Transfer Time request and response data

**sql::query.db -> sql.db**
DB Name of the query if specified in the statement, default db otherwise

**sql::query.db.grid -> sql.db**
DB Name of the query if specified in the statement, default db otherwise
Appendix: Migrating old filters

sql::query.duration -> sql.duration
Duration of the query from the request to the last packet of the response message

sql::query.duration.grid -> sql.duration
Duration of the query from the request to the last packet of the response message

sql::query.id.grid -> sql.start_time
Start time of the query corresponding to the timestamp of the first request packet

sql::query.network_delay -> sql.round_trip_time
Network Round Trip Time at request time

sql::query.operation -> sql.operation
Description of the query operation (e.g. SELECT, UPDATE, ...)

sql::query.operation.grid -> sql.operation
Description of the query operation (e.g. SELECT, UPDATE, ...)

sql::query.performance -> N/A
Performance (Service Response Time, Network RTT, Response Network Time)

sql::query.performance.str -> N/A
Performance Description (Service Response Time, Network RTT, Response Network Time)

sql::query.response_time -> sql.response_network_time
Response Network Time of the query

sql::query.result.grid -> sql.result
Result of the query (NoResponse = 0, Success = 1, Error = 2)

sql::query.server_delay -> sql.service_response_time
Service Response Time of the query

sql::query.start_time.grid -> sql.start_time
Start time of the query corresponding to the timestamp of the first request packet

sql::query.table -> sql.table
Table Name of the statement (for SELECT could be JOIN of tables)

sql::query.table.grid -> sql.table
Table Name of the statement (for SELECT could be JOIN of tables)

sql::query.text.grid -> sql.text
Statement of the query

sql::query_bits -> generic.bits
Bit Count

sql::query_bytes -> generic.bytes
Byte Count

sql::query_count -> sql.requests
Number of query messages

sql::query_pkts -> generic.packets
Packet Count

sql::sql_bits -> generic.bits
Bit Count

sql::sql_packets -> generic.packets
Packet Count
sql::stmt_count -> sql.requests
    Number of statements within a query message

sql::user -> sql.user
    SQL User

tcp::

tcp::ack_number -> tcp.is_ack
    If the ACK flag is set then this field is the next segment that the sender is expecting to receive

tcp::bits -> tcp.bits
    Number of bits, size in bits of the whole packet containing TCP

tcp::bytes -> tcp.bytes
    Number of bytes, size in bytes of the whole packet containing TCP

tcp::checksum -> tcp.checksum
    Checksum of the datagram

tcp::checksum.valid -> tcp.is_checksum_valid
    The validity of the checksum

tcp::checksum.valid.str -> N/A
    The validity of the checksum

tcp::control.ack -> tcp.flags.ack
    Controls whether or not the 'Acknowledgement Number' field is valid

tcp::control.fin -> tcp.flags.fin
    Signals the end of data

tcp::control.push -> tcp.flags.push
    Push flag

tcp::control.reset -> tcp.flags.reset
    Signals connection reset

tcp::control.syn -> tcp.flags.syn
    Signals to synchronize the sequence numbers

tcp::control.urgent -> tcp.flags.urgent
    Controls whether or not the 'Urgent Pointer' field is valid

tcp::destination_port -> tcp.dst_port
    Destination Port

tcp::destination_port.str -> tcp.dst_port_name
    The service name usually associated with the given port number

tcp::ecn.cwr -> tcp.ecn.cwr
    CWR

tcp::ecn.echo -> tcp.ecn.echo
    Echo

tcp::ecn.ns -> tcp.ecn.ns
    Signaling with Nonces.
Appendix: Migrating old filters

tcp::flags.str -> tcp.flags
Description of the TCP flags of the packet (SYN-ACK)

tcp::header_length -> tcp.header_length
Length of the TCP header in 32 bit words

tcp::identification_port -> tcp.identification_port
Identification Port

tcp::identification_port.group.str -> tcp.protocol_group
The type of traffic usually associated with the given port number or group of port numbers

tcp::identification_port.str -> tcp.identification_port_name
Service name usually associated with the given port number

tcp::lower_port -> tcp.lower_port
Lower Port

tcp::packets -> tcp.packets
Number of packets

tcp::payload_length.bits -> tcp.payload_bits
The length of the TCP payload in bits

tcp::payload_length.bytes -> tcp.payload_bytes
The length of the TCP payload in bytes

tcp::payload_range.str -> N/A
test

tcp::port -> tcp.port_pair
TCP Source or Destination Port

tcp::ports -> tcp.port_pair
Both source and destination ports

tcp::proto.str -> tcp.protocol
Service name usually associated with the given port number and the port number

tcp::reset.destination_ip.str -> N/A
The receivers of TCP reset packets

tcp::reset.source_ip.str -> N/A
The senders of TCP reset packets

tcp::sequence_number -> tcp.sequence_number
Sequence number of the first byte in this payload

tcp::source_port -> tcp.src_port
Source Port

tcp::source_port.str -> tcp.src_port_name
Service name usually associated with the given port number

tcp::upper_port -> tcp.upper_port
Upper Port

tcp::urgent_pointer -> tcp.urgent_pointer
Contains the sequence number of the last byte in a block of urgent data

tcp::window.zero -> tcp.window_zero
Number of times the window size is zero
**tcp::window.zero.destination_ip.str** -> N/A
The receivers of zero window packets

**tcp::window.zero.source_ip.str** -> N/A
The senders of zero window packets

**tcp_state::**

**tcp_state::bits.client.to.server** -> tcp.client_to_server_bits
Number of bits sent from the clients to the servers

**tcp_state::bits.server.to.client** -> tcp.server_to_client_bits
Number of bits sent from the servers to the clients

**tcp_state::buffer.arrival.time** -> N/A
Estimated time the just sent TCP buffer reaches the destination host.

**tcp_state::buffer.departure.time** -> N/A
Estimated time the just sent TCP buffer has left the sender.

**tcp_state::buffers** -> N/A
The number of buffers sent over TCP connections

**tcp_state::bytes.client.to.server** -> tcp.client_to_server_bytes
Number of bytes sent from the clients to the servers

**tcp_state::bytes.server.to.client** -> tcp.server_to_client_bytes
Number of bytes sent from the servers to the clients

**tcp_state::client.address** -> tcp.client_ip
IP address of the hosts that start TCP connections

**tcp_state::client.country.geoip** -> tcp.client_country
Country of the TCP client based on a GeoIP lookup.

**tcp_state::client.port** -> tcp.client_port
TCP port of the hosts that start TCP connections

**tcp_state::client.window** -> tcp.client_window
Size in bytes that the TCP client will accept

**tcp_state::client.server.direction.str** -> tcp.direction
TCP data direction (client-server or server-client)

**tcp_state::connection.aborted.count** -> tcp.aborted_connections
The number of TCP connections that were reset by one of the endpoints

**tcp_state::connection.attempt.count** -> tcp.connection_attempts
The number of TCP SYN packets

**tcp_state::connection.duration.time** -> tcp.connection_duration
Connection duration measured from the first to the last packet seen for a connection

**tcp_state::connection.event.type.direction.str** -> tcp.event_type_direction
Description of the connection event type (Open, Closed, Refused, Aborted). Open events contain the direction (Inbound, Outbound or Internal)

**tcp_state::connection.event.type.str** -> tcp.event_type
Description of the connection event type (Open, Closed, Refused, Aborted)
Appendix: Migrating old filters

tcp_state::connection.open.count -> tcp.open_connections
The number of TCP connections that successfully finish the three way handshake

tcp_state::connection.open.incoming.count -> tcp.open_connections_incoming
The number of TCP connections that successfully finish the three way handshake

tcp_state::connection.open.internal.count -> tcp.open_connections_internal
The number of TCP connections that successfully finish the three way handshake

tcp_state::connection.open.outgoing.count -> tcp.open_connections_outgoing
The number of TCP connections that successfully finish the three way handshake

tcp_state::connection.refused.count -> tcp.refused_connections
The number of TCP connections that failed during the three way handshake

tcp_state::connection.start.absolute.time -> tcp.connection_start_time
Time of the first packet seen for a connection

tcp_state::continual.round.trip.time -> tcp.continuous_round_trip_time
Round Trip Time, reported for each packet of the connection

tcp_state::destination.confidence -> N/A
How precise the timestamps for the specified destination IP address are, based on the observation of
the round trip time during the three way handshake.

tcp_state::error.dst -> tcp.error_dst
Addresses of the hosts that are destinations of TCP errors

tcp_state::error.src -> tcp.error_src
Addresses of the hosts that are sources of TCP errors

tcp_state::error.type.str -> tcp.error_type
Type of TCP error as a string. Can be one of the following: Retransmissions, Timeouts, Out of Order,
Lost Segments, Duplicate Acknowledgments, Zero Windows, Resets

tcp_state::external.round.trip.time -> tcp.round_trip_time_external
Round Trip Time experienced by the hosts in the local network when they communicate with hosts
outside the local network

tcp_state::external.service.response.time -> tcp.service_response_time_external
Service Response Time experienced by the hosts in the local network when they communicate with hosts
outside the local network

tcp_state::is.connection.attempt -> tcp.is_connection_attempt
Indication of whether the packet is a connection attempt

tcp_state::local.client.address -> tcp.local_client_ip
IP address of the local hosts (i.e. hosts in the local subnet) that start TCP connections

tcp_state::local.round.trip.time -> tcp.round_trip_time_local
Round Trip Time experienced by the hosts in the local network when they communicate with hosts
inside the local network

tcp_state::local.server.address -> tcp.local_server_ip
IP address of the local hosts (i.e. hosts in the local subnet) that receive TCP connections

tcp_state::local.service.response.time -> tcp.service_response_time_local
Service Response Time experienced by the hosts in the local network when they communicate with hosts
inside the local network

tcp_state::num.acked.missing.segments -> tcp.missing_segment_acks
Number of TCP ACKed Missing Segments
tcp_state::num.duplicate.acks -> tcp.duplicate_acks
  Number of TCP duplicate acknoledges

tcp_state::num.errors -> tcp.errors
  Count of the following TCP errors: retransmissions, timeouts, out of order segments, lost segments, duplicate acks, zero windows, resets

tcp_state::num.lost -> tcp.lost_segments
  Number of TCP lost segments

tcp_state::num.out.of.order -> tcp.out_of_order_segments
  Number of TCP out of order segments

tcp_state::num.request -> tcp.requests
  Number of Requests

tcp_state::num.request.double -> tcp.requests
  Number of Requests to the Server as a Floating Point Number

tcp_state::num.reset.connection.attempts -> tcp.resets
  Number of TCP reset connection attempts

tcp_state::num.retransmissions -> tcp.retransmissions
  Number of TCP Retransmissions

tcp_state::num.timeouts -> tcp.timeouts
  Number of TCP timeouts

tcp_state::packets.client.to.server -> tcp.client_to_server_packets
  Number of packets sent from the clients to the servers

tcp_state::packets.server.to.client -> tcp.server_to_client_packets
  Number of packets sent from the servers to the clients

tcp_state::request.network.time -> tcp.request_network_time
  Request Network Time

tcp_state::response.network.time -> tcp.response_network_time
  Response Network Time

tcp_state::retransmission.dst -> N/A
  Addresses of the hosts that receive TCP Retransmissions

tcp_state::retransmission.src -> N/A
  Addresses of the hosts that perform TCP Retransmissions

tcp_state::round.trip.time -> tcp.round_trip_time
  Round Trip Time

tcp_state::segment.arrival.time -> N/A
  Estimated time the TCP segment reaches the destination host. This time is calculated using the RTT of the TCP connection.

tcp_state::segment.departure.time -> N/A
  Estimated time the TCP segment leaves the sender host. This time is calculated using the RTT of the TCP connection.

tcp_state::server.address -> tcp.server_ip
  IP address of the hosts that receive TCP connections

tcp_state::server.country.geoip -> tcp.server_country
  Country of the TCP server based on a GeoIP lookup.
Appendix: Migrating old filters

**tcp_state::server.port** -> tcp.server_port
TCP port of the hosts that receive TCP connections

**tcp_state::server.window** -> tcp.server_window
Size in bytes that the TCP server will accept

**tcp_state::service.response.time** -> tcp.service_response_time
Service Response Time

**tcp_state::source.confidence** -> N/A
How precise the timestamps for the specified source IP address are, based on the observation of the round trip time during the three way handshake.

**tcp_state::transaction.detail.str** -> N/A
Transaction Timing detail

**tcp_state::transaction.detail.time** -> N/A
Transaction Timing detail

**tcp_state::transaction.total.time** -> tcp.transaction_total_time
Total Transaction Time

**tcp_state::transfer_rate.double** -> N/A
TCP segment transfer rate in bytes per second.

**tcp_state::window** -> tcp.window
Size in bytes that the sender will accept

**udp::**

**udp::bits** -> udp.bits
Number of bits, size in bits of the whole packet containing UDP

**udp::bytes** -> udp.bytes
Number of bytes, size in bytes of the whole packet containing UDP

**udp::checksum** -> udp.checksum
Checksum of the datagram

**udp::checksum.valid** -> udp.is_checksum_valid
The validity of the checksum

**udp::checksum.valid.str** -> N/A
The validity of the checksum

**udp::destination_port** -> udp.dst_port
Destination Port

**udp::destination_port.str** -> udp.dst_port_name
The service name usually associated with the given port number

**udp::identification_port** -> udp.identification_port
Identification Port

**udp::identification_port.str** -> udp.identification_port_name
Service name usually associated with the given port number

**udp::length** -> udp.length
Length of the UDP header plus payload
udp::packets -> udp.packets
Number of packets

udp::payload_length.bits -> udp.payload_bits
The length of the UDP payload in bits.

udp::payload_length.bytes -> udp.payload_bytes
The length of the UDP payload in bytes.

udp::port -> udp.port_pair
UDP Source or Destination Port

udp::proto.str -> udp.protocol
Service name usually associated with the given port number and the port number

udp::source_port -> udp.src_port
Source Port

udp::source_port.str -> udp.src_port_name
Service name usually associated with the given port number

voip::

voip::call.answered.count -> voip.answered_calls
Counts the number of answered calls (double for over time)

voip::call.arrival.time -> N/A
The arrival time of a VoIP or RTP packet.

voip::call.asr.completed.failed.str -> voip.asr_completion
Completed or Failed Calls

voip::call.asr.str -> voip.asr
Description of the ASR (Answered or Attempted)

voip::call.asr.value -> voip.asr_ratio
Answer/Seizure Ratio for all the calls

voip::call.attempted.count -> voip.attempted_calls
Counts the number of attempted calls

voip::call.attempted.count.double -> voip.attempted_calls_float
Counts the number of attempted calls (double for over time)

voip::call.call_id -> voip.call_id
Call-ID

voip::call.caller.ip.str -> voip.caller_ip
IP address of the Caller

voip::call.caller.name.str -> voip.caller_name
Caller Name

voip::call.caller.number.str -> voip.caller_number
Caller Phone Number

voip::call.caller.rtp.stream.jitter.distribution -> voip.rtp.caller_jitter_distribution
RTP caller stream Jitter Distribution (e.g. 20-40)
voip::call.completed.count.double -> voip.completed_calls_float
   The number of completed calls (double for over time)

voip::call.departure.time -> N/A
   The departure time of a VoIP or RTP packet.

voip::call.duration -> voip.duration
   Call duration, computed per packet

voip::call.end.cause.code.str -> voip.end_cause_code
   Description of the end cause code (e.g. CANCEL, BYE, 4xx, 5xx, 6xx/H.323, not available, ...)

voip::call.end.duration -> voip.final_duration
   Call duration computed at the call end

voip::call.end.status.str -> voip.final_status
   A call can terminate because has been completed, rejected or canceled by the called

voip::call.end.time -> voip.end_time
   Time the call ended

voip::call.failed.count -> voip.failed_calls
   Counts the number of failed calls

voip::call.failed.count.double -> voip.failed_calls_float
   Counts the number of failed calls (double for over time)

voip::call.has_early_stream -> voip.early_streams
   Cumulative number of early streams

voip::call.payload.type.str -> N/A
   The payload type of an individual VoIP or RTP packet.

voip::call.post.dial.delay -> voip.post_dial_delay
   Call Post Dial Delay

voip::call.post.dial.delay.grid -> voip.post_dial_delay
   Call Post Dial Delay

voip::call.receiver.ip.str -> voip.receiver_ip
   IP address of the Receiver

voip::call.receiver.name.str -> voip.receiver_name
   Receiver Name

voip::call.receiver.number.str -> voip.receiver_number
   Receiver Phone Number

voip::call.receiver.rtp.stream.jitter.distribution -> voip.rtp.receiver_jitter_distribution
   RTP receiver stream Jitter Distribution (e.g. 20-40)

voip::call.rtp.is_early_stream -> N/A
   The RTP Stream is an early stream

voip::call.rtp.stream.caller.to.receiver -> voip.rtp.caller_to_receiver
   Indication of whether the RTP stream is from the caller to the receiver

voip::call.rtp.stream.delta -> voip.rtp.delta
   Delta RTP stream Jitter

voip::call.rtp.stream.delta.grid -> voip.rtp.delta
   Delta RTP stream Jitter
Filters Guide

voip::call.rtp.stream.ip.dst.str -> voip.rtp.dst_ip
  RTP stream destination IP address
voip::call.rtp.stream.ip.src.str -> voip.rtp.src_ip
  RTP stream source IP address
voip::call.rtp.stream.jitter -> voip.rtp.jitter
  RTP stream Jitter
voip::call.rtp.stream.jitter.grid -> voip.rtp.jitter
  RTP stream Jitter
voip::call.rtp.stream.lost_packets -> voip.rtp.lost_packets
  Number of packets that have never been received for an RTP stream
voip::call.rtp.stream.lost_packets.grid -> voip.rtp.lost_packets
  Number of packets that have never been received for an RTP stream
voip::call.rtp.stream.mos -> voip.rtp.mos
  RTP stream MOS
voip::call.rtp.stream.mos.grid -> voip.rtp.mos
  RTP stream MOS
voip::call.rtp.stream.out_of_order_packets -> voip.rtp.out_of_order_packets
  Number of out of order RTP packets
voip::call.rtp.stream.out_of_order_packets.grid -> voip.rtp.out_of_order_packets
  Number of out of order RTP packets
voip::call.rtp.stream.payload.type.str -> voip.rtp.payload_type
  Description of the RTP stream payload type (e.g. PCMU)
voip::call.rtp.stream.port.dst -> voip.rtp.dst_port
  RTP stream destination port
voip::call.rtp.stream.port.src -> voip.rtp.src_port
  RTP stream source port
voip::call.rtp.stream.rfactor -> voip.rtp.rfactor
  RTP stream R-Factor
voip::call.rtp.stream.rfactor.grid -> voip.rtp.rfactor
  RTP stream R-Factor
voip::call.rtp.stream.ssrc.str -> voip.rtp.ssrc
  RTP stream Synchronization source
voip::call.sip.seer.value -> voip.sip.seer_ratio
  Session Establishment Effectiveness Ratio for all the SIP ended calls
voip::call.start.time -> voip.start_time
  Time the call started
voip::call.traffic.type.str -> voip.traffic_type
  VoIP call traffic type (H.323, SIP, Skinny, RTP or Other)
voip::call.user.ip.str -> voip.user_ip
  Caller or Receiver IP address
voip::call.user.name.str -> voip.user_name
  Name of the Caller and the Receiver
Appendix: Migrating old filters

\texttt{voip::call.user.number.str} \rightarrow \texttt{voip.user_number}

Phone Number of the Caller and the Receiver

\texttt{voip::call.voip.protocol} \rightarrow \texttt{voip.protocol}

Call VoIP Protocol

\texttt{voip::packets.double} \rightarrow \texttt{voip.packets_float}

Number of VoIP packets (float for avg computation)

\texttt{ws::}

The \texttt{ws::} filters use Wireshark fields. If \texttt{xxx} represents the Wireshark field, the general conversion from 9.x-and-earlier filters to 10.0-and-later filters is:

\begin{verbatim}
ws::xxx \rightarrow ws.xxx
\end{verbatim}

In addition, some NetShark fields equivalent to Wireshark ones (\texttt{ws::}) have been added:

\begin{verbatim}
ws::bootp.hw.mac_addr \rightarrow dhcp.client_mac
ws::bootp.ip.your \rightarrow dhcp.client_ip
ws::dns.qry.name \rightarrow dns.query.name
\end{verbatim}