Wireshark User's Guide

Version 4.1.0

Richard Sharpe, Ed Warnicke, Ulf Lamping

Preface

Foreword

Wireshark is the world's foremost network protocol analyzer, but the rich feature set can be daunting for the unfamiliar. This document is part of an effort by the Wireshark team to improve Wireshark's usability. We hope that you find it useful and look forward to your comments.

Who should read this document?

The intended audience of this book is anyone using Wireshark.

This book explains all of the basic and some advanced features of Wireshark. As Wireshark has become a very complex program, not every feature may be explained in this book.

This book is not intended to explain network sniffing in general and it will not provide details about specific network protocols. A lot of useful information regarding these topics can be found at the Wireshark Wiki at https://gitlab.com/wireshark/wireshark/wikis/.

By reading this book, you will learn how to install Wireshark, how to use the basic elements of the graphical user interface (such as the menu) and what's behind some of the advanced features that are not always obvious at first sight. It will hopefully guide you around some common problems that frequently appear for new (and sometimes even advanced) Wireshark users.

Acknowledgements

The authors would like to thank the whole Wireshark team for their assistance. In particular, the authors would like to thank:

- Gerald Combs, for initiating the Wireshark project and funding to do this documentation.
- Guy Harris, for many helpful hints and a great deal of patience in reviewing this document.
- Gilbert Ramirez, for general encouragement and helpful hints along the way.

The authors would also like to thank the following people for their helpful feedback on this document:

- Pat Eyler, for his suggestions on improving the example on generating a backtrace.
- Martin Regner, for his various suggestions and corrections.
- Graeme Hewson, for many grammatical corrections.

The authors would like to acknowledge those man page and README authors for the Wireshark project from who sections of this document borrow heavily:

- Scott Renfro from whose `mergecap` man page *mergecap*: Merging multiple capture files into one is derived.
- Ashok Narayanan from whose `text2pcap` man page *text2pcap*: Converting ASCII hexdumps to network captures is derived.

About this document

This book was originally developed by Richard Sharpe with funds provided from the Wireshark Fund. It was updated by Ed Warnicke and more recently redesigned and updated by Ulf Lamping.

It was originally written in DocBook/XML and converted to AsciiDoc by Gerald Combs.

Where to get the latest copy of this document?

The latest copy of this documentation can always be found at https://www.wireshark.org/docs/wsug_html_chunked/.

Providing feedback about this document

Should you have any feedback about this document, please send it to the authors through wireshark-dev[AT]wireshark.org.

Typographic Conventions

The following table shows the typographic conventions that are used in this guide.

Table 1. Typographic Conventions

Style	Description	Example
Italic	File names, folder names, and extensions	*C:\Development\wireshark*.
`Monospace`	Commands, flags, and environment variables	CMake's `-G` option.
`Bold Monospace`	Commands that should be run by the user	Run **`cmake -G Ninja ..`**.
[Button]	Dialog and window buttons	Press **[Launch]** to go to the Moon.
`Key`	Keyboard shortcut	Press `Ctrl` + `Down` to move to the next packet.
Menu	Menu item	Select **Go › Next Packet** to move to the next packet.

Admonitions

Important and notable items are marked as follows:

WARNING

This is a warning

You should pay attention to a warning, otherwise data loss might occur.

CAUTION

This is a caution

Act carefully (i.e., exercise care).

IMPORTANT

This is important information

RTFM - Read The Fine Manual

TIP

This is a tip

Tips are helpful for your everyday work using Wireshark.

NOTE

This is a note

A note will point you to common mistakes and things that might not be obvious.

Shell Prompt and Source Code Examples

Bourne shell, normal user

```
$ # This is a comment
$ git config --global log.abbrevcommit true
```

Bourne shell, root user

```
# # This is a comment
# ninja install
```

Command Prompt (cmd.exe)

```
>rem This is a comment
>cd C:\Development
```

PowerShell

```
PS$>#  This is a comment
PS$> choco list -l
```

C Source Code

```
#include "config.h"

/* This method dissects foos */
static int
dissect_foo_message(tvbuff_t *tvb, packet_info *pinfo _U_, proto_tree *tree _U_, void
*data _U_)
{
    /* TODO: implement your dissecting code */
    return tvb_captured_length(tvb);
}
```

Introduction

What is Wireshark?

Wireshark is a network packet analyzer. A network packet analyzer presents captured packet data in as much detail as possible.

You could think of a network packet analyzer as a measuring device for examining what's happening inside a network cable, just like an electrician uses a voltmeter for examining what's happening inside an electric cable (but at a higher level, of course).

In the past, such tools were either very expensive, proprietary, or both. However, with the advent of Wireshark, that has changed. Wireshark is available for free, is open source, and is one of the best packet analyzers available today.

Some intended purposes

Here are some reasons people use Wireshark:

- Network administrators use it to *troubleshoot network problems*
- Network security engineers use it to *examine security problems*
- QA engineers use it to *verify network applications*
- Developers use it to *debug protocol implementations*
- People use it to *learn network protocol* internals

Wireshark can also be helpful in many other situations.

Features

The following are some of the many features Wireshark provides:

- Available for *UNIX* and *Windows*.
- *Capture* live packet data from a network interface.
- *Open* files containing packet data captured with tcpdump/WinDump, Wireshark, and many other packet capture programs.
- *Import* packets from text files containing hex dumps of packet data.
- Display packets with *very detailed protocol information*.
- *Save* packet data captured.
- *Export* some or all packets in a number of capture file formats.
- *Filter packets* on many criteria.

- *Search* for packets on many criteria.
- *Colorize* packet display based on filters.
- Create various *statistics*.
- ...and *a lot more!*

However, to really appreciate its power you have to start using it.

Wireshark captures packets and lets you examine their contents. shows Wireshark having captured some packets and waiting for you to examine them.

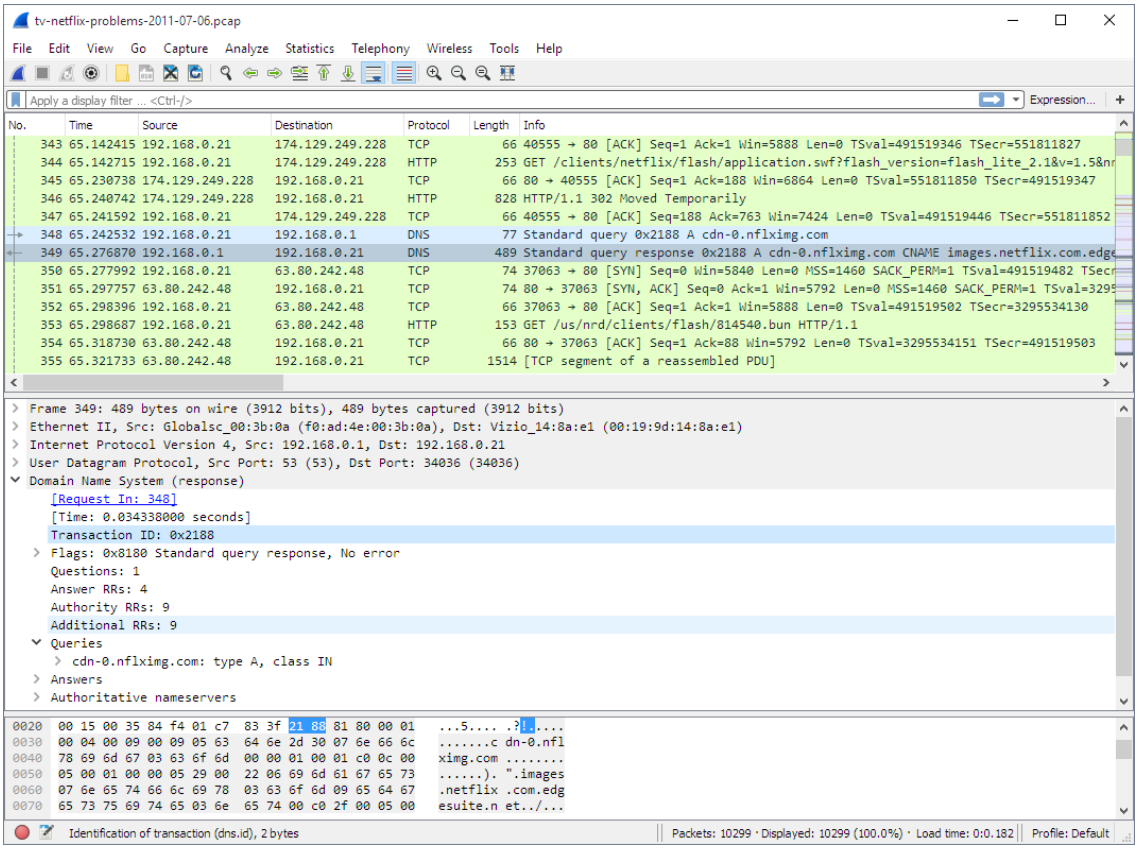


Figure 1. Wireshark captures packets and lets you examine their contents.

Live capture from many different network media

Wireshark can capture traffic from many different network media types, including Ethernet, Wireless LAN, Bluetooth, USB, and more. The specific media types supported may be limited by several factors, including your hardware and operating system. An overview of the supported media types can be found at https://gitlab.com/wireshark/wireshark/wikis/CaptureSetup/NetworkMedia.

Import files from many other capture programs

Wireshark can open packet captures from a large number of capture programs. For a list of input formats see Input File Formats.

Export files for many other capture programs

Wireshark can save captured packets in many formats, including those used by other capture programs. For a list of output formats see Output File Formats.

Many protocol dissectors

There are protocol dissectors (or decoders, as they are known in other products) for a great many protocols: see Protocols and Protocol Fields.

Open Source Software

Wireshark is an open source software project, and is released under the GNU General Public License (GPL). You can freely use Wireshark on any number of computers you like, without worrying about license keys or fees or such. In addition, all source code is freely available under the GPL. Because of that, it is very easy for people to add new protocols to Wireshark, either as plugins, or built into the source, and they often do!

What Wireshark is not

Here are some things Wireshark does not provide:

- Wireshark isn't an intrusion detection system. It will not warn you when someone does strange things on your network that he/she isn't allowed to do. However, if strange things happen, Wireshark might help you figure out what is really going on.
- Wireshark will not manipulate things on the network, it will only "measure" things from it. Wireshark doesn't send packets on the network or do other active things (except domain name resolution, but that can be disabled).

System Requirements

The amount of resources Wireshark needs depends on your environment and on the size of the capture file you are analyzing. The values below should be fine for small to medium-sized capture files no more than a few hundred MB. Larger capture files will require more memory and disk space.

NOTE	*Busy networks mean large captures* A busy network can produce huge capture files. Capturing on even a 100 megabit network can produce hundreds of megabytes of capture data in a short time. A computer with a fast processor, and lots of memory and disk space is always a good idea.

If Wireshark runs out of memory it will crash. See https://gitlab.com/wireshark/wireshark/wikis/KnownBugs/OutOfMemory for details and workarounds.

Although Wireshark uses a separate process to capture packets, the packet analysis is single-threaded and won't benefit much from multi-core systems.

Microsoft Windows

Wireshark should support any version of Windows that is still within its extended support lifetime. At the time of writing this includes Windows 10, 8.1, Server 2019, Server 2016, Server 2012 R2, and Server 2012. It also requires the following:

- The Universal C Runtime. This is included with Windows 10 and Windows Server 2019 and is installed automatically on earlier versions if Microsoft Windows Update is enabled. Otherwise you must install KB2999226 or KB3118401.
- Any modern 64-bit AMD64/x86-64 or 32-bit x86 processor.
- 500 MB available RAM. Larger capture files require more RAM.
- 500 MB available disk space. Capture files require additional disk space.
- Any modern display. 1280 × 1024 or higher resolution is recommended. Wireshark will make use of HiDPI or Retina resolutions if available. Power users will find multiple monitors useful.
- A supported network card for capturing
    - Ethernet. Any card supported by Windows should work. See the wiki pages on Ethernet capture and offloading for issues that may affect your environment.
    - 802.11. See the Wireshark wiki page. Capturing raw 802.11 information may be difficult without special equipment.
    - Other media. See https://gitlab.com/wireshark/wireshark/wikis/CaptureSetup/NetworkMedia.

Older versions of Windows which are outside Microsoft's extended lifecycle support window are no longer supported. It is often difficult or impossible to support these systems due to circumstances beyond our control, such as third party libraries on which we depend or due to necessary features that are only present in newer versions of Windows such as hardened security or memory management.

- Wireshark 3.6 was the last release branch to officially support 32-bit Windows.
- Wireshark 3.2 was the last release branch to officially support Windows 7 and Windows Server 2008 R2.
- Wireshark 2.2 was the last release branch to support Windows Vista and Windows Server 2008 sans R2
- Wireshark 1.12 was the last release branch to support Windows Server 2003.
- Wireshark 1.10 was the last release branch to officially support Windows XP.

See the Wireshark release lifecycle page for more details.

macOS

Wireshark supports macOS 10.14 and later. Similar to Windows, supported macOS versions depend on third party libraries and on Apple's requirements. Apple Silicon hardware is supported natively starting with version 4.0

- Wireshark 3.6 was the last release branch to support macOS 10.13.
- Wireshark 3.4 was the last release branch to support macOS 10.12.
- Wireshark 2.6 was the last release branch to support Mac OS X 10.6 and 10.7 and OS X 10.8 to 10.11.
- Wireshark 2.0 was the last release branch to support OS X on 32-bit Intel.
- Wireshark 1.8 was the last release branch to support Mac OS X on PowerPC.

The system requirements should be comparable to the specifications listed above for Windows.

UNIX, Linux, and BSD

Wireshark runs on most UNIX and UNIX-like platforms including Linux and most BSD variants. The system requirements should be comparable to the specifications listed above for Windows.

Binary packages are available for most Unices and Linux distributions including the following platforms:

- Alpine Linux
- Arch Linux
- Canonical Ubuntu
- Debian GNU/Linux
- FreeBSD
- Gentoo Linux
- HP-UX
- NetBSD
- OpenPKG
- Oracle Solaris
- Red Hat Enterprise Linux / CentOS / Fedora

If a binary package is not available for your platform you can download the source and try to build it. Please report your experiences to wireshark-dev[AT]wireshark.org.

Where To Get Wireshark

You can get the latest copy of the program from the Wireshark website at https://www.wireshark.org/download.html. The download page should automatically highlight the appropriate download for your platform and direct you to the nearest mirror. Official Windows and macOS installers are signed using trusted certificates on those platforms. macOS installers are additionally notarized.

A new Wireshark version typically becomes available every six weeks.

If you want to be notified about new Wireshark releases you should subscribe to the wireshark-announce mailing list. You will find more details in Mailing Lists.

Each release includes a list of file hashes which are sent to the wireshark-announce mailing list and placed in a file named SIGNATURES-*x.y.z*.txt. Announcement messages are archived at https://www.wireshark.org/lists/wireshark-announce/ and SIGNATURES files can be found at https://www.wireshark.org/download/src/all-versions/. Both are GPG-signed and include verification instructions for Windows, Linux, and macOS. As noted above, you can also verify downloads on Windows and macOS using the code signature validation features on those systems.

A Brief History Of Wireshark

In late 1997 Gerald Combs needed a tool for tracking down network problems and wanted to learn more about networking so he started writing Ethereal (the original name of the Wireshark project) as a way to solve both problems.

Ethereal was initially released after several pauses in development in July 1998 as version 0.2.0. Within days patches, bug reports, and words of encouragement started arriving and Ethereal was on its way to success.

Not long after that Gilbert Ramirez saw its potential and contributed a low-level dissector to it.

In October, 1998 Guy Harris was looking for something better than tcpview so he started applying patches and contributing dissectors to Ethereal.

In late 1998 Richard Sharpe, who was giving TCP/IP courses, saw its potential on such courses and started looking at it to see if it supported the protocols he needed. While it didn't at that point new protocols could be easily added. So he started contributing dissectors and contributing patches.

The list of people who have contributed to the project has become very long since then, and almost all of them started with a protocol that they needed that Wireshark did not already handle. So they copied an existing dissector and contributed the code back to the team.

In 2006 the project moved house and re-emerged under a new name: Wireshark.

In 2008, after ten years of development, Wireshark finally arrived at version 1.0. This release was

the first deemed complete, with the minimum features implemented. Its release coincided with the first Wireshark Developer and User Conference, called Sharkfest.

In 2015 Wireshark 2.0 was released, which featured a new user interface.

Development And Maintenance Of Wireshark

Wireshark was initially developed by Gerald Combs. Ongoing development and maintenance of Wireshark is handled by the Wireshark team, a loose group of individuals who fix bugs and provide new functionality.

There have also been a large number of people who have contributed protocol dissectors to Wireshark, and it is expected that this will continue. You can find a list of the people who have contributed code to Wireshark by checking the about dialog box of Wireshark, or at the authors page on the Wireshark web site.

Wireshark is an open source software project, and is released under the GNU General Public License (GPL) version 2. All source code is freely available under the GPL. You are welcome to modify Wireshark to suit your own needs, and it would be appreciated if you contribute your improvements back to the Wireshark team.

You gain three benefits by contributing your improvements back to the community:

1. Other people who find your contributions useful will appreciate them, and you will know that you have helped people in the same way that the developers of Wireshark have helped you.
2. The developers of Wireshark can further improve your changes or implement additional features on top of your code, which may also benefit you.
3. The maintainers and developers of Wireshark will maintain your code, fixing it when API changes or other changes are made, and generally keeping it in tune with what is happening with Wireshark. So when Wireshark is updated (which is often), you can get a new Wireshark version from the website and your changes will already be included without any additional effort from you.

The Wireshark source code and binary kits for some platforms are all available on the download page of the Wireshark website: https://www.wireshark.org/download.html.

Reporting Problems And Getting Help

If you have problems or need help with Wireshark there are several places that may be of interest (besides this guide, of course).

Website

You will find lots of useful information on the Wireshark homepage at https://www.wireshark.org/.

Wiki

The Wireshark Wiki at https://gitlab.com/wireshark/wireshark/wikis/ provides a wide range of information related to Wireshark and packet capture in general. You will find a lot of information not part of this user's guide. For example, it contains an explanation how to capture on a switched network, an ongoing effort to build a protocol reference, protocol-specific information, and much more.

And best of all, if you would like to contribute your knowledge on a specific topic (maybe a network protocol you know well), you can edit the wiki pages with your web browser.

Q&A Site

The Wireshark Q&A site at https://ask.wireshark.org/ offers a resource where questions and answers come together. You can search for questions asked before and see what answers were given by people who knew about the issue. Answers are ranked, so you can easily pick out the best ones. If your question hasn't been discussed before you can post one yourself.

FAQ

The Frequently Asked Questions lists often asked questions and their corresponding answers.

NOTE	*Read the FAQ* Before sending any mail to the mailing lists below, be sure to read the FAQ. It will often answer any questions you might have. This will save yourself and others a lot of time. Keep in mind that a lot of people are subscribed to the mailing lists.

You will find the FAQ inside Wireshark by clicking the menu item Help/Contents and selecting the FAQ page in the dialog shown.

An online version is available at the Wireshark website at https://www.wireshark.org/faq.html. You might prefer this online version, as it's typically more up to date and the HTML format is easier to use.

Mailing Lists

There are several mailing lists of specific Wireshark topics available:

wireshark-announce

Information about new program releases, which usually appear about every six weeks.

wireshark-users

Topics of interest to users of Wireshark. People typically post questions about using Wireshark and others (hopefully) provide answers.

wireshark-dev

Topics of interest to developers of Wireshark. If you want to develop a protocol dissector or update the user interface, join this list.

You can subscribe to each of these lists from the Wireshark web site: https://www.wireshark.org/lists/. From there, you can choose which mailing list you want to subscribe to by clicking on the Subscribe/Unsubscribe/Options button under the title of the relevant list. The links to the archives are included on that page as well.

TIP

The lists are archived

You can search in the list archives to see if someone asked the same question some time before and maybe already got an answer. That way you don't have to wait until someone answers your question.

Reporting Problems

NOTE

Before reporting any problems, please make sure you have installed the latest version of Wireshark.

When reporting problems with Wireshark please supply the following information:

1. The version number of Wireshark and the dependent libraries linked with it, such as Qt or GLib. You can obtain this from Wireshark's about box or the command *wireshark -v*.
2. Information about the platform you run Wireshark on (Windows, Linux, etc. and 32-bit, 64-bit, etc.).
3. A detailed description of your problem.
4. If you get an error/warning message, copy the text of that message (and also a few lines before and after it, if there are some) so others may find the place where things go wrong. Please don't give something like: "I get a warning while doing x" as this won't give a good idea where to look.

WARNING

Don't send confidential information!

If you send capture files to the mailing lists be sure they don't contain any sensitive or confidential information like passwords or personally identifiable information (PII).

In many cases you can use a tool like TraceWrangler to sanitize a capture file before sharing it.

NOTE

Don't send large files

Do not send large files (> 1 MB) to the mailing lists. Instead, provide a download link. For bugs and feature requests, you can create an issue on Gitlab Issues and upload the file there.

Reporting Crashes on UNIX/Linux platforms

When reporting crashes with Wireshark it is helpful if you supply the traceback information along with the information mentioned in “Reporting Problems”.

You can obtain this traceback information with the following commands on UNIX or Linux (note the backticks):

```
$ gdb `whereis wireshark | cut -f2 -d: | cut -d' ' -f2` core >& backtrace.txt
backtrace
^D
```

If you do not have *gdb* available, you will have to check out your operating system’s debugger.

Email *backtrace.txt* to wireshark-dev[AT]wireshark.org.

Reporting Crashes on Windows platforms

The Windows distributions don’t contain the symbol files (.pdb) because they are very large. You can download them separately at https://www.wireshark.org/download/win32/all-versions/ and https://www.wireshark.org/download/win64/all-versions/ .

Building and Installing Wireshark

Introduction

As with all things there must be a beginning and so it is with Wireshark. To use Wireshark you must first install it. If you are running Windows or macOS you can download an official release at https://www.wireshark.org/download.html, install it, and skip the rest of this chapter.

If you are running another operating system such as Linux or FreeBSD you might want to install from source. Several Linux distributions offer Wireshark packages but they commonly provide out-of-date versions. No other versions of UNIX ship Wireshark so far. For that reason, you will need to know where to get the latest version of Wireshark and how to install it.

This chapter shows you how to obtain source and binary packages and how to build Wireshark from source should you choose to do so.

The general steps are the following:

1. Download the relevant package for your needs, e.g., source or binary distribution.
2. For source distributions, compile the source into a binary. This may involve building and/or installing other necessary packages.
3. Install the binaries into their final destinations.

Obtaining the source and binary distributions

You can obtain both source and binary distributions from the Wireshark main page or the download page at https://www.wireshark.org/download.html. Select the package most appropriate for your system.

Installing Wireshark under Windows

The official Windows packages can be downloaded from the Wireshark main page or the download page. Installer names contain the platform and version. For example, Wireshark-win64-4.1.0.exe installs Wireshark 4.1.0 for 64-bit Windows. The Wireshark installer includes Npcap which is required for packet capture. Windows packages automatically update. See Updating Wireshark for details.

Simply download the Wireshark installer from https://www.wireshark.org/download.html and execute it. Official packages are signed by **Sysdig, Inc.**. You can choose to install several optional components and select the location of the installed package. The default settings are recommended for most users.

Installation Components

On the *Choose Components* page of the installer you can select from the following:

- **Wireshark** - The network protocol analyzer that we all know and mostly love.
- **TShark** - A command-line network protocol analyzer. If you haven't tried it you should.
- **Plugins & Extensions** - Extras for the Wireshark and TShark dissection engines
  - **Dissector Plugins** - Plugins with some extended dissections.
  - **Tree Statistics Plugins** - Extended statistics.
  - **Mate - Meta Analysis and Tracing Engine** - User configurable extension(s) of the display filter engine, see MATE for details.
  - **SNMP MIBs** - SNMP MIBs for a more detailed SNMP dissection.
- **Tools** - Additional command line tools to work with capture files
  - **Editcap** - Reads a capture file and writes some or all of the packets into another capture file.
  - **Text2Pcap** - Reads in an ASCII hex dump and writes the data into a pcap capture file.
  - **Reordercap** - Reorders a capture file by timestamp.
  - **Mergecap** - Combines multiple saved capture files into a single output file.
  - **Capinfos** - Provides information on capture files.
  - **Rawshark** - Raw packet filter.
- **User's Guide** - Local installation of the User's Guide. The Help buttons on most dialogs will require an internet connection to show help pages if the User's Guide is not installed locally.

Additional Tasks

- **Start Menu Shortcuts** - Add some start menu shortcuts.
- **Desktop Icon** - Add a Wireshark icon to the desktop.
- **Quick Launch Icon** - add a Wireshark icon to the Explorer quick launch toolbar.
- **Associate file extensions to Wireshark** - Associate standard network trace files to Wireshark.

Install Location

By default Wireshark installs into `%ProgramFiles%\Wireshark` on 32-bit Windows and `%ProgramFiles64%\Wireshark` on 64-bit Windows. This expands to `C:\Program Files\Wireshark` on most systems.

Installing Npcap

The Wireshark installer contains the latest Npcap installer.

If you don't have Npcap installed you won't be able to capture live network traffic but you will still be able to open saved capture files. By default the latest version of Npcap will be installed. If you don't wish to do this or if you wish to reinstall Npcap you can check the *Install Npcap* box as needed.

For more information about Npcap see https://npcap.com/ and https://gitlab.com/wireshark/wireshark/wikis/Npcap.

Windows installer command line options

For special cases, there are some command line parameters available:

- `/S` runs the installer or uninstaller silently with default values. The silent installer **will not** install Npcap.
- `/desktopicon` installation of the desktop icon, `=yes` - force installation, `=no` - don't install, otherwise use default settings. This option can be useful for a silent installer.
- `/quicklaunchicon` installation of the quick launch icon, `=yes` - force installation, `=no` - don't install, otherwise use default settings.
- `/D` sets the default installation directory ($INSTDIR), overriding InstallDir and InstallDirRegKey. It must be the last parameter used in the command line and must not contain any quotes even if the path contains spaces.
- `/NCRC` disables the CRC check. We recommend against using this flag.
- `/EXTRACOMPONENTS` comma separated list of optional components to install. The following extcap binaries are supported.
  - `androiddump` - Provide interfaces to capture from Android devices
  - `ciscodump` - Provide interfaces to capture from a remote Cisco router through SSH
  - `randpktdump` - Provide an interface to generate random captures using randpkt
  - `sshdump` - Provide interfaces to capture from a remote host through SSH using a remote capture binary
  - `udpdump` - Provide a UDP receiver that gets packets from network devices

Example:

```
> Wireshark-win64-wireshark-2.0.5.exe /NCRC /S /desktopicon=yes /quicklaunchicon=no
/D=C:\Program Files\Foo

> Wireshark-win64-3.3.0.exe /S /EXTRACOMPONENTS=sshdump,udpdump
```

Running the installer without any parameters shows the normal interactive installer.

Manual Npcap Installation

As mentioned above, the Wireshark installer also installs Npcap. If you prefer to install Npcap manually or want to use a different version than the one included in the Wireshark installer, you can download Npcap from the main Npcap site at https://npcap.com/.

Update Npcap

Wireshark updates may also include a new version of Npcap. Manual Npcap updates instructions can be found on the Npcap web site at https://npcap.com/. You may have to reboot your machine after installing a new Npcap version.

Uninstall Wireshark

You can uninstall Wireshark using the *Programs and Features* control panel. Select the "Wireshark" entry to start the uninstallation procedure.

The Wireshark uninstaller provides several options for removal. The default is to remove the core components but keep your personal settings and Npcap. Npcap is kept in case other programs need it.

Uninstall Npcap

You can uninstall Npcap independently of Wireshark using the *Npcap* entry in the *Programs and Features* control panel. Remember that if you uninstall Npcap you won't be able to capture anything with Wireshark.

Building from source under Windows

We strongly recommended using the binary installer for Windows unless you want to start developing Wireshark on the Windows platform.

For further information how to obtain sources and build Wireshark for Windows from the sources see the Developer's Guide at:

- https://www.wireshark.org/docs/wsdg_html_chunked/ChSrcObtain
- https://www.wireshark.org/docs/wsdg_html_chunked/ChSetupWindows

You may also want to have a look at the Development Wiki (https://gitlab.com/wireshark/wireshark/wikis/Development) for the latest available development documentation.

Installing Wireshark under macOS

The official macOS packages can be downloaded from the Wireshark main page or the download page. Packages are distributed as disk images (.dmg) containing the application bundle. Package

names contain the platform and version. To install Wireshark simply open the disk image and drag *Wireshark* to your */Applications* folder. macOS packages automatically update. See Updating Wireshark for details.

In order to capture packets, you must install the "ChmodBPF" launch daemon. You can do so by opening the *Install ChmodBPF.pkg* file in the Wireshark .dmg or from Wireshark itself by opening **Wireshark › About Wireshark** selecting the "Folders" tab, and double-clicking "macOS Extras".

The installer package includes Wireshark along with ChmodBPF and system path packages. See the included *Read me first.html* file for more details.

Installing the binaries under UNIX

In general installing the binary under your version of UNIX will be specific to the installation methods used with your version of UNIX. For example, under AIX, you would use *smit* to install the Wireshark binary package, while under Tru64 UNIX (formerly Digital UNIX) you would use *setld*.

Installing from RPMs under Red Hat and alike

Building RPMs from Wireshark's source code results in several packages (most distributions follow the same system):

- The `wireshark` package contains the core Wireshark libraries and command-line tools.
- The `wireshark` or `wireshark-qt` package contains the Qt-based GUI.

Many distributions use `yum` or a similar package management tool to make installation of software (including its dependencies) easier. If your distribution uses `yum`, use the following command to install Wireshark together with the Qt GUI:

```
yum install wireshark wireshark-qt
```

If you've built your own RPMs from the Wireshark sources you can install them by running, for example:

```
rpm -ivh wireshark-2.0.0-1.x86_64.rpm wireshark-qt-2.0.0-1.x86_64.rpm
```

If the above command fails because of missing dependencies, install the dependencies first, and then retry the step above.

Installing from debs under Debian, Ubuntu and other Debian derivatives

If you can just install from the repository then use

```
apt install wireshark
```

Apt should take care of all of the dependency issues for you.

NOTE	*Capturing requires privileges* By installing Wireshark packages non-root, users won't gain rights automatically to capture packets. To allow non-root users to capture packets follow the procedure described in https://gitlab.com/wireshark/wireshark/blob/master/packaging/debian/README.Debian (/usr/share/doc/wireshark-common/README.Debian.gz)

Installing from portage under Gentoo Linux

Use the following command to install Wireshark under Gentoo Linux with all of the extra features:

```
USE="c-ares ipv6 snmp ssl kerberos threads selinux" emerge wireshark
```

Installing from packages under FreeBSD

Use the following command to install Wireshark under FreeBSD:

```
pkg_add -r wireshark
```

pkg_add should take care of all of the dependency issues for you.

Building from source under UNIX or Linux

We recommended using the binary installer for your platform unless you want to start developing Wireshark.

Building Wireshark requires the proper build environment including a compiler and many supporting libraries. For more information, see the Developer's Guide at:

- https://www.wireshark.org/docs/wsdg_html_chunked/ChSrcObtain
- https://www.wireshark.org/docs/wsdg_html_chunked/ChapterSetup#ChSetupUNIX

Updating Wireshark

By default, Wireshark on Windows and macOS will check for new versions and notify you when they are available. If you have the *Check for updates* preference disabled or if you run Wireshark in an isolated environment you should subscribe to the *wireshark-announce* mailing list to be notified of new versions. See Mailing Lists for details on subscribing to this list.

New versions of Wireshark are usually released every four to six weeks. Updating Wireshark is done the same way as installing it. Simply download and run the installer on Windows, or download and drag the application on macOS. A reboot is usually not required and all your personal settings will remain unchanged.

We offer two update channels, *Stable* and *Development*. The Stable channel is the default, and only installs packages from stable (even-numbered) release branches. The Development channel installs development and release candidate packages when they are available, and stable releases otherwise. To configure your release channel, go to **Preferences › Advanced** and search for "update.channel". See Preferences for details.

User Interface

Introduction

By now you have installed Wireshark and are likely keen to get started capturing your first packets. In the next chapters we will explore:

- How the Wireshark user interface works
- How to capture packets in Wireshark
- How to view packets in Wireshark
- How to filter packets in Wireshark
- ... and many other things!

Start Wireshark

You can start Wireshark from your shell or window manager.

> **TIP**
>
> *Power user tip*
>
> When starting Wireshark it's possible to specify optional settings using the command line. See Start Wireshark from the command line for details.

The following chapters contain many screenshots of Wireshark. As Wireshark runs on many different platforms with many different window managers, different styles applied and there are different versions of the underlying GUI toolkit used, your screen might look different from the provided screenshots. But as there are no real differences in functionality these screenshots should still be well understandable.

The Main window

Let's look at Wireshark's user interface. The Main window shows Wireshark as you would usually see it after some packets are captured or loaded (how to do this will be described later).

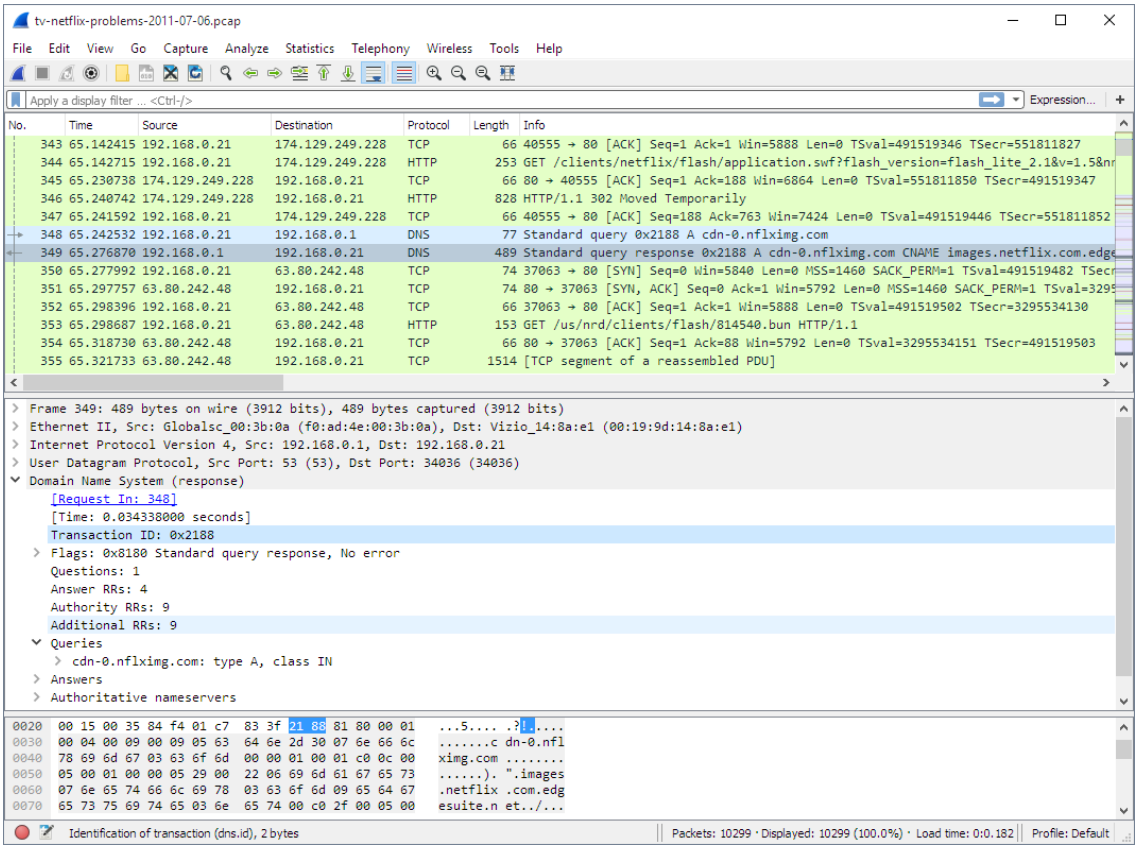


Figure 2. The Main window

Wireshark's main window consists of parts that are commonly known from many other GUI programs.

1. The *menu* (see The Menu) is used to start actions.
2. The *main toolbar* (see The "Main" Toolbar) provides quick access to frequently used items from the menu.
3. The *filter toolbar* (see The "Filter" Toolbar) allows users to set *display filters* to filter which packets are displayed (see Filtering Packets While Viewing).
4. The *packet list pane* (see The "Packet List" Pane) displays a summary of each packet captured. By clicking on packets in this pane you control what is displayed in the other two panes.
5. The *packet details pane* (see The "Packet Details" Pane) displays the packet selected in the packet list pane in more detail.
6. The *packet bytes pane* (see The "Packet Bytes" Pane) displays the data from the packet selected in the packet list pane, and highlights the field selected in the packet details pane.
7. The *packet diagram pane* (see The "Packet Diagram" Pane) displays the packet selected in the packet list as a textbook-style diagram.
8. The *statusbar* (see The Statusbar) shows some detailed information about the current program state and the captured data.

TIP	The layout of the main window can be customized by changing preference settings. See Preferences for details.

Main Window Navigation

Packet list and detail navigation can be done entirely from the keyboard. Keyboard Navigation shows a list of keystrokes that will let you quickly move around a capture file. See Go menu items for additional navigation keystrokes.

Table 2. Keyboard Navigation

Accelerator	Description
`Tab` or `Shift` + `Tab`	Move between screen elements, e.g., from the toolbars to the packet list to the packet detail.
`▯`	Move to the next packet or detail item.
`▯`	Move to the previous packet or detail item.
`Ctrl` + `▯` or `F8`	Move to the next packet, even if the packet list isn't focused.
`Ctrl` + `▯` or `F7`	Move to the previous packet, even if the packet list isn't focused.
`Ctrl` + `.`	Move to the next packet of the conversation (TCP, UDP or IP).
`Ctrl` + `,`	Move to the previous packet of the conversation (TCP, UDP or IP).
`Alt` + `→` or `Option` + `→` (macOS)	Move to the next packet in the selection history.
`Alt` + `←` or `Option` + `←` (macOS)	Move to the previous packet in the selection history.
`←`	In the packet detail, closes the selected tree item. If it's already closed, jumps to the parent node.
`→`	In the packet detail, opens the selected tree item.
`Shift` + `→`	In the packet detail, opens the selected tree item and all of its subtrees.
`Ctrl` + `→`	In the packet detail, opens all tree items.
`Ctrl` + `←`	In the packet detail, closes all tree items.
`Backspace`	In the packet detail, jumps to the parent node.
`Return` or `Enter`	In the packet detail, toggles the selected tree item.

Help › About Wireshark › Keyboard Shortcuts will show a list of all shortcuts in the main window. Additionally, typing anywhere in the main window will start filling in a display filter.

The Menu

Wireshark's main menu is located either at the top of the main window (Windows, Linux) or at the top of your main screen (macOS). An example is shown in The Menu.

NOTE	Some menu items will be disabled (greyed out) if the corresponding feature isn't available. For example, you cannot save a capture file if you haven't captured or loaded any packets.

Figure 3. The Menu

The main menu contains the following items:

File

This menu contains items to open and merge capture files, save, print, or export capture files in whole or in part, and to quit the Wireshark application. See The "File" Menu.

Edit

This menu contains items to find a packet, time reference or mark one or more packets, handle configuration profiles, and set your preferences; (cut, copy, and paste are not presently implemented). See The "Edit" Menu.

View

This menu controls the display of the captured data, including colorization of packets, zooming the font, showing a packet in a separate window, expanding and collapsing trees in packet details, …. See The "View" Menu.

Go

This menu contains items to go to a specific packet. See The "Go" Menu.

Capture

This menu allows you to start and stop captures and to edit capture filters. See The "Capture" Menu.

Analyze

This menu contains items to manipulate display filters, enable or disable the dissection of protocols, configure user specified decodes and follow a TCP stream. See The "Analyze" Menu.

Statistics

This menu contains items to display various statistic windows, including a summary of the packets that have been captured, display protocol hierarchy statistics and much more. See The "Statistics" Menu.

Telephony

This menu contains items to display various telephony related statistic windows, including a media analysis, flow diagrams, display protocol hierarchy statistics and much more. See The “Telephony” Menu.

Wireless

This menu contains items to display Bluetooth and IEEE 802.11 wireless statistics.

Tools

This menu contains various tools available in Wireshark, such as creating Firewall ACL Rules. See The “Tools” Menu.

Help

This menu contains items to help the user, e.g., access to some basic help, manual pages of the various command line tools, online access to some of the webpages, and the usual about dialog. See The “Help” Menu.

Each of these menu items is described in more detail in the sections that follow.

TIP

Shortcuts make life easier

Most common menu items have keyboard shortcuts. For example, you can press the Control and the K keys together to open the “Capture Options” dialog.

The “File” Menu

The Wireshark file menu contains the fields shown in File menu items.

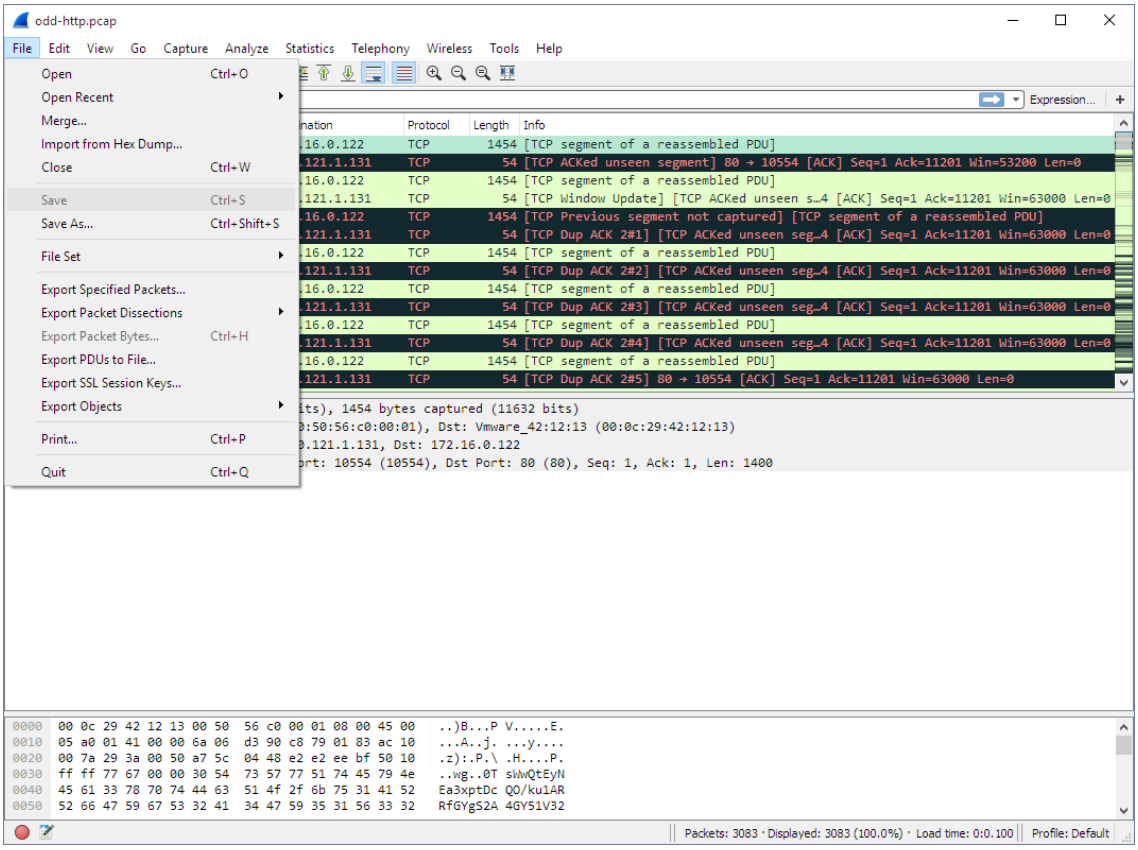


Figure 4. The “File” Menu

Table 3. File menu items

Menu Item	Accelerator	Description
Open…	`Ctrl` + `O`	This shows the file open dialog box that allows you to load a capture file for viewing. It is discussed in more detail in The “Open Capture File” Dialog Box.
Open Recent		This lets you open recently opened capture files. Clicking on one of the submenu items will open the corresponding capture file directly.
Merge…		This menu item lets you merge a capture file into the currently loaded one. It is discussed in more detail in Merging Capture Files.
Import from Hex Dump…		This menu item brings up the import file dialog box that allows you to import a text file containing a hex dump into a new temporary capture. It is discussed in more detail in Import Hex Dump.

Menu Item	Accelerator	Description
Close	`Ctrl` + `W`	This menu item closes the current capture. If you haven't saved the capture, you will be asked to do so first (this can be disabled by a preference setting).
Save	`Ctrl` + `S`	This menu item saves the current capture. If you have not set a default capture file name (perhaps with the -w <capfile> option), Wireshark pops up the Save Capture File As dialog box (which is discussed further in The "Save Capture File As" Dialog Box). If you have already saved the current capture, this menu item will be greyed out. You cannot save a live capture while the capture is in progress. You must stop the capture in order to save.
Save As…	`Shift` + `Ctrl` + `S`	This menu item allows you to save the current capture file to whatever file you would like. It pops up the Save Capture File As dialog box (which is discussed further in The "Save Capture File As" Dialog Box).
File Set › List Files		This menu item allows you to show a list of files in a file set. It pops up the Wireshark List File Set dialog box (which is discussed further in File Sets).
File Set › Next File		If the currently loaded file is part of a file set, jump to the next file in the set. If it isn't part of a file set or just the last file in that set, this item is greyed out.
File Set › Previous File		If the currently loaded file is part of a file set, jump to the previous file in the set. If it isn't part of a file set or just the first file in that set, this item is greyed out.
Export Specified Packets…		This menu item allows you to export all (or some) of the packets in the capture file to file. It pops up the Wireshark Export dialog box (which is discussed further in Exporting Data).

Menu Item	Accelerator	Description
Export Packet Dissections...	Ctrl + H	These menu items allow you to export the currently selected bytes in the packet bytes pane to a text file in a number of formats including plain, CSV, and XML. It is discussed further in The “Export Selected Packet Bytes” Dialog Box.
Export Objects		These menu items allow you to export captured DICOM, HTTP, IMF, SMB, or TFTP objects into local files. It pops up a corresponding object list (which is discussed further in The “Export Objects” Dialog Box)
Print...	Ctrl + P	This menu item allows you to print all (or some) of the packets in the capture file. It pops up the Wireshark Print dialog box (which is discussed further in Printing Packets).
Quit	Ctrl + Q	This menu item allows you to quit from Wireshark. Wireshark will ask to save your capture file if you haven’t previously saved it (this can be disabled by a preference setting).

The “Edit” Menu

The Wireshark Edit menu contains the fields shown in Edit menu items.

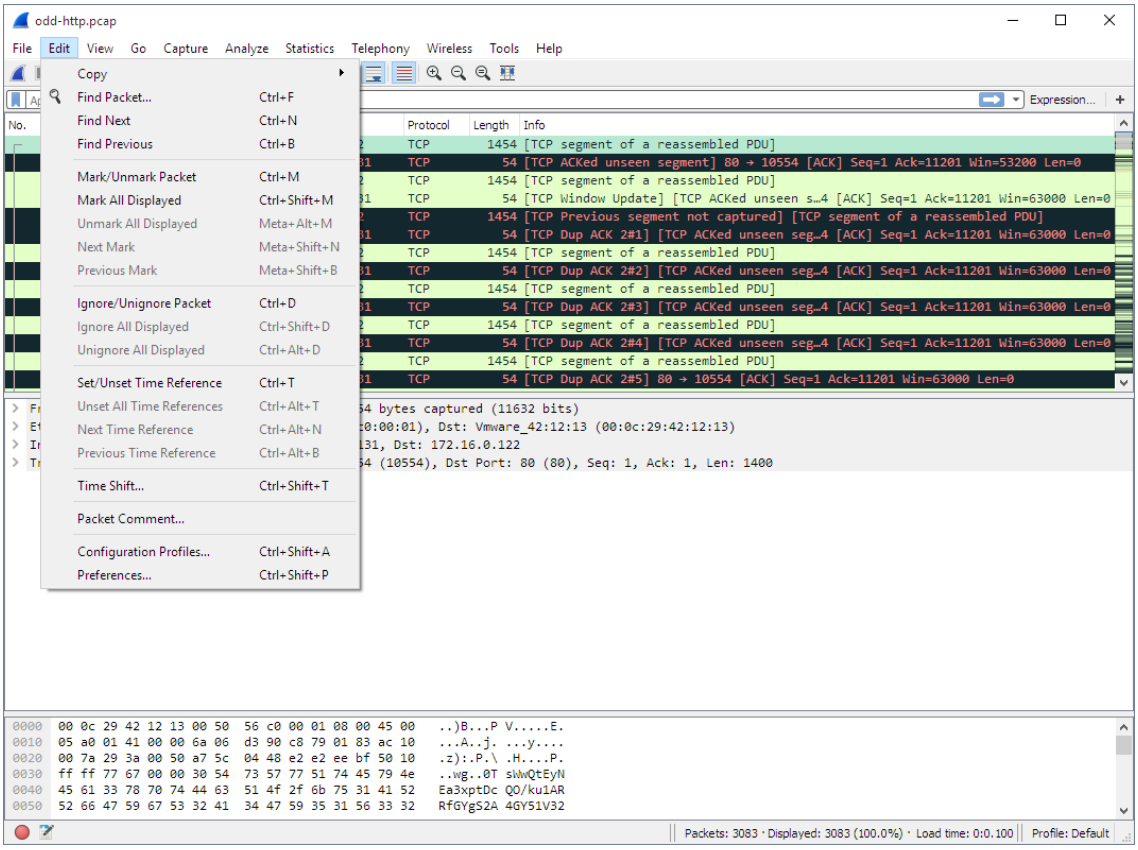


Figure 5. The “Edit” Menu

Table 4. Edit menu items

Menu Item	Accelerator	Description
Copy		These menu items will copy the packet list, packet detail, or properties of the currently selected packet to the clipboard.
Find Packet...	`Ctrl` + `F`	This menu item brings up a toolbar that allows you to find a packet by many criteria. There is further information on finding packets in Finding Packets.
Find Next	`Ctrl` + `N`	This menu item tries to find the next packet matching the settings from “Find Packet...”.
Find Previous	`Ctrl` + `B`	This menu item tries to find the previous packet matching the settings from “Find Packet...”.
Mark/Unmark Packet	`Ctrl` + `M`	This menu item marks the currently selected packet. See Marking Packets for details.
Mark All Displayed Packets	`Ctrl` + `Shift` + `M`	This menu item marks all displayed packets.
Unmark All Displayed Packets	`Ctrl` + `Alt` + `M`	This menu item unmarks all displayed packets.
Next Mark	`Ctrl` + `Shift` + `N`	Find the next marked packet.

Menu Item	Accelerator	Description
Previous Mark	Ctrl + Shift + B	Find the previous marked packet.
Ignore/Unignore Packet	Ctrl + D	This menu item marks the currently selected packet as ignored. See Ignoring Packets for details.
Ignore All Displayed	Ctrl + Shift + D	This menu item marks all displayed packets as ignored.
Unignore All Displayed	Ctrl + Alt + D	This menu item unmarks all ignored packets.
Set/Unset Time Reference	Ctrl + T	This menu item set a time reference on the currently selected packet. See Packet Time Referencing for more information about the time referenced packets.
Unset All Time References	Ctrl + Alt + T	This menu item removes all time references on the packets.
Next Time Reference	Ctrl + Alt + N	This menu item tries to find the next time referenced packet.
Previous Time Reference	Ctrl + Alt + B	This menu item tries to find the previous time referenced packet.
Time Shift...	Ctrl + Shift + T	Opens the “Time Shift” dialog, which allows you to adjust the timestamps of some or all packets.
Packet Comment...	Ctrl + Alt + C	Opens the “Packet Comment” dialog, which lets you add a comment to a single packet. Note that the ability to save packet comments depends on your file format. E.g., pcapng supports comments, pcap does not.
Delete All Packet Comments		This will delete all comments from all packets. Note that the ability to save capture comments depends on your file format. E.g., pcapng supports comments, pcap does not.
Configuration Profiles...	Ctrl + Shift + A	This menu item brings up a dialog box for handling configuration profiles. More detail is provided in Configuration Profiles.
Preferences...	Ctrl + Shift + P or Cmd + , (macOS)	This menu item brings up a dialog box that allows you to set preferences for many parameters that control Wireshark. You can also save your preferences so Wireshark will use them the next time you start it. More detail is provided in Preferences.

The “View” Menu

The Wireshark View menu contains the fields shown in View menu items.

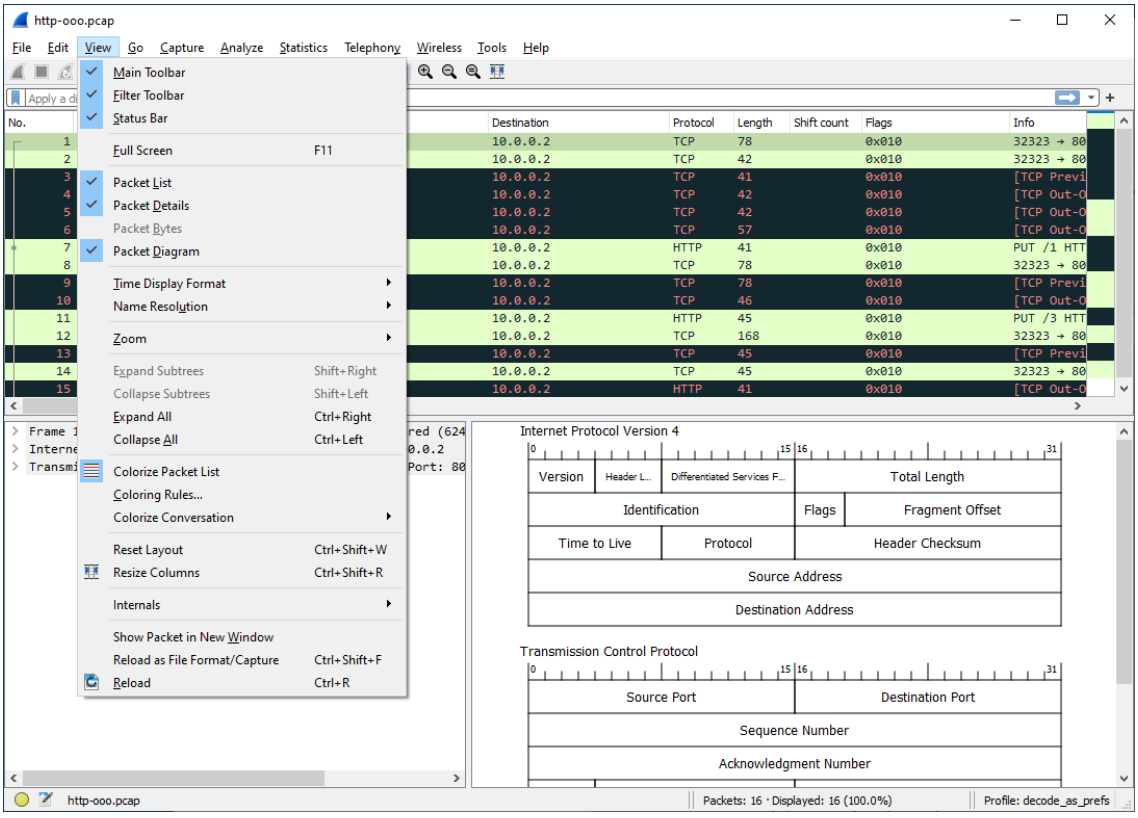


Figure 6. The “View” Menu

Table 5. View menu items

Menu Item	Accelerator	Description
Main Toolbar		This menu item hides or shows the main toolbar, see The “Main” Toolbar.
Filter Toolbar		This menu item hides or shows the filter toolbar, see The “Filter” Toolbar.
Wireless Toolbar		This menu item hides or shows the wireless toolbar. May not be present on some platforms.
Statusbar		This menu item hides or shows the statusbar, see The Statusbar.
Packet List		This menu item hides or shows the packet list pane, see The “Packet List” Pane.
Packet Details		This menu item hides or shows the packet details pane, see The “Packet Details” Pane.
Packet Bytes		This menu item hides or shows the packet bytes pane, see The “Packet Bytes” Pane.

Menu Item	Accelerator	Description
Packet Diagram		This menu item hides or shows the packet diagram pane. See The “Packet Diagram” Pane.
Time Display Format › Date and Time of Day: 1970-01-01 01:02:03.123456		Selecting this tells Wireshark to display the time stamps in date and time of day format, see Time Display Formats And Time References. The fields “Time of Day”, “Date and Time of Day”, “Seconds Since First Captured Packet”, “Seconds Since Previous Captured Packet” and “Seconds Since Previous Displayed Packet” are mutually exclusive.
Time Display Format › Time of Day: 01:02:03.123456		Selecting this tells Wireshark to display time stamps in time of day format, see Time Display Formats And Time References.
Time Display Format › Seconds Since Epoch (1970-01-01): 1234567890.123456		Selecting this tells Wireshark to display time stamps in seconds since 1970-01-01 00:00:00, see Time Display Formats And Time References.
Time Display Format › Seconds Since First Captured Packet: 123.123456		Selecting this tells Wireshark to display time stamps in seconds since first captured packet format, see Time Display Formats And Time References.
Time Display Format › Seconds Since Previous Captured Packet: 1.123456		Selecting this tells Wireshark to display time stamps in seconds since previous captured packet format, see Time Display Formats And Time References.
Time Display Format › Seconds Since Previous Displayed Packet: 1.123456		Selecting this tells Wireshark to display time stamps in seconds since previous displayed packet format, see Time Display Formats And Time References.
Time Display Format › Automatic (File Format Precision)		Selecting this tells Wireshark to display time stamps with the precision given by the capture file format used, see Time Display Formats And Time References. The fields “Automatic”, “Seconds” and “... seconds” are mutually exclusive.
Time Display Format › Seconds: 0		Selecting this tells Wireshark to display time stamps with a precision of one second, see Time Display Formats And Time References.

Menu Item	Accelerator	Description
Time Display Format › … seconds: 0….		Selecting this tells Wireshark to display time stamps with a precision of one second, decisecond, centisecond, millisecond, microsecond or nanosecond, see Time Display Formats And Time References.
Time Display Format › Display Seconds with hours and minutes		Selecting this tells Wireshark to display time stamps in seconds, with hours and minutes.
Name Resolution › Resolve Name		This item allows you to trigger a name resolve of the current packet only, see Name Resolution.
Name Resolution › Enable for MAC Layer		This item allows you to control whether or not Wireshark translates MAC addresses into names, see Name Resolution.
Name Resolution › Enable for Network Layer		This item allows you to control whether or not Wireshark translates network addresses into names, see Name Resolution.
Name Resolution › Enable for Transport Layer		This item allows you to control whether or not Wireshark translates transport addresses into names, see Name Resolution.
Colorize Packet List		This item allows you to control whether or not Wireshark should colorize the packet list. Enabling colorization will slow down the display of new packets while capturing or loading capture files.
Auto Scroll in Live Capture		This item allows you to specify that Wireshark should scroll the packet list pane as new packets come in, so you are always looking at the last packet. If you do not specify this, Wireshark simply adds new packets onto the end of the list, but does not scroll the packet list pane.
Zoom In	`Ctrl` + `+`	Zoom into the packet data (increase the font size).
Zoom Out	`Ctrl` + `-`	Zoom out of the packet data (decrease the font size).
Normal Size	`Ctrl` + `=`	Set zoom level back to 100% (set font size back to normal).

Menu Item	Accelerator	Description
Resize All Columns	Shift + Ctrl + R	Resize all column widths so the content will fit into it. Resizing may take a significant amount of time, especially if a large capture file is loaded.
Displayed Columns		This menu items folds out with a list of all configured columns. These columns can now be shown or hidden in the packet list.
Expand Subtrees	Shift + →	This menu item expands the currently selected subtree in the packet details tree.
Collapse Subtrees	Shift + ←	This menu item collapses the currently selected subtree in the packet details tree.
Expand All	Ctrl + →	Wireshark keeps a list of all the protocol subtrees that are expanded, and uses it to ensure that the correct subtrees are expanded when you display a packet. This menu item expands all subtrees in all packets in the capture.
Collapse All	Ctrl + ←	This menu item collapses the tree view of all packets in the capture list.
Colorize Conversation		This menu item brings up a submenu that allows you to color packets in the packet list pane based on the addresses of the currently selected packet. This makes it easy to distinguish packets belonging to different conversations. Packet colorization.
Colorize Conversation › Color 1-10		These menu items enable one of the ten temporary color filters based on the currently selected conversation.
Colorize Conversation › Reset coloring		This menu item clears all temporary coloring rules.
Colorize Conversation › New Coloring Rule...		This menu item opens a dialog window in which a new permanent coloring rule can be created based on the currently selected conversation.
Coloring Rules...		This menu item brings up a dialog box that allows you to color packets in the packet list pane according to filter expressions you choose. It can be very useful for spotting certain types of packets, see Packet colorization.

Menu Item	Accelerator	Description
Internals		Information about various internal data structures. See Internals menu items below for more information.
Show Packet in New Window		Shows the selected packet in a separate window. The separate window shows only the packet details and bytes. See Viewing a packet in a separate window for details.
Reload	`Ctrl` + `R`	This menu item allows you to reload the current capture file.

Table 6. Internals menu items

Menu Item	Description
Conversation Hash Tables	Shows the tuples (address and port combinations) used to identify each conversation.
Dissector Tables	Shows tables of subdissector relationships.
Supported Protocols	Displays supported protocols and protocol fields.

The “Go” Menu

The Wireshark Go menu contains the fields shown in Go menu items.

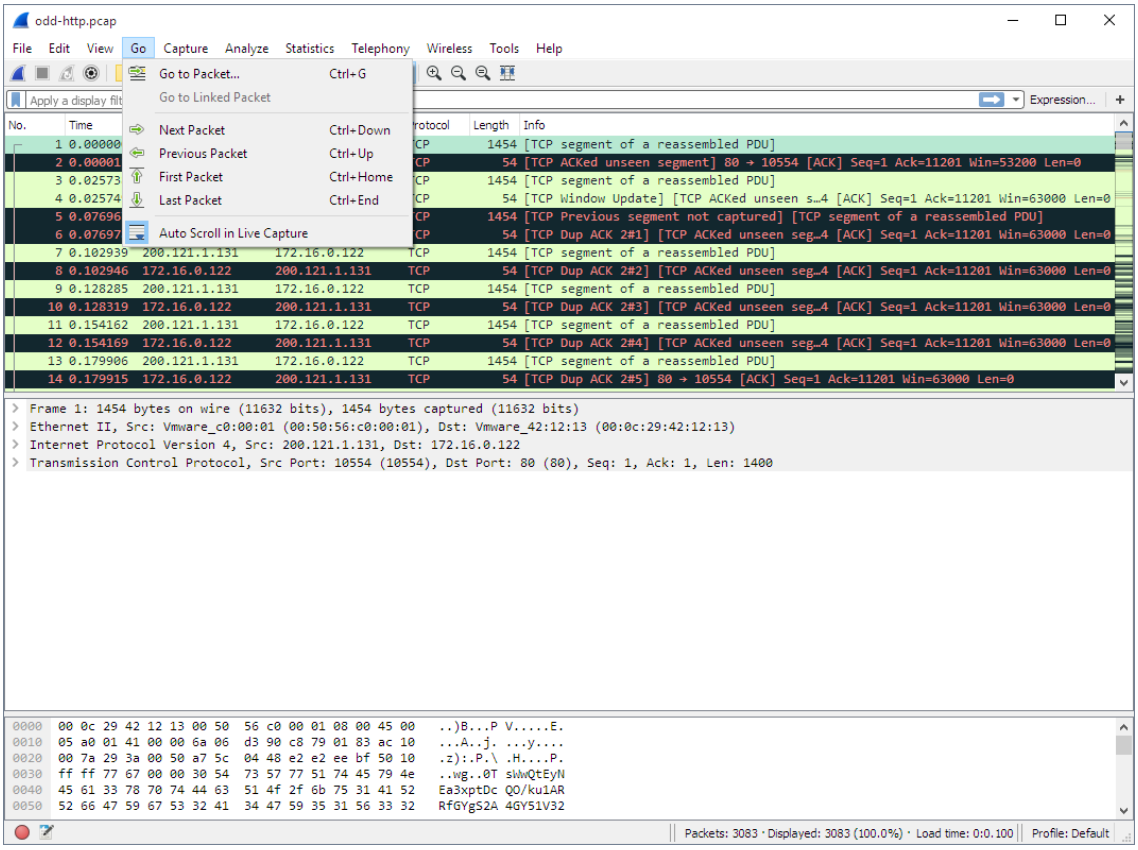


Figure 7. The "Go" Menu

Table 7. Go menu items

Menu Item	Accelerator	Description
Back	Alt + ←	Jump to the recently visited packet in the packet history, much like the page history in a web browser.
Forward	Alt + →	Jump to the next visited packet in the packet history, much like the page history in a web browser.
Go to Packet...	Ctrl + G	Bring up a window frame that allows you to specify a packet number, and then goes to that packet. See Go To A Specific Packet for details.
Go to Corresponding Packet		Go to the corresponding packet of the currently selected protocol field. If the selected field doesn't correspond to a packet, this item is greyed out.
Previous Packet	Ctrl + ▯	Move to the previous packet in the list. This can be used to move to the previous packet even if the packet list doesn't have keyboard focus.

Menu Item	Accelerator	Description
Next Packet	Ctrl + ▯	Move to the next packet in the list. This can be used to move to the previous packet even if the packet list doesn't have keyboard focus.
First Packet	Ctrl + Home	Jump to the first packet of the capture file.
Last Packet	Ctrl + End	Jump to the last packet of the capture file.
Previous Packet In Conversation	Ctrl + ,	Move to the previous packet in the current conversation. This can be used to move to the previous packet even if the packet list doesn't have keyboard focus.
Next Packet In Conversation	Ctrl + .	Move to the next packet in the current conversation. This can be used to move to the previous packet even if the packet list doesn't have keyboard focus.

The "Capture" Menu

The Wireshark Capture menu contains the fields shown in Capture menu items.

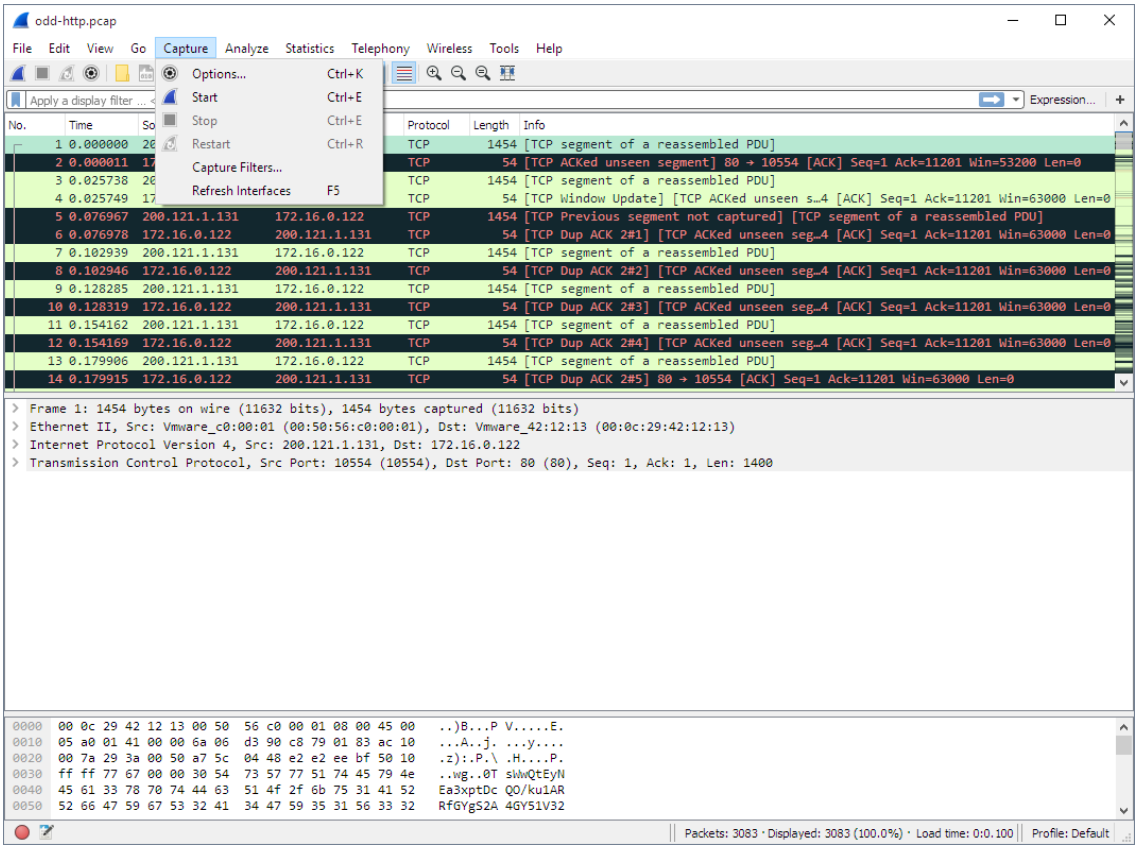


Figure 8. The "Capture" Menu

Table 8. Capture menu items

Menu Item	Accelerator	Description
Options…	Ctrl + K	Shows the Capture Options dialog box, which allows you to configure interfaces and capture options. See The “Capture Options” Dialog Box.
Start	Ctrl + E	Immediately starts capturing packets with the same settings as the last time.
Stop	Ctrl + E	Stops the currently running capture. See Stop the running capture.
Restart	Ctrl + R	Stops the currently running capture and starts it again with the same options.
Capture Filters…		Shows a dialog box that allows you to create and edit capture filters. You can name filters and save them for future use. See Defining And Saving Filters.
Refresh Interfaces	F5	Clear and recreate the interface list.

The “Analyze” Menu

The Wireshark Analyze menu contains the fields shown in Analyze menu items.

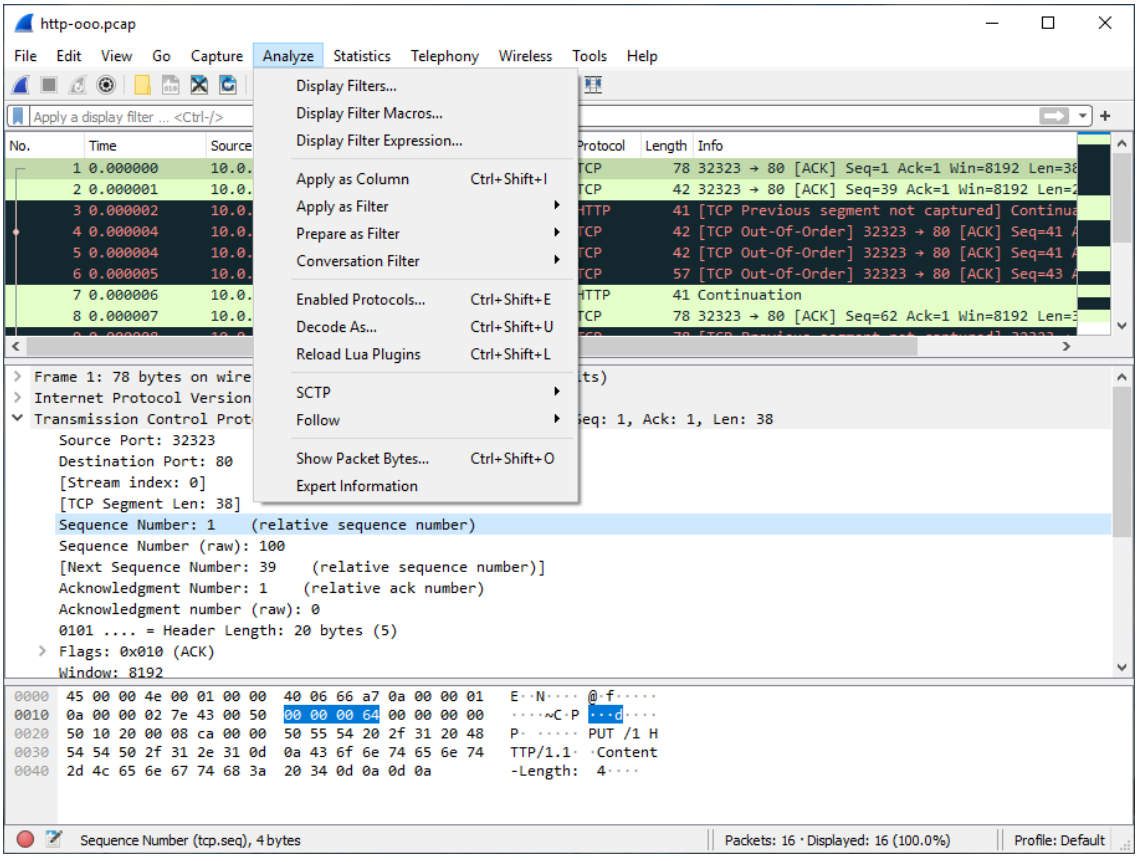


Figure 9. The “Analyze” Menu

Table 9. Analyze menu items

Menu Item	Accelerator	Description
Display Filters…		Displays a dialog box that allows you to create and edit display filters. You can name filters, and you can save them for future use. See Defining And Saving Filters.
Display Filter Macros…		Shows a dialog box that allows you to create and edit display filter macros. You can name filter macros, and you can save them for future use. See Defining And Saving Filter Macros.
Apply as Column	`Shift` + `Ctrl` + `I`	Adds the selected protocol item in the packet details pane as a column to the packet list.
Apply as Filter		Change the current display filter and apply it immediately. Depending on the chosen menu item, the current display filter string will be replaced or appended to by the selected protocol field in the packet details pane.
Prepare as Filter		Change the current display filter but won't apply it. Depending on the chosen menu item, the current display filter string will be replaced or appended to by the selected protocol field in the packet details pane.
Conversation Filter		Apply a conversation filter for various protocols.
Enabled Protocols…	`Shift` + `Ctrl` + `E`	Enable or disable various protocol dissectors. See The "Enabled Protocols" dialog box.
Decode As…		Decode certain packets as a particular protocol. See User Specified Decodes.
Follow › TCP Stream		Open a window that displays all the TCP segments captured that are on the same TCP connection as a selected packet. See Following Protocol Streams.
Follow › UDP Stream		Same functionality as "Follow TCP Stream" but for UDP "streams".
Follow › TLS Stream		Same functionality as "Follow TCP Stream" but for TLS or SSL streams. See the wiki page on TLS for instructions on providing TLS keys.
Follow › HTTP Stream		Same functionality as "Follow TCP Stream" but for HTTP streams.

Menu Item	Accelerator	Description
Expert Info		Open a window showing expert information found in the capture. Some protocol dissectors add packet detail items for notable or unusual behavior, such as invalid checksums or retransmissions. Those items are shown here. See Expert Information for more information. The amount of information will vary depend on the protocol

The “Statistics” Menu

The Wireshark Statistics menu contains the fields shown in Statistics menu items.

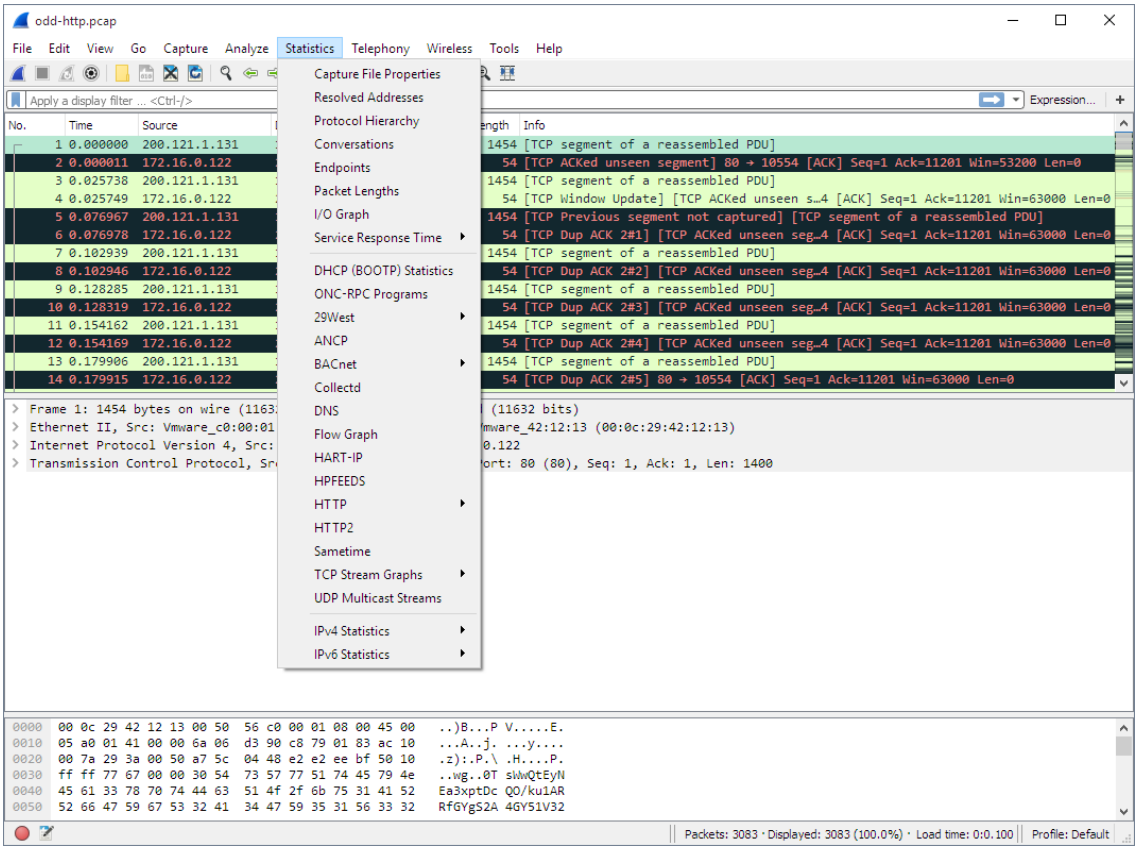


Figure 10. The “Statistics” Menu

Each menu item brings up a new window showing specific statistics.

Table 10. Statistics menu items

Menu Item	Accelerator	Description
Capture File Properties		Show information about the capture file, see The “Capture File Properties” Dialog.

Menu Item	Accelerator	Description
Resolved Addresses		See Resolved Addresses
Protocol Hierarchy		Display a hierarchical tree of protocol statistics, see The “Protocol Hierarchy” Window.
Conversations		Display a list of conversations (traffic between two endpoints), see The “Conversations” Window.
Endpoints		Display a list of endpoints (traffic to/from an address), see The “Endpoints” Window.
Packet Lengths		See Packet Lengths
I/O Graphs		Display user specified graphs (e.g., the number of packets in the course of time), see The “I/O Graphs” Window.
Service Response Time		Display the time between a request and the corresponding response, see Service Response Time.
DHCP (BOOTP)		See DHCP (BOOTP) Statistics
NetPerfMeter		See NetPerfMeter Statistics
ONC-RPC Programs		See ONC-RPC Programs
29West		See 29West
ANCP		See ANCP
BACnet		See BACnet
Collectd		See Collectd
DNS		See DNS
Flow Graph		See Flow Graph
HART-IP		See HART-IP
HPFEEDS		See HPFEEDS
HTTP		HTTP request/response statistics, see HTTP Statistics
HTTP2		See HTTP2
Sametime		See Sametime
TCP Stream Graphs		See TCP Stream Graphs
UDP Multicast Streams		See UDP Multicast Streams
Reliable Server Pooling (RSerPool)		See Reliable Server Pooling (RSerPool)

Menu Item	Accelerator	Description
F5		See F5
IPv4 Statistics		See IPv4 Statistics
IPv6 Statistics		See IPv6 Statistics

The “Telephony” Menu

The Wireshark Telephony menu contains the fields shown in Telephony menu items.

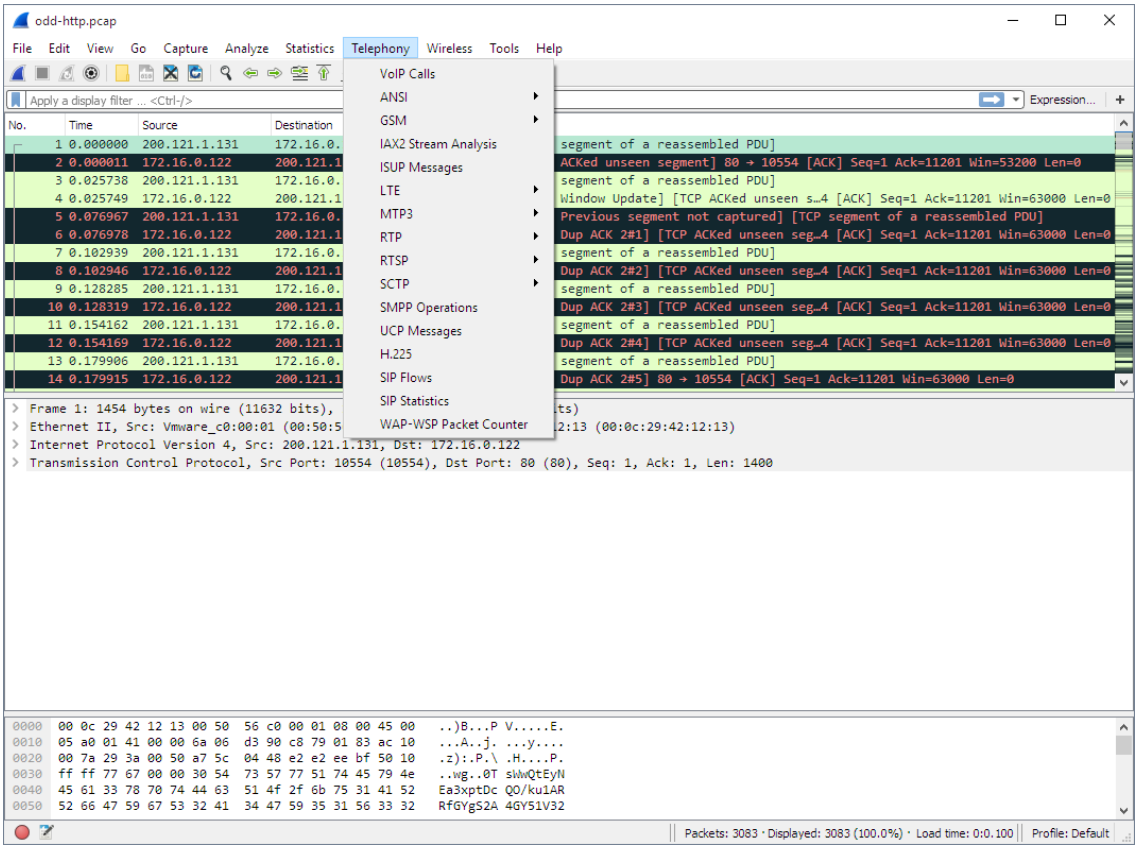


Figure 11. The “Telephony” Menu

Each menu item shows specific telephony related statistics.

Table 11. Telephony menu items

Menu Item	Accelerator	Description
VoIP Calls...		See VoIP Calls Window
ANSI		See ANSI
GSM		See GSM Windows
IAX2 Stream Analysis		See IAX2 Stream Analysis Window
ISUP Messages		See ISUP Messages Window

Menu Item	Accelerator	Description
LTE		See LTE
MTP3		See MTP3 Windows
Osmux		See Osmux Windows
RTP		See RTP Streams Window and RTP Stream Analysis Window
RTSP		See RTSP Window
SCTP		See SCTP Windows
SMPP Operations		See SMPP Operations Window
UCP Messages		See UCP Messages Window
H.225		See H.225 Window
SIP Flows		See SIP Flows Window
SIP Statistics		See SIP Statistics Window
WAP-WSP Packet Counter		See WAP-WSP Packet Counter Window

The “Wireless” Menu

The Wireless menu lets you analyze Bluetooth and IEEE 802.11 wireless LAN activity as shown in The “Wireless” Menu.

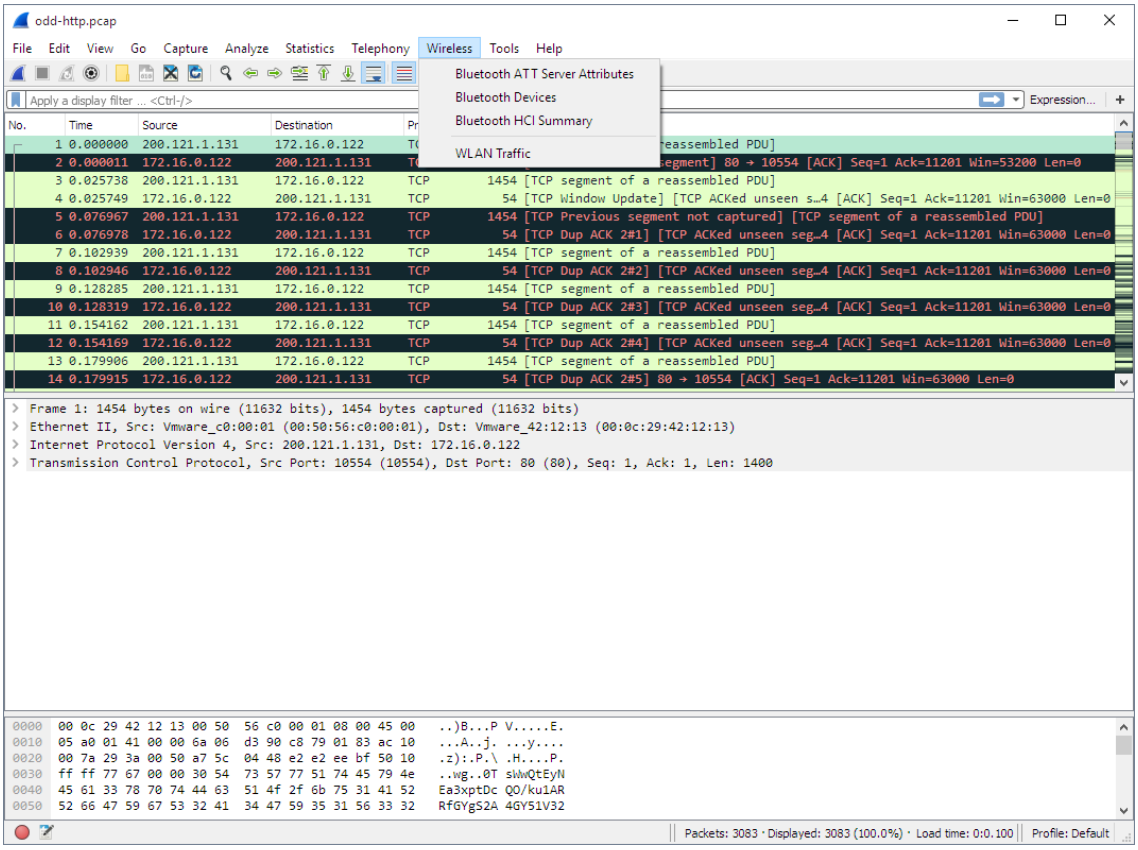


Figure 12. The “Wireless” Menu

Each menu item shows specific Bluetooth and IEEE 802.11 statistics.

Table 12. Wireless menu items

Menu Item	Accelerator	Description
Bluetooth ATT Server Attributes		See Bluetooth ATT Server Attributes
Bluetooth Devices		See Bluetooth Devices
Bluetooth HCI Summary		See Bluetooth HCI Summary
WLAN Traffic		See WLAN Traffic

The “Tools” Menu

The Wireshark Tools menu contains the fields shown in Tools menu items.

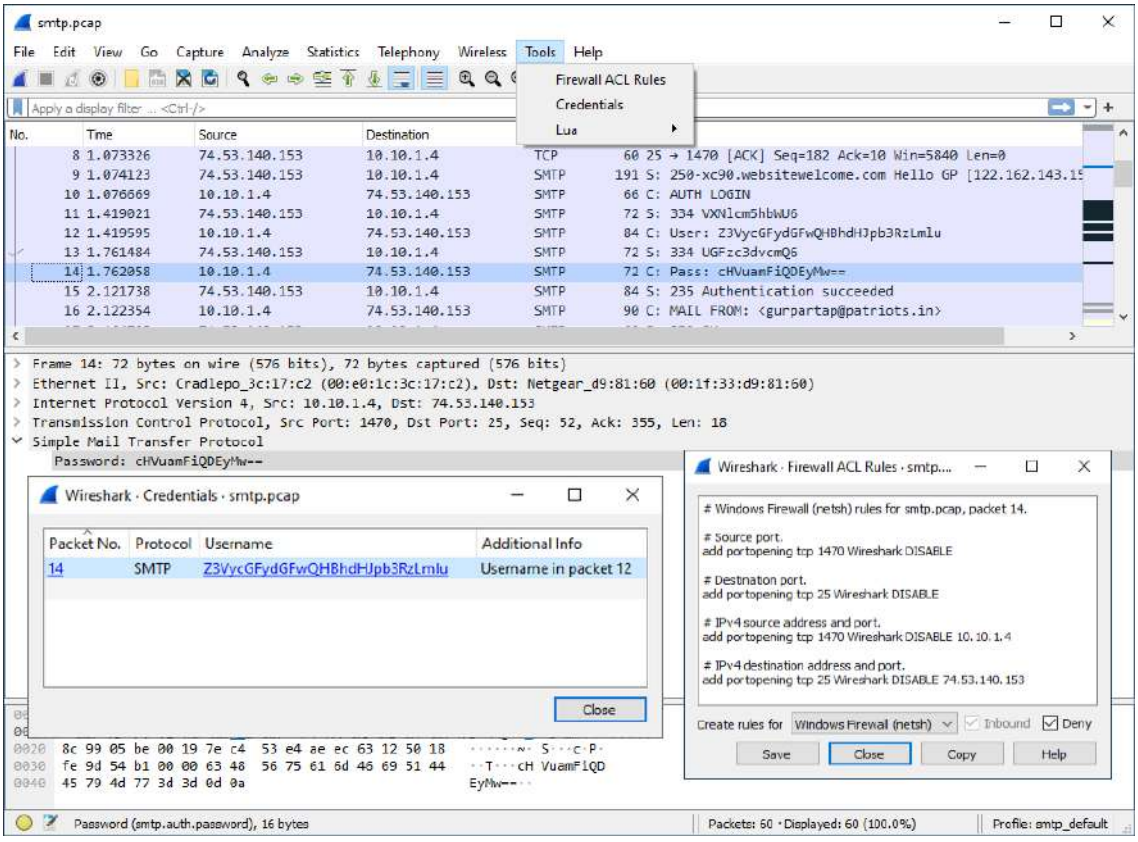


Figure 13. The "Tools" Menu

Table 13. Tools menu items

Menu Item	Accelerator	Description
Firewall ACL Rules		This allows you to create command-line ACL rules for many different firewall products, including Cisco IOS, Linux Netfilter (iptables), OpenBSD pf and Windows Firewall (via netsh). Rules for MAC addresses, IPv4 addresses, TCP and UDP ports, and IPv4+port combinations are supported. It is assumed that the rules will be applied to an outside interface. Menu item is greyed out unless one (and only one) frame is selected in the packet list.

Menu Item	Accelerator	Description
Credentials		This allows you to extract credentials from the current capture file. Some of the dissectors (ftp, http, imap, pop, smtp) have been instrumented to provide the module with usernames and passwords and more will be instrumented in the future. The window dialog provides you the packet number where the credentials have been found, the protocol that provided them, the username and protocol specific information.
Lua		These options allow you to work with the Lua interpreter optionally built into Wireshark. See “Lua Support in Wireshark” in the Wireshark Developer’s Guide. The Lua menu structure is set by console.lua in the Wireshark install directory.

The “Help” Menu

The Wireshark Help menu contains the fields shown in Help menu items.

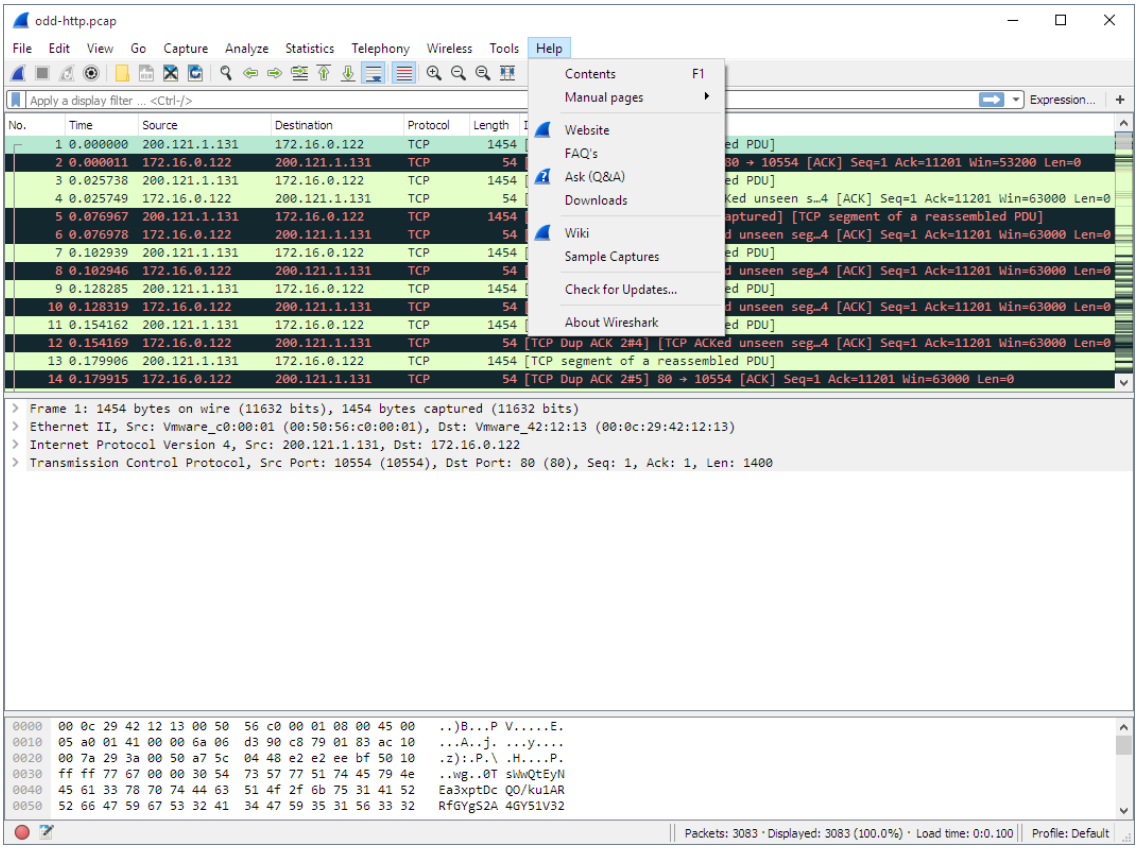


Figure 14. The “Help” Menu

Table 14. Help menu items

Menu Item	Accelerator	Description
Contents	F1	This menu item brings up a basic help system.
Manual Pages › ...		This menu item starts a Web browser showing one of the locally installed html manual pages.
Website		This menu item starts a Web browser showing the webpage from: https://www.wireshark.org/.
FAQs		This menu item starts a Web browser showing various FAQs.
Downloads		This menu item starts a Web browser showing the downloads from: https://www.wireshark.org/download.html.
Wiki		This menu item starts a Web browser showing the front page from: https://gitlab.com/wireshark/wireshark/wikis/.
Sample Captures		This menu item starts a Web browser showing the sample captures from: https://gitlab.com/wireshark/wireshark/wikis/SampleCaptures.
About Wireshark		This menu item brings up an information window that provides various detailed information items on Wireshark, such as how it's built, the plugins loaded, the used folders, ...

NOTE

Opening a Web browser might be unsupported in your version of Wireshark. If this is the case the corresponding menu items will be hidden.

If calling a Web browser fails on your machine, nothing happens, or the browser starts but no page is shown, have a look at the web browser setting in the preferences dialog.

The "Main" Toolbar

The main toolbar provides quick access to frequently used items from the menu. This toolbar cannot be customized by the user, but it can be hidden using the View menu if the space on the screen is needed to show more packet data.

Items in the toolbar will be enabled or disabled (greyed out) similar to their corresponding menu items. For example, in the image below shows the main window toolbar after a file has been opened. Various file-related buttons are enabled, but the stop capture button is disabled because a capture is not in progress.

Figure 15. The “Main” toolbar

Table 15. Main toolbar items

Toolbar Icon	Toolbar Item	Menu Item	Description
	**[Start]**	**Capture › Start**	Starts capturing packets with the same options as the last capture or the default options if none were set (Start Capturing).
	**[Stop]**	**Capture › Stop**	Stops the currently running capture (Start Capturing).
	**[Restart]**	**Capture › Restart**	Restarts the current capture session.
	**[Options...]**	**Capture › Options...**	Opens the “Capture Options” dialog box. See Start Capturing for details.
	[Open...]	**File › Open...**	Opens the file open dialog box, which allows you to load a capture file for viewing. It is discussed in more detail in The “Open Capture File” Dialog Box.
	**[Save As...]**	**File › Save As...**	Save the current capture file to whatever file you would like. See The “Save Capture File As” Dialog Box for details. If you currently have a temporary capture file open the “Save” icon will be shown instead.
	**[Close]**	**File › Close**	Closes the current capture. If you have not saved the capture, you will be asked to save it first.
	**[Reload]**	**View › Reload**	Reloads the current capture file.
	**[Find Packet...]**	**Edit › Find Packet...**	Find a packet based on different criteria. See Finding Packets for details.
	**[Go Back]**	**Go › Go Back**	Jump back in the packet history. Hold down the `Alt` key (`Option` on macOS) to go back in the selection history.
	**[Go Forward]**	**Go › Go Forward**	Jump forward in the packet history. Hold down the `Alt` key (`Option` on macOS) to go forward in the selection history.
	**[Go to Packet...]**	**Go › Go to Packet...**	Go to a specific packet.

Toolbar Icon	Toolbar Item	Menu Item	Description
	**[Go To First Packet]**	**Go › First Packet**	Jump to the first packet of the capture file.
	**[Go To Last Packet]**	**Go › Last Packet**	Jump to the last packet of the capture file.
	**[Auto Scroll in Live Capture]**	**View › Auto Scroll in Live Capture**	Auto scroll packet list while doing a live capture (or not).
	**[Colorize]**	**View › Colorize**	Colorize the packet list (or not).
	**[Zoom In]**	**View › Zoom In**	Zoom into the packet data (increase the font size).
	**[Zoom Out]**	**View › Zoom Out**	Zoom out of the packet data (decrease the font size).
	**[Normal Size]**	**View › Normal Size**	Set zoom level back to 100%.
	**[Resize Columns]**	**View › Resize Columns**	Resize columns, so the content fits into them.

The “Filter” Toolbar

The filter toolbar lets you quickly edit and apply display filters. More information on display filters is available in Filtering Packets While Viewing.

Figure 16. The “Filter” toolbar

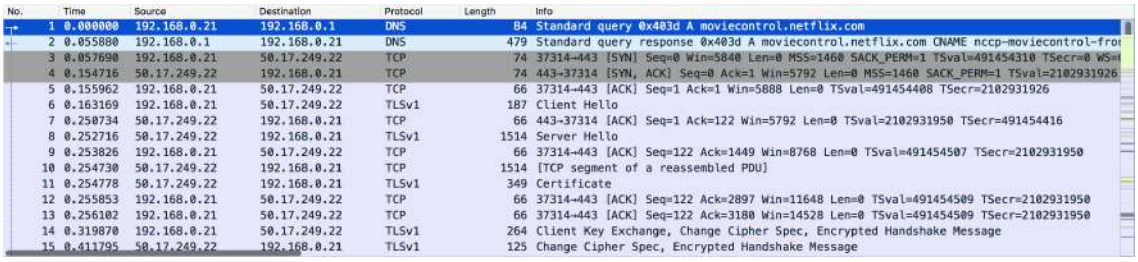
Table 16. Filter toolbar items

Toolbar Icon	Name	Description
	Bookmarks	Manage or select saved filters.

Toolbar Icon	Name	Description
	Filter Input	The area to enter or edit a display filter string, see Building Display Filter Expressions. A syntax check of your filter string is done while you are typing. The background will turn red if you enter an incomplete or invalid string, and will become green when you enter a valid string. After you've changed something in this field, don't forget to press the Apply button (or the Enter/Return key), to apply this filter string to the display. This field is also where the current applied filter is displayed.
	Clear	Reset the current display filter and clear the edit area.
	Apply	Apply the current value in the edit area as the new display filter. Applying a display filter on large capture files might take quite a long time.
	Recent	Select from a list of recently applied filters.
	Add Button	Add a new filter button.
[Squirrels]	Filter Button	Filter buttons are handy shortcuts that apply a display filter as soon as you press them. You can create filter buttons by pressing the **[+]** button, right-clicking in the filter button area, or opening the Filter Button section of the Preferences Dialog. The example shows a filter button with the label "Squirrels". If you have lots of buttons you can arrange them into groups by using "//" as a label separator. For example, if you create buttons named "Not Squirrels // Rabbits" and "Not Squirrels // Capybaras" they will show up in the toolbar under a single button named "Not Squirrels".

The "Packet List" Pane

The packet list pane displays all the packets in the current capture file.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.21	192.168.0.1	DNS	84	Standard query 0x403d A moviecontrol.netflix.com
2	0.055880	192.168.0.1	192.168.0.21	DNS	479	Standard query response 0x403d A moviecontrol.netflix.com CNAME nccp-moviecontrol-fro
3	0.057690	192.168.0.21	50.17.249.22	TCP	74	37314→443 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=491454310 TSecr=0 WS=
4	0.154716	50.17.249.22	192.168.0.21	TCP	74	443→37314 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=2102931926
5	0.155962	192.168.0.21	50.17.249.22	TCP	66	37314→443 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=491454408 TSecr=2102931926
6	0.163169	192.168.0.21	50.17.249.22	TLSv1	187	Client Hello
7	0.250734	50.17.249.22	192.168.0.21	TCP	66	443→37314 [ACK] Seq=1 Ack=122 Win=5792 Len=0 TSval=2102931950 TSecr=491454416
8	0.252716	50.17.249.22	192.168.0.21	TLSv1	1514	Server Hello
9	0.253826	192.168.0.21	50.17.249.22	TCP	66	37314→443 [ACK] Seq=122 Ack=1449 Win=8768 Len=0 TSval=491454507 TSecr=2102931950
10	0.254730	50.17.249.22	192.168.0.21	TCP	1514	[TCP segment of a reassembled PDU]
11	0.254778	50.17.249.22	192.168.0.21	TLSv1	349	Certificate
12	0.255853	192.168.0.21	50.17.249.22	TCP	66	37314→443 [ACK] Seq=122 Ack=2897 Win=11648 Len=0 TSval=491454509 TSecr=2102931950
13	0.256102	192.168.0.21	50.17.249.22	TCP	66	37314→443 [ACK] Seq=122 Ack=3180 Win=14528 Len=0 TSval=491454509 TSecr=2102931950
14	0.319870	192.168.0.21	50.17.249.22	TLSv1	264	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
15	0.411795	50.17.249.22	192.168.0.21	TLSv1	125	Change Cipher Spec, Encrypted Handshake Message

Figure 17. The "Packet List" pane

Each line in the packet list corresponds to one packet in the capture file. If you select a line in this pane, more details will be displayed in the "Packet Details" and "Packet Bytes" panes.

While dissecting a packet, Wireshark will place information from the protocol dissectors into the columns. As higher-level protocols might overwrite information from lower levels, you will typically see the information from the highest possible level only.

For example, let's look at a packet containing TCP inside IP inside an Ethernet packet. The Ethernet dissector will write its data (such as the Ethernet addresses), the IP dissector will overwrite this by its own (such as the IP addresses), the TCP dissector will overwrite the IP information, and so on.

There are many different columns available. You can choose which columns are displayed in the preferences. See Preferences.

The default columns will show:

- **[No.]** The number of the packet in the capture file. This number won't change, even if a display filter is used.
- **[Time]** The timestamp of the packet. The presentation format of this timestamp can be changed, see Time Display Formats And Time References.
- **[Source]** The address where this packet is coming from.
- **[Destination]** The address where this packet is going to.
- **[Protocol]** The protocol name in a short (perhaps abbreviated) version.
- **[Length]** The length of each packet.
- **[Info]** Additional information about the packet content.

The first column shows how each packet is related to the selected packet. For example, in the image above the first packet is selected, which is a DNS request. Wireshark shows a rightward arrow for the request itself, followed by a leftward arrow for the response in packet 2. Why is there a dashed line? There are more DNS packets further down that use the same port numbers. Wireshark treats them as belonging to the same conversation and draws a line connecting them.

Related packet symbols

First packet in a conversation.

Part of the selected conversation.

Not part of the selected conversation.

Last packet in a conversation.

Request.

Response.

The selected packet acknowledges this packet.

The selected packet is a duplicate acknowledgement of this packet.

The selected packet is related to this packet in some other way, e.g., as part of reassembly.

The packet list has an *Intelligent Scrollbar* which shows a miniature map of nearby packets. Each raster line of the scrollbar corresponds to a single packet, so the number of packets shown in the map depends on your physical display and the height of the packet list. A tall packet list on a high-resolution (“Retina”) display will show you quite a few packets. In the image above the scrollbar shows the status of more than 500 packets along with the 15 shown in the packet list itself.

Right clicking will show a context menu, described in Pop-up menu of the “Packet List” pane.

The “Packet Details” Pane

The packet details pane shows the current packet (selected in the “Packet List” pane) in a more detailed form.

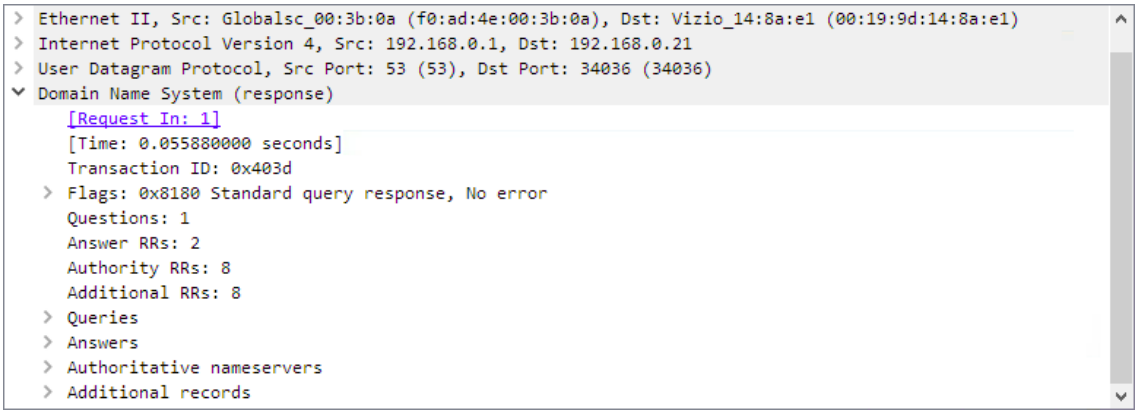


Figure 18. The “Packet Details” pane

This pane shows the protocols and protocol fields of the packet selected in the “Packet List” pane.

The protocol summary lines (subtree labels) and fields of the packet are shown in a tree which can be expanded and collapsed.

There is a context menu (right mouse click) available. See details in Pop-up menu of the “Packet Details” pane.

Some protocol fields have special meanings.

- **Generated fields.** Wireshark itself will generate additional protocol information which isn’t present in the captured data. This information is enclosed in square brackets (“[” and “]”). Generated information includes response times, TCP analysis, IP geolocation information, and checksum validation.
- **Links.** If Wireshark detects a relationship to another packet in the capture file it will generate a link to that packet. Links are underlined and displayed in blue. If you double-clicked on a link Wireshark will jump to the corresponding packet.

The “Packet Bytes” Pane

The packet bytes pane shows the data of the current packet (selected in the “Packet List” pane) in a hexdump style.

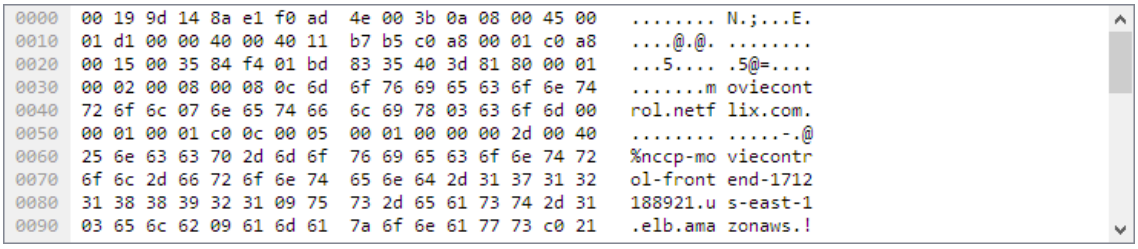


Figure 19. The “Packet Bytes” pane

The “Packet Bytes” pane shows a canonical hex dump of the packet data. Each line contains the data offset, sixteen hexadecimal bytes, and sixteen ASCII bytes. Non-printable bytes are replaced with a period (“.”).

Depending on the packet data, sometimes more than one page is available, e.g. when Wireshark has reassembled some packets into a single chunk of data. (See Packet Reassembly for details). In this case you can see each data source by clicking its corresponding tab at the bottom of the pane.

The default mode for viewing will highlight the bytes for a field where the mouse pointer is hovering above. The highlight will follow the mouse cursor as it moves. If this highlighting is not required or wanted, there are two methods for deactivating the functionality:

- **Temporary** By holding down the Ctrl button while moving the mouse, the highlighted field will not change
- **Permanently** Using the context menu (right mouse click) the hover highlighting may be activated/deactivated. This setting is stored in the selected profile *recent* file.

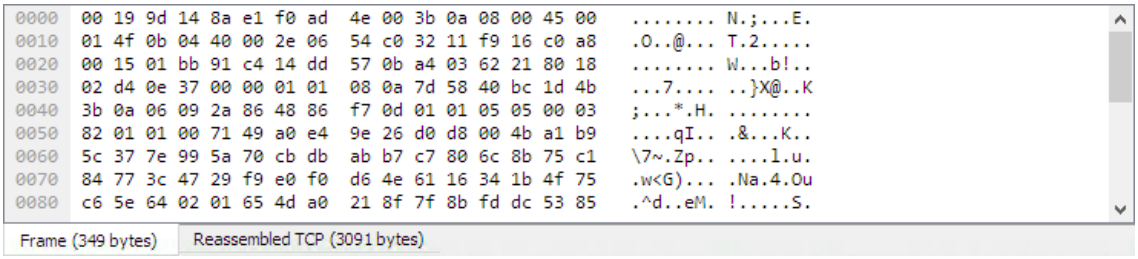


Figure 20. The “Packet Bytes” pane with tabs

Additional tabs typically contain data reassembled from multiple packets or decrypted data.

The “Packet Diagram” Pane

The packet diagram pane shows the current packet (selected in the “Packet List” pane) as a diagram, similar to ones used in textbooks and IETF RFCs.

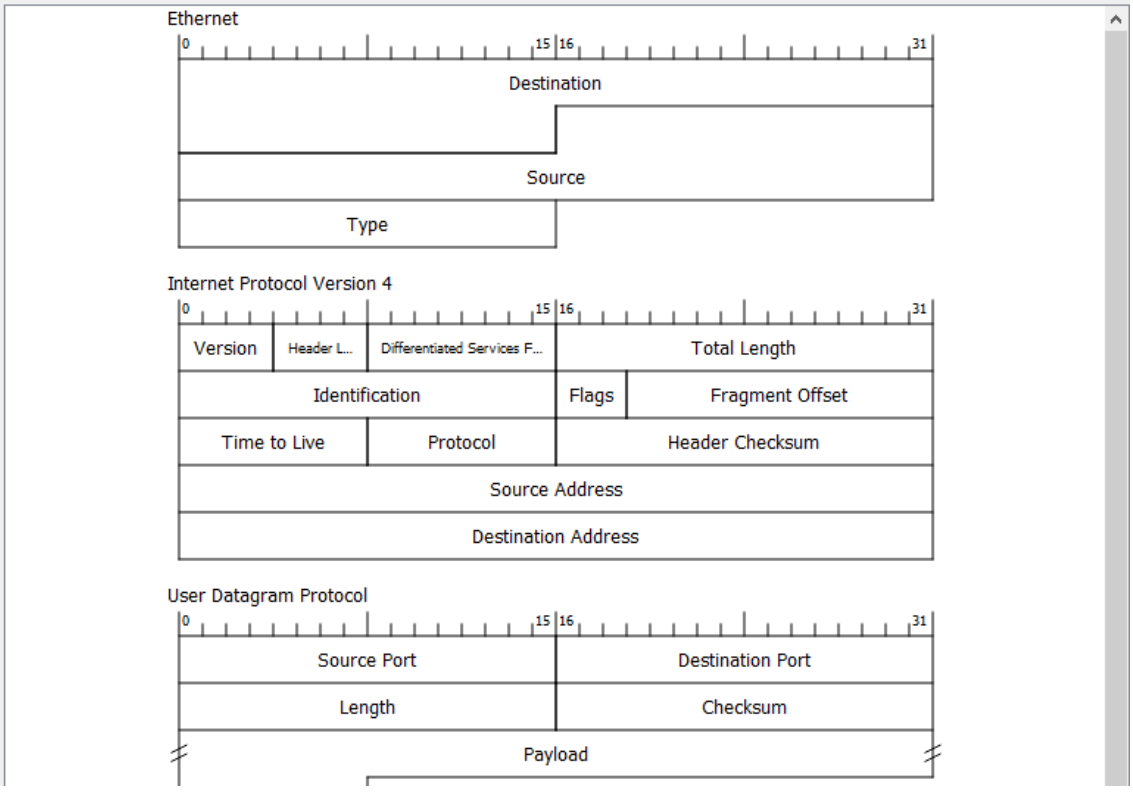


Figure 21. The “Packet Diagram” pane

This pane shows the protocols and top-level protocol fields of the packet selected in the “Packet List” pane as a series of diagrams.

There is a context menu (right mouse click) available. For details see Pop-up menu of the “Packet Diagram” pane.

The Statusbar

The statusbar displays informational messages.

In general, the left side will show context related information, the middle part will show information about the current capture file, and the right side will show the selected configuration profile. Drag the handles between the text areas to change the size.

Figure 22. The initial Statusbar

This statusbar is shown while no capture file is loaded, e.g., when Wireshark is started.

Figure 23. The Statusbar with a loaded capture file

The colorized bullet...

on the left shows the highest expert information level found in the currently loaded capture file. Hovering the mouse over this icon will show a description of the expert info level, and clicking the icon will bring up the Expert Information dialog box. For a detailed description of this dialog and each expert level, see Expert Information.

The edit icon...

on the left side lets you add a comment to the capture file using the Capture File Properties dialog.

The left side...

shows the capture file name by default. It also shows field information when hovering over and selecting items in the packet detail and packet bytes panes, as well as general notifications.

The middle...

shows the current number of packets in the capture file. The following values are displayed:

Packets

The number of captured packets.

Displayed

The number of packets currently being displayed.

Marked

The number of marked packets. Only displayed if you marked any packets.

Dropped

The number of dropped packets Only displayed if Wireshark was unable to capture all packets.

Ignored

The number of ignored packets Only displayed if you ignored any packets.

The right side…

shows the selected configuration profile. Clicking on this part of the statusbar will bring up a menu with all available configuration profiles, and selecting from this list will change the configuration profile.

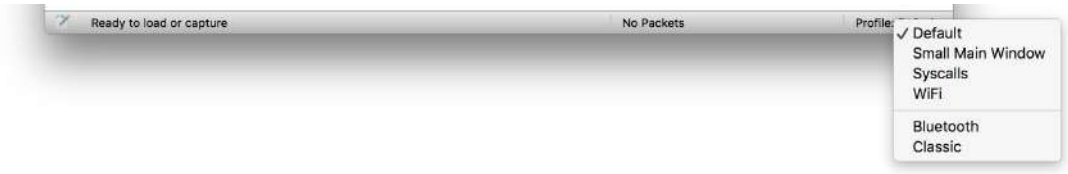


Figure 24. The Statusbar with a configuration profile menu

For a detailed description of configuration profiles, see Configuration Profiles.

Figure 25. The Statusbar with a selected protocol field

This is displayed if you have selected a protocol field in the “Packet Details” pane.

TIP The value between the parentheses (in this example “ipv6.src”) is the display filter field for the selected item. You can become more familiar with display filter fields by selecting different packet detail items.

Figure 26. The Statusbar with a display filter message

This is displayed if you are trying to use a display filter which may have unexpected results.

Capturing Live Network Data

Introduction

Capturing live network data is one of the major features of Wireshark.

The Wireshark capture engine provides the following features:

- Capture from different kinds of network hardware such as Ethernet or 802.11.
- Simultaneously capture from multiple network interfaces.
- Stop the capture on different triggers such as the amount of captured data, elapsed time, or the number of packets.
- Simultaneously show decoded packets while Wireshark is capturing.
- Filter packets, reducing the amount of data to be captured. See Filtering while capturing.
- Save packets in multiple files while doing a long-term capture, optionally rotating through a fixed number of files (a “ringbuffer”). See Capture files and file modes.

The capture engine still lacks the following features:

- Stop capturing (or perform some other action) depending on the captured data.

Prerequisites

Setting up Wireshark to capture packets for the first time can be tricky. A comprehensive guide “How To setup a Capture” is available at https://gitlab.com/wireshark/wireshark/wikis/CaptureSetup.

Here are some common pitfalls:

- You may need special privileges to start a live capture.
- You need to choose the right network interface to capture packet data from.
- You need to capture at the right place in the network to see the traffic you want to see.

If you have any problems setting up your capture environment, you should have a look at the guide mentioned above.

Start Capturing

The following methods can be used to start capturing packets with Wireshark:

- You can double-click on an interface in the welcome screen.
- You can select an interface in the welcome screen, then select **Capture › Start** or click the first

toolbar button.

- You can get more detailed information about available interfaces using The “Capture Options” Dialog Box (**Capture › Options…**).
- If you already know the name of the capture interface you can start Wireshark from the command line:

```
$ wireshark -i eth0 -k
```

This will start Wireshark capturing on interface `eth0`. More details can be found at Start Wireshark from the command line.

The “Capture” Section Of The Welcome Screen

When you open Wireshark without starting a capture or opening a capture file it will display the “Welcome Screen,” which lists any recently opened capture files and available capture interfaces. Network activity for each interface will be shown in a sparkline next to the interface name. It is possible to select more than one interface and capture from them simultaneously.

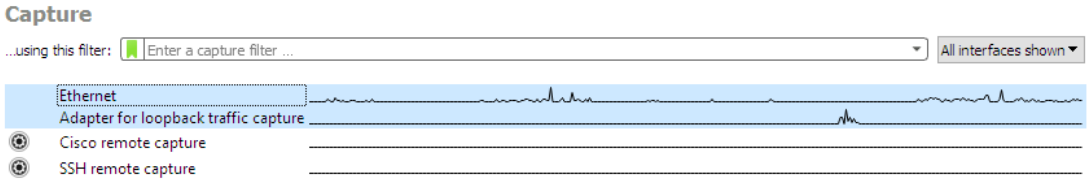


Figure 27. Capture interfaces on Microsoft Windows

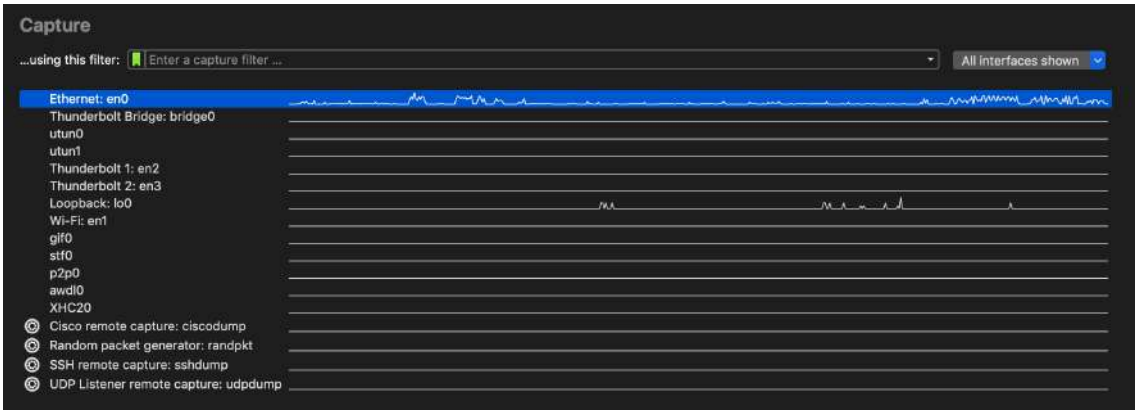


Figure 28. Capture interfaces on macOS

Some interfaces allow or require configuration prior to capture. This will be indicated by a configuration icon (⚙) to the left of the interface name. Clicking on the icon will show the configuration dialog for that interface.

Hovering over an interface will show any associated IPv4 and IPv6 addresses and its capture filter.

Wireshark isn’t limited to just network interfaces — on most systems you can also capture USB,

Bluetooth, and other types of packets. Note also that an interface might be hidden if it's inaccessible to Wireshark or if it has been hidden as described in The "Manage Interfaces" Dialog Box.

The "Capture Options" Dialog Box

When you select **Capture › Options...** (or use the corresponding item in the main toolbar), Wireshark pops up the "Capture Options" dialog box as shown in The "Capture Options" input tab. If you are unsure which options to choose in this dialog box, leaving the defaults settings as they are should work well in many cases.

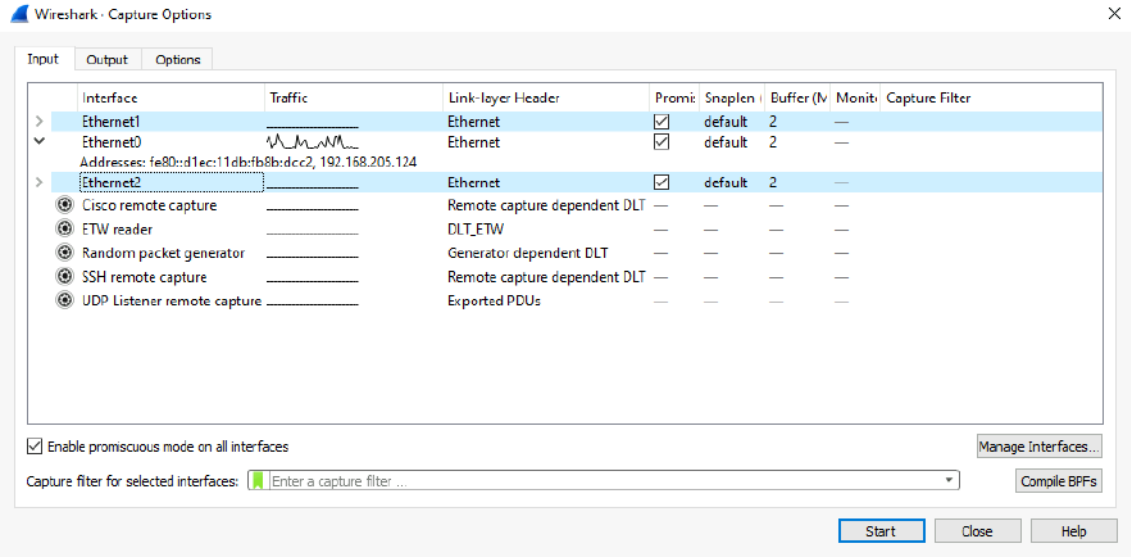


Figure 29. The "Capture Options" input tab

The "Input" tab contains the "Interface" table, which shows the following columns:

Interface

The interface name.

Some interfaces allow or require configuration prior to capture. This will be indicated by a configuration icon (⚙) to the left of the interface name. Clicking on the icon will show the configuration dialog for that interface.

Traffic

A sparkline showing network activity over time.

Link-layer Header

The type of packet captured by this interface. In some cases it is possible to change this. See Link-layer header type for more details.

Promiscuous

Lets you put this interface in promiscuous mode while capturing. Note that another application might override this setting.

Snaplen

The snapshot length, or the number of bytes to capture for each packet. You can set an explicit length if needed, e.g., for performance or privacy reasons.

Buffer

The size of the kernel buffer that is reserved for capturing packets. You can increase or decrease this as needed, but the default is usually sufficient.

Monitor Mode

Lets you capture full, raw 802.11 headers. Support depends on the interface type, hardware, driver, and OS. Note that enabling this might disconnect you from your wireless network.

Capture Filter

The capture filter applied to this interface. You can edit the filter by double-clicking on it. See Filtering while capturing for more details about capture filters.

Hovering over an interface or expanding it will show any associated IPv4 and IPv6 addresses.

If “Enable promiscuous mode on all interfaces” is enabled, the individual promiscuous mode settings above will be overridden.

“Capture filter for selected interfaces” can be used to set a filter for more than one interface at the same time.

[Manage Interfaces] opens the The “Manage Interfaces” dialog box where pipes can be defined, local interfaces scanned or hidden, or remote interfaces added.

[Compile Selected BPFs] opens The “Compiled Filter Output” dialog box, which shows you the compiled bytecode for your capture filter. This can help to better understand the capture filter you created.

TIP

Linux power user tip

The execution of BPFs can be sped up on Linux by turning on BPF Just In Time compilation by executing

```
$ echo 1 >/proc/sys/net/core/bpf_jit_enable
```

if it is not enabled already. To make the change persistent you can use sysfsutils.

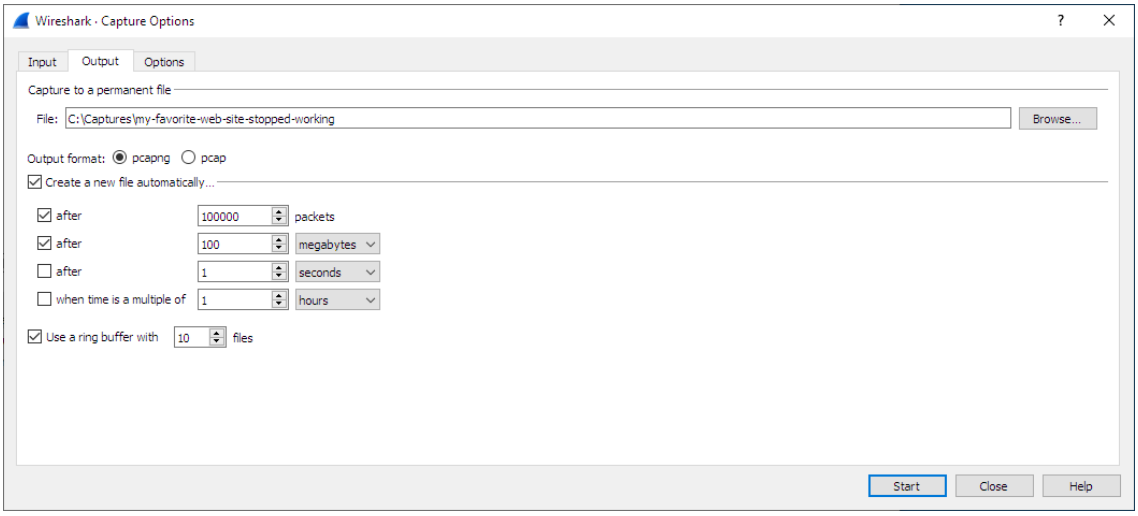


Figure 30. The "Capture Options" output tab

The "Output" tab shows the following information:

Capture to a permanent file

File

This field allows you to specify the file name that will be used for the capture file. It is left blank by default. If left blank, the capture data will be stored in a temporary file. See Capture files and file modes for details. You can also click on the button to the right of this field to browse through the filesystem.

Output format

Allows you to set the format of the capture file. pcapng is the default and is more flexible than pcap. pcapng might be required, e.g., if more than one interface is chosen for capturing. See https://gitlab.com/wireshark/wireshark/wikis/Development/PcapNg for more details on pcapng.

Create a new file automatically...

Sets the conditions for switching a new capture file. A new capture file can be created based on the following conditions:

- The number of packets in the capture file.
- The size of the capture file.
- The duration of the capture file.
- The wall clock time.

Use a ring buffer with

Multiple files only. Form a ring buffer of the capture files with the given number of files.

More details about capture files can be found in Capture files and file modes.

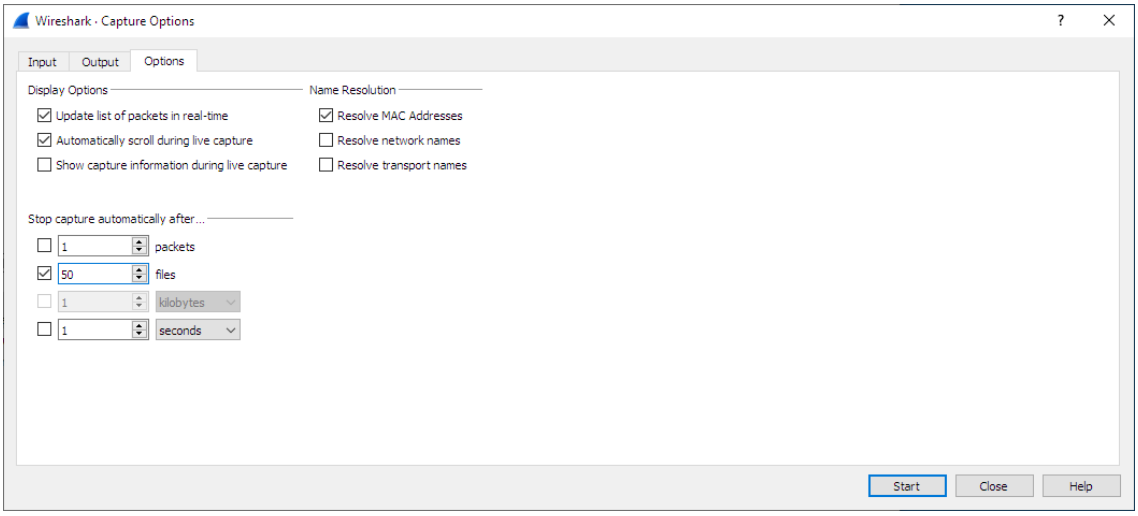


Figure 31. The “Capture Options” options tab

The “Options” tab shows the following information:

Display Options

Update list of packets in real-time

Updates the packet list pane in real time during capture. If you do not enable this, Wireshark will not display any packets until you stop the capture. When you check this, Wireshark captures in a separate process and feeds the captures to the display process.

Automatically scroll during live capture

Scroll the packet list pane as new packets come in, so you are always looking at the most recent packet. If you do not specify this Wireshark adds new packets to the packet list but does not scroll the packet list pane. This option is greyed out if “Update list of packets in real-time” is disabled.

Show capture information during capture

If this option is enabled, the capture information dialog described in While a Capture is running … will be shown while packets are captured.

Name Resolution

Resolve MAC addresses

Translate MAC addresses into names.

Resolve network names

Translate network addresses into names.

Resolve transport names

Translate transport names (port numbers).

See Name Resolution for more details on each of these options.

Stop capture automatically after…

Capturing can be stopped based on the following conditions:

- The number of packets in the capture file.
- The number of capture files.
- The capture file size.
- The capture file duration.

You can double click on an interface row in the “Input“ tab or click **[Start]** from any tab to commence the capture. You can click **[Cancel]** to apply your changes and close the dialog.

The “Manage Interfaces” Dialog Box

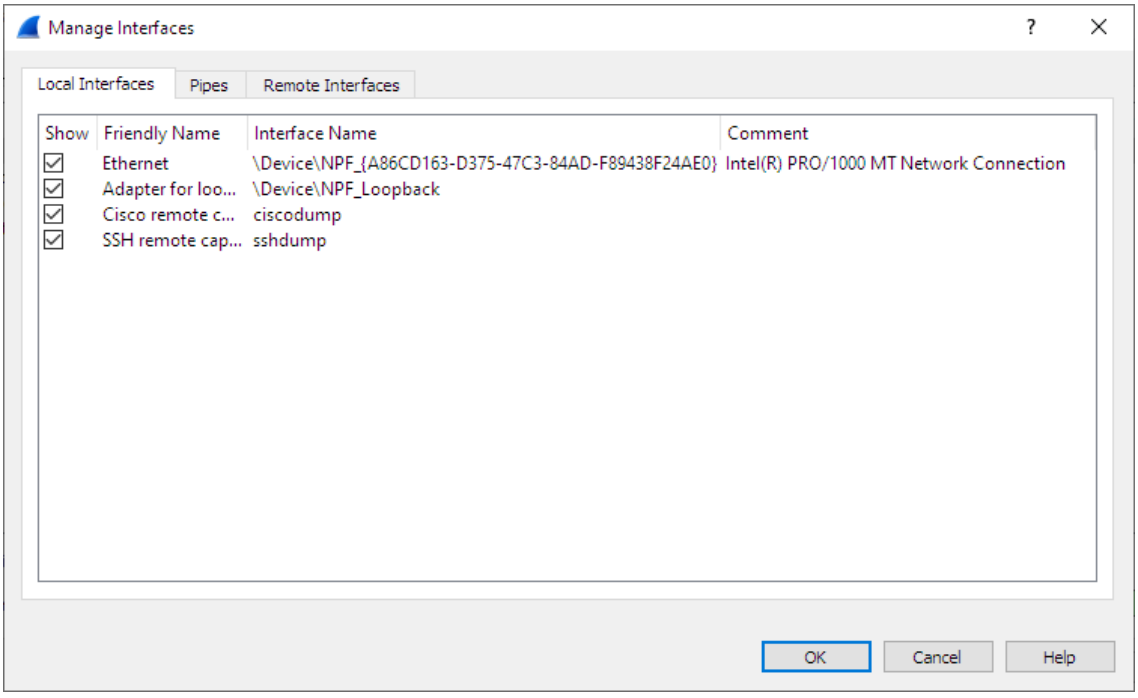


Figure 32. The “Manage Interfaces” dialog box

The “Manage Interfaces” dialog box initially shows the “Local Interfaces” tab, which lets you manage the following:

Show

Whether or not to show or hide this interface in the welcome screen and the “Capture Options” dialog.

Friendly Name

A name for the interface that is human readable.

Interface Name

The device name of the interface.

Comment

Can be used to add a descriptive comment for the interface.

The “Pipes” tab lets you capture from a named pipe. To successfully add a pipe, its associated named pipe must have already been created. Click **[+]** and type the name of the pipe including its path. Alternatively, **[Browse]** can be used to locate the pipe.

To remove a pipe from the list of interfaces, select it and press **[-]**.

On Microsoft Windows, the “Remote Interfaces” tab lets you capture from an interface on a different machine. The Remote Packet Capture Protocol service must first be running on the target platform before Wireshark can connect to it.

On Linux or Unix you can capture (and do so more securely) through an SSH tunnel.

To add a new remote capture interface, click **[+]** and specify the following:

Host

The IP address or host name of the target platform where the Remote Packet Capture Protocol service is listening. The drop-down list contains the hosts that have previously been successfully contacted. The list can be emptied by choosing “Clear list” from the drop-down list.

Port

Set the port number where the Remote Packet Capture Protocol service is listening on. Leave blank to use the default port (2002).

Null authentication

Select this if you don’t need authentication to take place for a remote capture to be started. This depends on the target platform. This is exactly as secure as it appears, i.e., it is not secure at all.

Password authentication

Lets you specify the username and password required to connect to the Remote Packet Capture Protocol service.

Each interface can optionally be hidden. In contrast to the local interfaces, they are not saved in the `preferences` file.

NOTE	Make sure you have outside access to port 2002 on the target platform. This is the default port used by the Remote Packet Capture Protocol service.

To remove a host including all its interfaces from the list, select it and click the **[-]** button.

The “Compiled Filter Output” Dialog Box

This figure shows the results of compiling the BPF filter for the selected interfaces.

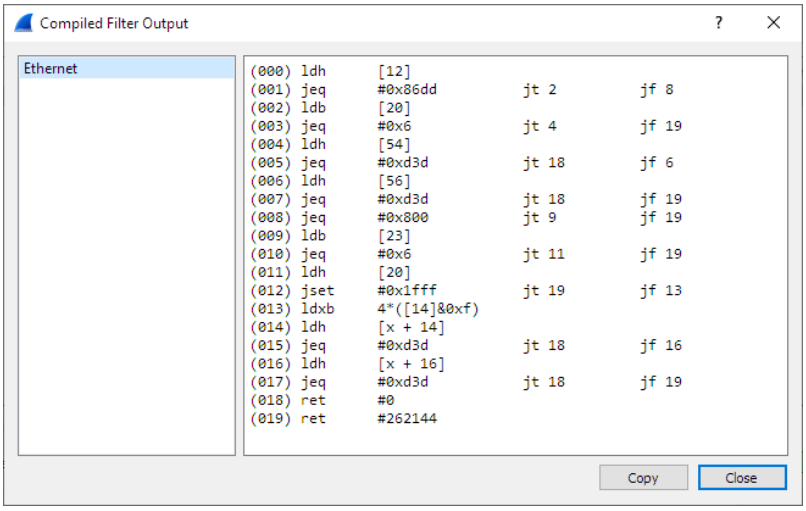


Figure 33. The “Compiled Filter Output” dialog box

In the list on the left the interface names are listed. The results of compiling a filter for the selected interface are shown on the right.

Capture files and file modes

While capturing, the underlying libpcap capturing engine will grab the packets from the network card and keep the packet data in a (relatively) small kernel buffer. This data is read by Wireshark and saved into a capture file.

By default, Wireshark saves packets to a temporary file. You can also tell Wireshark to save to a specific (“permanent”) file and switch to a different file after a given time has elapsed or a given number of packets have been captured. These options are controlled in the “Capture Options” dialog’s “Output” tab.

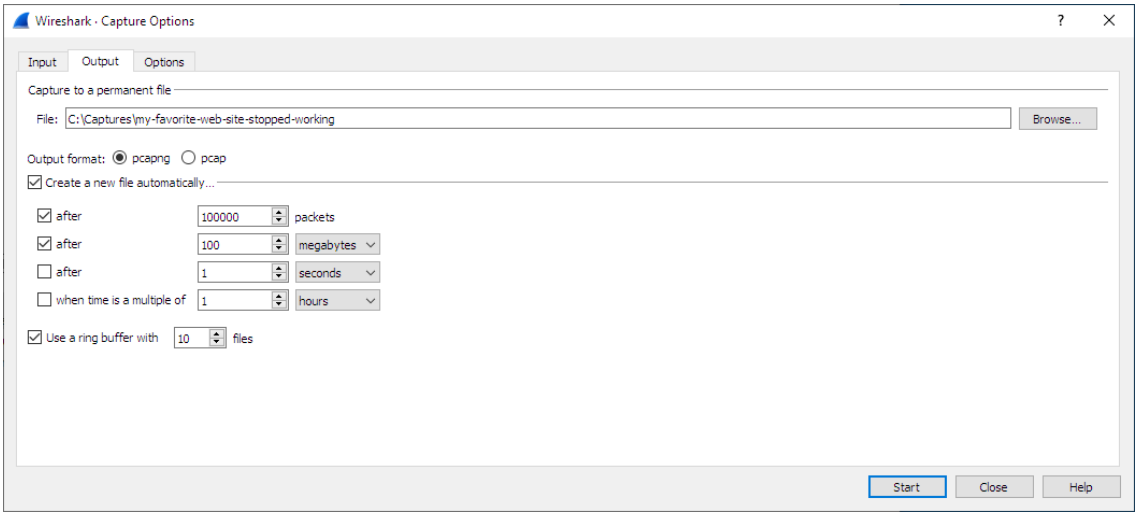


Figure 34. Capture output options

TIP	Working with large files (several hundred MB) can be quite slow. If you plan to do a long-term capture or capturing from a high traffic network, think about using one of the “Multiple files” options. This will spread the captured packets over several smaller files which can be much more pleasant to work with.

Using the “Multiple files” option may cut context related information. Wireshark keeps context information of the loaded packet data, so it can report context related problems (like a stream error) and keeps information about context related protocols (e.g., where data is exchanged at the establishing phase and only referred to in later packets). As it keeps this information only for the loaded file, using one of the multiple file modes may cut these contexts. If the establishing phase is saved in one file and the things you would like to see is in another, you might not see some of the valuable context related information.

Information about the folders used for capture files can be found in Files and Folders.

Table 17. Capture file mode selected by capture options

File Name	“Create a new file…”	“Use a ring buffer…”	Mode	Resulting filename(s) used
-	-	-	Single temporary file	wiresharkXXXXXX (where XXXXXX is a unique number)
foo.cap	-	-	Single named file	foo.cap
foo.cap	x	-	Multiple files, continuous	foo_00001_20220714110102.cap, foo_00002_20220714110318.cap, …
foo.cap	x	x	Multiple files, ring buffer	foo_00001_20220714110102.cap, foo_00002_20220714110318.cap, …

Single temporary file

A temporary file will be created and used (this is the default). After capturing is stopped this file can be saved later under a user specified name.

Single named file

A single capture file will be used. Choose this mode if you want to place the new capture file in a specific folder.

Multiple files, continuous

Like the “Single named file” mode, but a new file is created and used after reaching one of the multiple file switch conditions (one of the “Next file every…” values).

Multiple files, ring buffer

Much like “Multiple files continuous”, reaching one of the multiple files switch conditions (one of the “Next file every …” values) will switch to the next file. This will be a newly created file if value of “Ring buffer with n files” is not reached, otherwise it will replace the oldest of the formerly used files (thus forming a “ring”).

This mode will limit the maximum disk usage, even for an unlimited amount of capture input data, only keeping the latest captured data.

Link-layer header type

In most cases you won't have to modify link-layer header type. Some exceptions are as follows:

If you are capturing on an Ethernet device you might be offered a choice of "Ethernet" or "DOCSIS". If you are capturing traffic from a Cisco Cable Modem Termination System that is putting DOCSIS traffic onto the Ethernet to be captured, select "DOCSIS", otherwise select "Ethernet".

If you are capturing on an 802.11 device on some versions of BSD you might be offered a choice of "Ethernet" or "802.11". "Ethernet" will cause the captured packets to have fake ("cooked") Ethernet headers. "802.11" will cause them to have full IEEE 802.11 headers. Unless the capture needs to be read by an application that doesn't support 802.11 headers you should select "802.11".

If you are capturing on an Endace DAG card connected to a synchronous serial line you might be offered a choice of "PPP over serial" or "Cisco HDLC". If the protocol on the serial line is PPP, select "PPP over serial" and if the protocol on the serial line is Cisco HDLC, select "Cisco HDLC".

If you are capturing on an Endace DAG card connected to an ATM network you might be offered a choice of "RFC 1483 IP-over-ATM" or "Sun raw ATM". If the only traffic being captured is RFC 1483 LLC-encapsulated IP, or if the capture needs to be read by an application that doesn't support SunATM headers, select "RFC 1483 IP-over-ATM", otherwise select "Sun raw ATM".

Filtering while capturing

Wireshark supports limiting the packet capture to packets that match a *capture filter*. Wireshark capture filters are written in libpcap filter language. Below is a brief overview of the libpcap filter language's syntax. Complete documentation can be found at the pcap-filter man page. You can find many Capture Filter examples at https://gitlab.com/wireshark/wireshark/wikis/CaptureFilters.

You enter the capture filter into the "Filter" field of the Wireshark "Capture Options" dialog box, as shown in The "Capture Options" input tab.

A capture filter takes the form of a series of primitive expressions connected by conjunctions (*and/or*) and optionally preceded by *not*:

```
[not] primitive [and|or [not] primitive ...]
```

An example is shown in A capture filter for telnet that captures traffic to and from a particular host.

Example 1. A capture filter for telnet that captures traffic to and from a particular host

```
tcp port 23 and host 10.0.0.5
```

This example captures telnet traffic to and from the host 10.0.0.5, and shows how to use two primitives and the *and* conjunction. Another example is shown in Capturing all telnet traffic not from 10.0.0.5, and shows how to capture all telnet traffic except that from 10.0.0.5.

Example 2. Capturing all telnet traffic not from 10.0.0.5

```
tcp port 23 and not src host 10.0.0.5
```

A primitive is simply one of the following: *[src|dst] host <host>*

This primitive allows you to filter on a host IP address or name. You can optionally precede the primitive with the keyword *src|dst* to specify that you are only interested in source or destination addresses. If these are not present, packets where the specified address appears as either the source or the destination address will be selected.

ether [src|dst] host <ehost>

This primitive allows you to filter on Ethernet host addresses. You can optionally include the keyword *src|dst* between the keywords *ether* and *host* to specify that you are only interested in source or destination addresses. If these are not present, packets where the specified address appears in either the source or destination address will be selected.

gateway host <host>

This primitive allows you to filter on packets that used *host* as a gateway. That is, where the Ethernet source or destination was *host* but neither the source nor destination IP address was *host*.

[src|dst] net <net> [{mask <mask>}|{len <len>}]

This primitive allows you to filter on network numbers. You can optionally precede this primitive with the keyword *src|dst* to specify that you are only interested in a source or destination network. If neither of these are present, packets will be selected that have the specified network in either the source or destination address. In addition, you can specify either the netmask or the CIDR prefix for the network if they are different from your own.

[tcp|udp] [src|dst] port <port>

This primitive allows you to filter on TCP and UDP port numbers. You can optionally precede this primitive with the keywords *src|dst* and *tcp|udp* which allow you to specify that you are only interested in source or destination ports and TCP or UDP packets respectively. The keywords *tcp|udp* must appear before *src|dst*.

If these are not specified, packets will be selected for both the TCP and UDP protocols and when the specified address appears in either the source or destination port field.

less|greater <length>

This primitive allows you to filter on packets whose length was less than or equal to the specified length, or greater than or equal to the specified length, respectively.

ip|ether proto <protocol>

This primitive allows you to filter on the specified protocol at either the Ethernet layer or the IP layer.

ether|ip broadcast|multicast

This primitive allows you to filter on either Ethernet or IP broadcasts or multicasts.

<expr> relop <expr>

This primitive allows you to create complex filter expressions that select bytes or ranges of bytes in packets. Please see the pcap-filter man page at https://www.tcpdump.org/manpages/pcap-filter.7.html for more details.

Automatic Remote Traffic Filtering

If Wireshark is running remotely (using e.g., SSH, an exported X11 window, a terminal server, ...), the remote content has to be transported over the network, adding a lot of (usually unimportant) packets to the actually interesting traffic.

To avoid this, Wireshark tries to figure out if it's remotely connected (by looking at some specific environment variables) and automatically creates a capture filter that matches aspects of the connection.

The following environment variables are analyzed:

`SSH_CONNECTION` **(ssh)**

<remote IP> <remote port> <local IP> <local port>

`SSH_CLIENT` **(ssh)**

<remote IP> <remote port> <local port>

`REMOTEHOST` **(tcsh, others?)**

<remote name>

`DISPLAY` **(x11)**

[remote name]:<display num>

`SESSIONNAME` **(terminal server)**

<remote name>

On Windows it asks the operating system if it's running in a Remote Desktop Services environment.

While a Capture is running …

You might see the following dialog box while a capture is running:

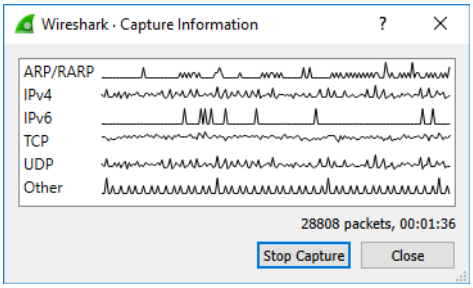


Figure 35. The "Capture Information" dialog box

This dialog box shows a list of protocols and their activity over time. It can be enabled via the "capture.show_info" setting in the "Advanced" preferences.

Stop the running capture

A running capture session will be stopped in one of the following ways:

1. The **[Stop Capture]** button in the "Capture Information" dialog box.
2. The **Capture › Stop** menu item.
3. The **[Stop]** toolbar button.
4. Pressing `Ctrl` + `E`.
5. The capture will be automatically stopped if one of the *Stop Conditions* is met, e.g., the maximum amount of data was captured.

Restart a running capture

A running capture session can be restarted with the same capture options as the last time, this will remove all packets previously captured. This can be useful, if some uninteresting packets are captured and there's no need to keep them.

Restart is a convenience function and equivalent to a capture stop following by an immediate capture start. A restart can be triggered in one of the following ways:

1. Using the **Capture › Restart** menu item.
2. Using the **[Restart]** toolbar button.

File Input, Output, And Printing

Introduction

This chapter will describe input and output of capture data.

- Open capture files in various capture file formats
- Save and export capture files in various formats
- Merge capture files together
- Import text files containing hex dumps of packets
- Print packets

Open Capture Files

Wireshark can read in previously saved capture files. To read them, simply select the **File › Open** menu or toolbar item. Wireshark will then pop up the "File Open" dialog box, which is discussed in more detail in The "Open Capture File" Dialog Box.

TIP	*You can use drag and drop to open files* On most systems you can open a file by simply dragging it in your file manager and dropping it onto Wireshark's main window.

If you haven't previously saved the current capture file you will be asked to do so to prevent data loss. This warning can be disabled in the preferences.

In addition to its native file format (pcapng), Wireshark can read and write capture files from a large number of other packet capture programs as well. See Input File Formats for the list of capture formats Wireshark understands.

The "Open Capture File" Dialog Box

The "Open Capture File" dialog box allows you to search for a capture file containing previously captured packets for display in Wireshark. The following sections show some examples of the Wireshark "Open File" dialog box. The appearance of this dialog depends on the system. However, the functionality should be the same across systems.

Common dialog behavior on all systems:

- Select files and directories.
- Click the **[Open]** button to accept your selected file and open it.
- Click the **[Cancel]** button to go back to Wireshark and not load a capture file.

- The **[Help]** button will take you to this section of the “User’s Guide”.

Wireshark adds the following controls:

- View file preview information such as the size and the number of packets in a selected a capture file.
- Specify a read filter with the “Read filter” field. This filter will be used when opening the new file. The text field background will turn green for a valid filter string and red for an invalid one. Read filters can be used to exclude various types of traffic, which can be useful for large capture files. They use the same syntax as display filters, which are discussed in detail in Filtering Packets While Viewing.
- Optionally force Wireshark to read a file as a particular type using the “Automatically detect file type” drop-down.

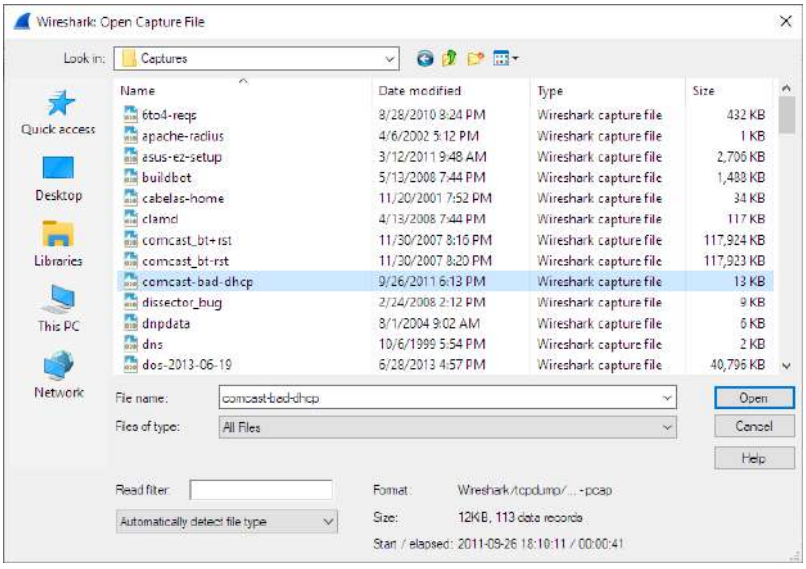


Figure 36. “Open” on Microsoft Windows

This is the common Windows file open dialog along with some Wireshark extensions.

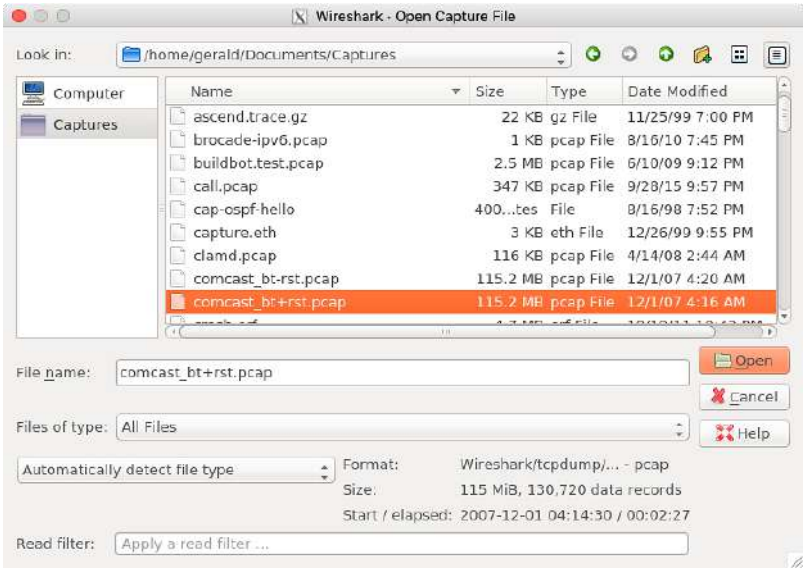


Figure 37. “Open” - Linux and UNIX

This is the common Qt file open dialog along with some Wireshark extensions.

Input File Formats

The native capture file formats used by Wireshark are:

- pcap. The default format used by the *libpcap* packet capture library. Used by *tcpdump, _Snort*, *Nmap*, *Ntop*, and many other tools.
- pcapng. A flexible, extensible successor to the pcap format. Wireshark 1.8 and later save files as pcapng by default. Versions prior to 1.8 used pcap. Used by Wireshark and by *tcpdump* in newer versions of macOS.

The following file formats from other capture tools can be opened by Wireshark:

- Oracle (previously Sun) *snoop* and *atmsnoop* captures
- Finisar (previously Shomiti) *Surveyor* captures
- Microsoft *Network Monitor* captures
- Novell *LANalyzer* captures
- AIX *iptrace* captures
- Cinco Networks NetXray captures
- NETSCOUT (previously Network Associates/Network General) Windows-based Sniffer and Sniffer Pro captures
- Network General/Network Associates DOS-based Sniffer captures (compressed or uncompressed) captures
- LiveAction (previously WildPackets/Savvius) *Peek/EtherHelp/PacketGrabber captures
- RADCOM’s WAN/LAN Analyzer captures

- Viavi (previously Network Instruments) Observer captures
- Lucent/Ascend router debug output
- captures from HP-UX nettl
- Toshiba's ISDN routers dump output
- output from *i4btrace* from the ISDN4BSD project
- traces from the EyeSDN USB S0
- the IPLog format output from the Cisco Secure Intrusion Detection System
- pppd logs (pppdump format)
- the output from VMS's TCPIPtrace/TCPtrace/UCX$TRACE utilities
- the text output from the DBS Etherwatch VMS utility
- Visual Networks' Visual UpTime traffic capture
- the output from CoSine L2 debug
- the output from InfoVista (previously Accellent) 5Views LAN agents
- Endace Measurement Systems' ERF format captures
- Linux Bluez Bluetooth stack hcidump -w traces
- Catapult (now Ixia/Keysight) DCT2000 .out files
- Gammu generated text output from Nokia DCT3 phones in Netmonitor mode
- IBM Series (OS/400) Comm traces (ASCII & UNICODE)
- Juniper Netscreen snoop captures
- Symbian OS btsnoop captures
- Tamosoft CommView captures
- Tektronix K12xx 32bit .rf5 format captures
- Tektronix K12 text file format captures
- Apple PacketLogger captures
- Captures from Aethra Telecommunications' PC108 software for their test instruments
- Citrix NetScaler Trace files
- Android Logcat binary and text format logs
- Colasoft Capsa and PacketBuilder captures
- Microprossmplog files
- Unigraf DPA-400 DisplayPort AUX channel monitor traces
- 802.15.4 traces from Daintree's Sensor Network Analyzer
- MPEG-2 Transport Streams as defined in ISO/IEC 13818-1

- Log files from the *candump* utility
- Logs from the BUSMASTER tool
- Ixia IxVeriWave raw captures
- Rabbit Labs CAM Inspector files
- *systemd* journal files
- 3GPP TS 32.423 trace files

New file formats are added from time to time.

It may not be possible to read some formats dependent on the packet types captured. Ethernet captures are usually supported for most file formats but it may not be possible to read other packet types such as PPP or IEEE 802.11 from all file formats.

Saving Captured Packets

You can save captured packets by using the **File › Save** or **File › Save As…** menu items. You can choose which packets to save and which file format to be used.

Not all information will be saved in a capture file. For example, most file formats don't record the number of dropped packets. See Capture Files for details.

The "Save Capture File As" Dialog Box

The "Save Capture File As" dialog box allows you to save the current capture to a file. The exact appearance of this dialog depends on your system. However, the functionality is the same across systems. Examples are shown below.

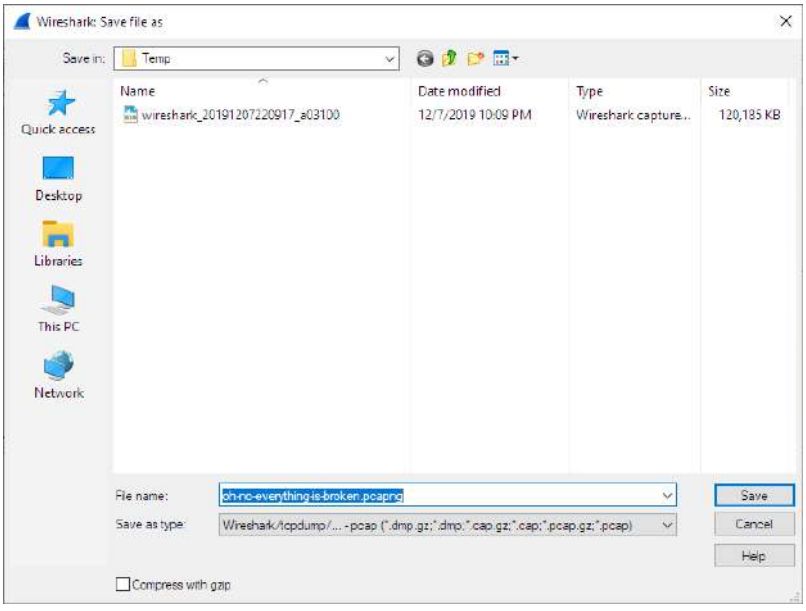


Figure 38. "Save" on Microsoft Windows

This is the common Windows file save dialog with some additional Wireshark extensions.

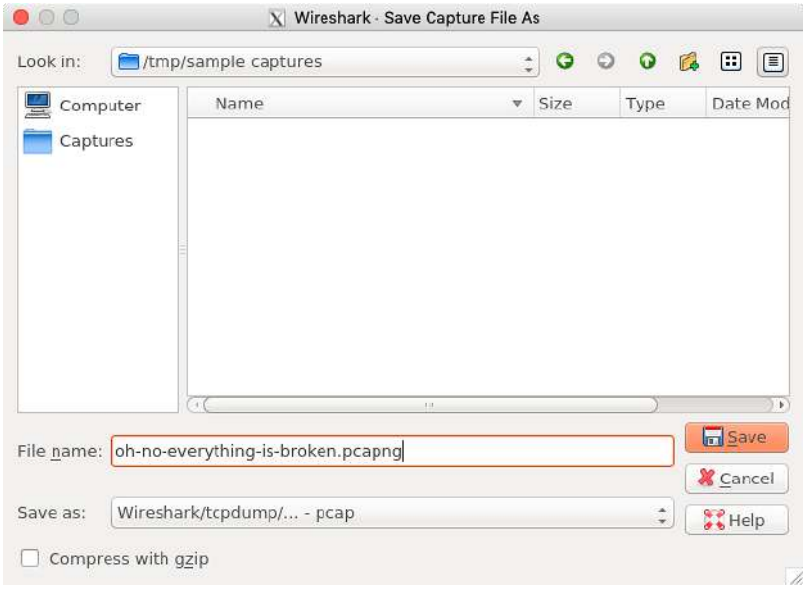


Figure 39. "Save" on Linux and UNIX

This is the common Qt file save dialog with additional Wireshark extensions.

You can perform the following actions:

- Type in the name of the file in which you wish to save the captured packets.
- Select the directory to save the file into.
- Specify the format of the saved capture file by clicking on the "Save as" drop-down box. You can choose from the types described in Output File Formats. Some capture formats may not be available depending on the packet types captured.
- The **[Help]** button will take you to this section of the "User's Guide".
- "Compress with gzip" will compress the capture file as it is being written to disk.
- Click the **[Save]** button to accept your selected file and save it.
- Click on the **[Cancel]** button to go back to Wireshark without saving any packets.

If you don't provide a file extension to the filename (e.g., `.pcap`) Wireshark will append the standard file extension for that file format.

TIP

Wireshark can convert file formats

You can convert capture files from one format to another by opening a capture and saving it as a different format.

If you wish to save some of the packets in your capture file you can do so via The "Export Specified Packets" Dialog Box.

Output File Formats

Wireshark can save the packet data in its native file format (pcapng) and in the file formats of other protocol analyzers so other tools can read the capture data.

NOTE	*Saving in a different format might lose data* Saving your file in a different format might lose information such as comments, name resolution, and time stamp resolution. See Time Stamps for more information on time stamps.

The following file formats can be saved by Wireshark (with the known file extensions):

- pcapng (*.pcapng). A flexible, extensible successor to the libpcap format. Wireshark 1.8 and later save files as pcapng by default. Versions prior to 1.8 used libpcap.
- pcap (*.pcap). The default format used by the *libpcap* packet capture library. Used by *tcpdump, _Snort*, *Nmap*, *Ntop*, and many other tools.
- Accellent 5Views (*.5vw)
- captures from HP-UX nettl ({asterisktrc0,*.trc1)
- Microsoft Network Monitor - NetMon (*.cap)
- Network Associates Sniffer - DOS (*.cap,*.enc,*.trc,*.fdc,*.syc)
- Cinco Networks NetXray captures (*.cap)
- Network Associates Sniffer - Windows (*.cap)
- Network Instruments/Viavi Observer (*.bfr)
- Novell LANalyzer (*.tr1)
- Oracle (previously Sun) snoop (*.snoop,*.cap)
- Visual Networks Visual UpTime traffic (*.*)
- Symbian OS btsnoop captures (*.log)
- Tamosoft CommView captures (*.ncf)
- Catapult (now Ixia/Keysight) DCT2000 .out files (*.out)
- Endace Measurement Systems' ERF format capture(*.erf)
- EyeSDN USB S0 traces (*.trc)
- Tektronix K12 text file format captures (*.txt)
- Tektronix K12xx 32bit .rf5 format captures (*.rf5)
- Android Logcat binary logs (*.logcat)
- Android Logcat text logs (*.*)
- Citrix NetScaler Trace files (*.cap)

New file formats are added from time to time.

Whether or not the above tools will be more helpful than Wireshark is a different question ;-)

NOTE

Third party protocol analyzers may require specific file extensions

Wireshark examines a file's contents to determine its type. Some other protocol analyzers only look at a file's extension. For example, you might need to use the `.cap` extension in order to open a file using the Windows version of *Sniffer*.

Merging Capture Files

Sometimes you need to merge several capture files into one. For example, this can be useful if you have captured simultaneously from multiple interfaces at once (e.g., using multiple instances of Wireshark).

There are three ways to merge capture files using Wireshark:

- Use the **File › Merge** menu to open the "Merge" dialog. See The "Merge With Capture File" Dialog Box for details. This menu item will be disabled unless you have loaded a capture file.
- Use *drag and drop* to drop multiple files on the main window. Wireshark will try to merge the packets in chronological order from the dropped files into a newly created temporary file. If you drop a single file, it will simply replace the existing capture.
- Use the `mergecap` tool from the command line to merge capture files. This tool provides the most options to merge capture files. See *mergecap*: Merging multiple capture files into one for details.

The "Merge With Capture File" Dialog Box

This lets you select a file to be merged into the currently loaded file. If your current data has not been saved you will be asked to save it first.

Most controls of this dialog will work the same way as described in the "Open Capture File" dialog box. See The "Open Capture File" Dialog Box for details.

Specific controls of this merge dialog are:

Prepend packets

Prepend the packets from the selected file before the currently loaded packets.

Merge chronologically

Merge both the packets from the selected and currently loaded file in chronological order.

Append packets

Append the packets from the selected file after the currently loaded packets.

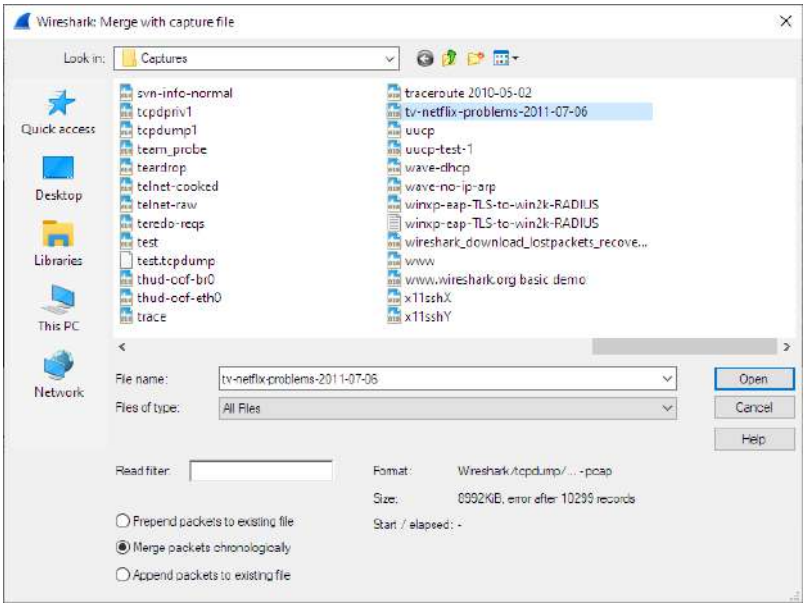


Figure 40. “Merge” on Microsoft Windows

This is the common Windows file open dialog with additional Wireshark extensions.

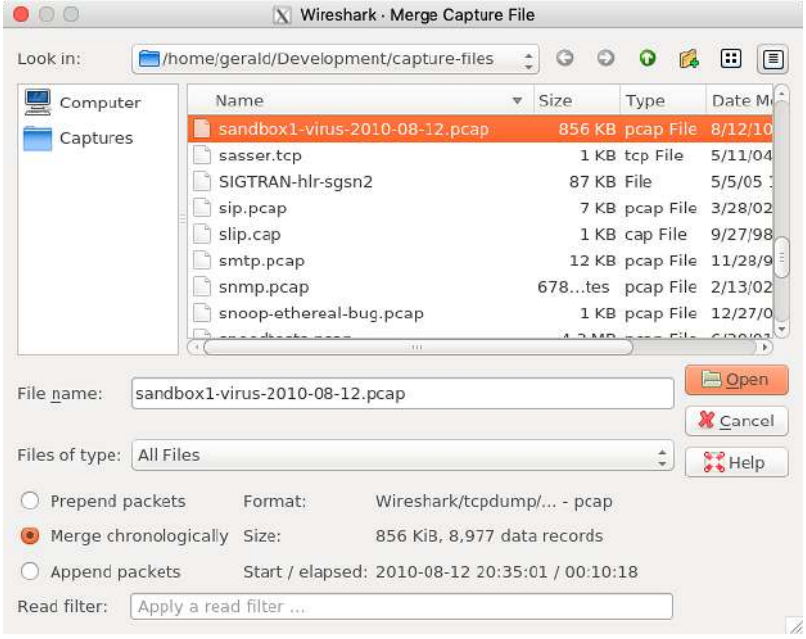


Figure 41. “Merge” on Linux and UNIX

This is the Qt file open dialog with additional Wireshark extensions.

Import Hex Dump

Wireshark can read in a hex dump and write the data described into a temporary libpcap capture file. It can read hex dumps with multiple packets in them, and build a capture file of multiple packets. It is also capable of generating dummy Ethernet, IP and UDP, TCP, or SCTP headers, in order to build fully processable packet dumps from hexdumps of application-level data only. Alternatively, a Dummy PDU header can be added to specify a dissector the data should be passed

to initially.

Two methods for converting the input are supported:

Standard ASCII Hexdumps

Wireshark understands a hexdump of the form generated by `od -Ax -tx1 -v`. In other words, each byte is individually displayed, with spaces separating the bytes from each other. Hex digits can be upper or lowercase.

In normal operation, each line must begin with an offset describing the position in the packet, followed a colon, space, or tab separating it from the bytes. There is no limit on the width or number of bytes per line, but lines with only hex bytes without a leading offset are ignored (i.e., line breaks should not be inserted in long lines that wrap.) Offsets are more than two digits; they are in hex by default, but can also be in octal or decimal. Each packet must begin with offset zero, and an offset zero indicates the beginning of a new packet. Offset values must be correct; an unexpected value causes the current packet to be aborted and the next packet start awaited. There is also a single packet mode with no offsets.

Packets may be preceded by a direction indicator ('I' or 'O') and/or a timestamp if indicated. If both are present, the direction indicator precedes the timestamp. The format of the timestamps must be specified. If no timestamp is parsed, in the case of the first packet the current system time is used, while subsequent packets are written with timestamps one microsecond later than that of the previous packet.

Other text in the input data is ignored. Any text before the offset is ignored, including email forwarding characters '>'. Any text on a line after the bytes is ignored, e.g., an ASCII character dump (but see **-a** to ensure that hex digits in the character dump are ignored). Any line where the first non-whitespace character is a '#' will be ignored as a comment. Any lines of text between the bytestring lines are considered preamble; the beginning of the preamble is scanned for the direction indicator and timestamp as mentioned above and otherwise ignored.

Any line beginning with #TEXT2PCAP is a directive and options can be inserted after this command to be processed by Wireshark. Currently there are no directives implemented; in the future, these may be used to give more fine-grained control on the dump and the way it should be processed e.g., timestamps, encapsulation type etc.

In general, short of these restrictions, Wireshark is pretty liberal about reading in hexdumps and has been tested with a variety of mangled outputs (including being forwarded through email multiple times, with limited line wrap etc.)

Here is a sample dump that can be imported, including optional directional indicator and timestamp:

```
I 2019-05-14T19:04:57Z
000000 00 e0 1e a7 05 6f 00 10 ........
000008 5a a0 b9 12 08 00 46 00 ........
000010 03 68 00 00 00 00 0a 2e ........
000018 ee 33 0f 19 08 7f 0f 19 ........
000020 03 80 94 04 00 00 10 01 ........
000028 16 a2 0a 00 03 50 00 0c ........
000030 01 01 0f 19 03 80 11 01 ........
```

Regular Text Dumps

Wireshark is also capable of scanning the input using a custom Perl regular expression as specified by GLib's GRegex here. Using a regex capturing a single packet in the given file Wireshark will search the given file from start to the second to last character (the last character has to be `\n` and is ignored) for non-overlapping (and non-empty) strings matching the given regex and then identify the fields to import using named capturing subgroups. Using provided format information for each field they are then decoded and translated into a standard libpcap file retaining packet order.

Note that each named capturing subgroup has to match *exactly* once a packet, but they may be present multiple times in the regex.

For example, the following dump:

```
> 0:00:00.265620 a130368b000000080060
> 0:00:00.280836 a1216c8b00000000000089086b0b82020407
< 0:00:00.295459 a201080000000000000000800000000
> 0:00:00.296982 a1303c8b000000008007088286b0bc1ffcbf0f9ff
> 0:00:00.305644 a121718b0000000000008ba86a0b8008
< 0:00:00.319061 a2010900000000000000001000600000
> 0:00:00.330937 a130428b000000008007589186b0bb9ffd9f0fdfa3eb4295e99f3aaffd2f005
> 0:00:00.356037 a121788b0000000000008a18
```

could be imported using these settings:

```
regex: ^(?<dir>[<>])\s(?<time>\d+:\d\d:\d\d.\d+)\s(?<data>[0-9a-fA-F]+)$
timestamp: %H:%M:%S.%f
dir: in: <   out: >
encoding: HEX
```

Caution has to be applied when discarding the anchors `^` and `$`, as the input is searched, not parsed, meaning even most incorrect regexes will produce valid looking results when not anchored (however, anchors are not guaranteed to prevent this). It is generally recommended to sanity check any files created using this conversion.

Supported fields:

- data: Actual captured frame data

  The only mandatory field. This should match the encoded binary data captured and is used as the actual frame data to import.

- time: timestamp for the packet

  The captured field will be parsed according to the given timestamp format into a timestamp.

  If no timestamp is present an arbitrary counter will count up seconds and nanoseconds by one each packet.

- dir: the direction the packet was sent over the wire

  The captured field is expected to be one character in length, any remaining characters are ignored (e.g., given "Input" only the 'I' is looked at). This character is compared to lists of characters corresponding to inbound and outbound and the packet is assigned the corresponding direction. If neither list yields a match, the direction is set to unknown.

  If this field is not specified the entire file has no directional information.

- seqno: an ID for this packet

  Each packet can be assigned an arbitrary ID that can used as field by Wireshark. This field is assumed to be a positive integer base 10. This field can e.g. be used to reorder out of order captures after the import.

  If this field is not given, no IDs will be present in the resulting file.

The “Import From Hex Dump” Dialog Box

This dialog box lets you select a text file, containing a hex dump of packet data, to be imported and set import parameters.

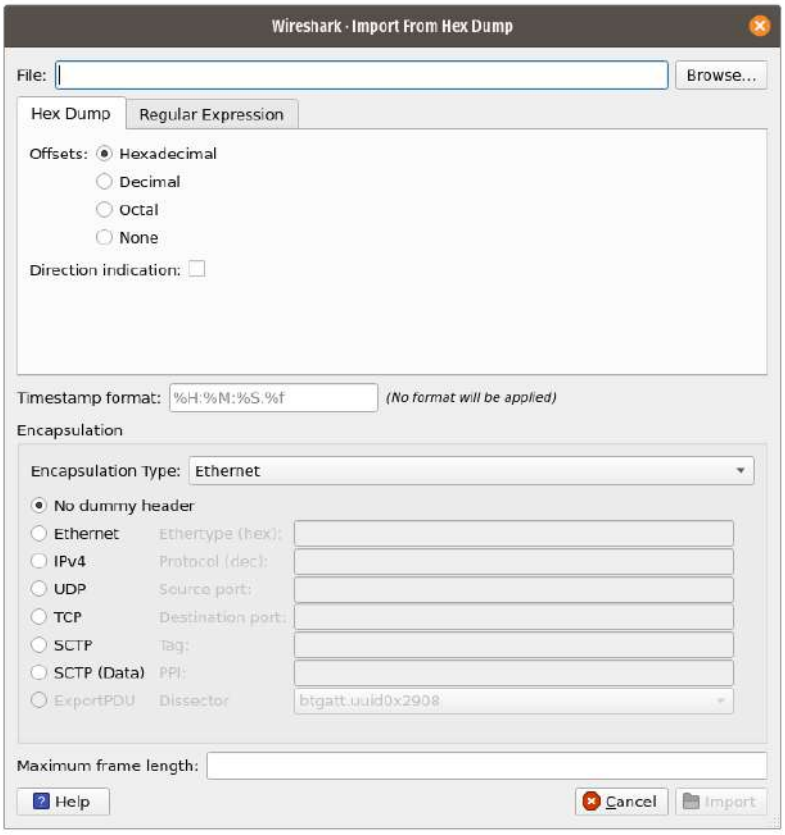


Figure 42. The "Import from Hex Dump" dialog in Hex Dump mode

Specific controls of this import dialog are split in three sections:

File Source

Determine which input file has to be imported

Input Format

Determine how the input file has to be interpreted.

Encapsulation

Determine how the data is to be encapsulated.

File source

Filename / Browse

Enter the name of the text file to import. You can use *Browse* to browse for a file.

Input Format

This section is split in the two alternatives for input conversion, accessible in the two Tabs "Hex Dump" and "Regular Expression"

In addition to the conversion mode specific inputs, there are also common parameters, currently only the timestamp format.

The Hex Dump tab

Offsets

Select the radix of the offsets given in the text file to import. This is usually hexadecimal, but decimal and octal are also supported. Select *None* when only the bytes are present. These will be imported as a single packet.

Direction indication

Tick this box if the text file to import has direction indicators before each frame. These are on a separate line before each frame and start with either *I* or *i* for input and *O* or *o* for output.

The Regular Expression tab

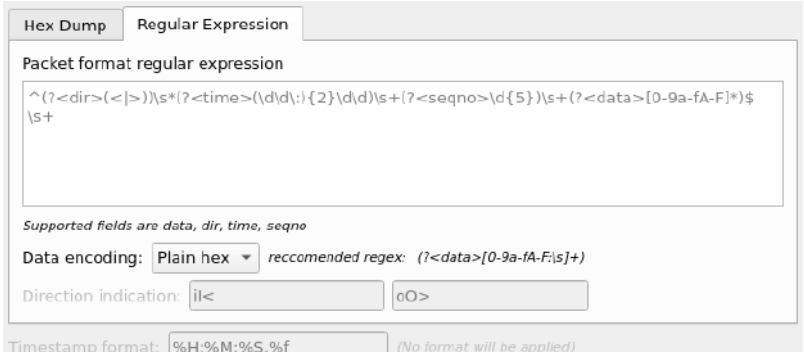


Figure 43. The "Regular Expression" tab inside the "Import from Hex Dump" dialog.

Packet format regular expression

This is the regex used for searching packets and metadata inside the input file. Named capturing subgroups are used to find the individual fields. Anchors `^` and `$` are set to match directly before and after newlines `\n` or `\r\n`. See GRegex for a full documentation.

Data encoding

The Encoding used for the binary data. Supported encodings are plain-hexadecimal, -octal, -binary and base64. Plain here means no additional characters are present in the data field beyond whitespaces, which are ignored. Any unexpected characters abort the import process.

Ignored whitespaces are `\r`, `\n`, `\t`, `\v`, `` ` `` and only for hex `:`, only for base64 `=`.

Any incomplete bytes at the field's end are assumed to be padding to fill the last complete byte. These bits should be zero, however, this is not checked.

Direction indication

The lists of characters indicating incoming vs. outgoing packets. These fields are only available when the regex contains a `(?<dir>⋯)` group.

Common items

Timestamp Format

This is the format specifier used to parse the timestamps in the text file to import. It uses the same format as `strptime(3)` with the addition of `%f` for zero padded fractions of seconds. The precision of `%f` is determined from its length. The most common fields are `%H`, `%M` and `%S` for hours, minutes and seconds. The straightforward HH:MM:SS format is covered by %T. For a full definition of the syntax look for `strptime(3)`,

In Regex mode this field is only available when a `(?<time>⋯)` group is present.

In Hex Dump mode if there are no timestamps in the text file to import, leave this field empty and timestamps will be generated based on the time of import.

Encapsulation

Encapsulation type

Here you can select which type of frames you are importing. This all depends on from what type of medium the dump to import was taken. It lists all types that Wireshark understands, so as to pass the capture file contents to the right dissector.

Dummy header

When Ethernet encapsulation is selected you have to option to prepend dummy headers to the frames to import. These headers can provide artificial Ethernet, IP, UDP, TCP or SCTP headers or SCTP data chunks. When selecting a type of dummy header, the applicable entries are enabled, others are greyed out and default values are used. When the *Wireshark Upper PDU export* encapsulation is selected the option *ExportPDU* becomes available. This allows you to select the name of the dissector these frames are to be directed to.

Maximum frame length

You may not be interested in the full frames from the text file, just the first part. Here you can define how much data from the start of the frame you want to import. If you leave this open the maximum is set to 256kiB.

Once all input and import parameters are setup click **[Import]** to start the import. If your current data wasn't saved before you will be asked to save it first.

If the import button doesn't unlock, make sure all encapsulation parameters are in the expected range and all unlocked fields are populated when using regex mode (the placeholder text is not used as default).

When completed there will be a new capture file loaded with the frames imported from the text file.

File Sets

When using the "Multiple Files" option while doing a capture (see: Capture files and file modes), the

capture data is spread over several capture files, called a file set.

As it can become tedious to work with a file set by hand, Wireshark provides some features to handle these file sets in a convenient way.

How does Wireshark detect the files of a file set?

A filename in a file set uses the format Prefix_Number_DateTimeSuffix which might look something like `test_00001_20220714183910.pcap`. All files of a file set share the same prefix (e.g., "test") and suffix (e.g., ".pcap") and a varying middle part.

To find the files of a file set, Wireshark scans the directory where the currently loaded file resides and checks for files matching the filename pattern (prefix and suffix) of the currently loaded file.

This simple mechanism usually works well but has its drawbacks. If several file sets were captured with the same prefix and suffix, Wireshark will detect them as a single file set. If files were renamed or spread over several directories the mechanism will fail to find all files of a set.

The following features in the **File › File Set** submenu are available to work with file sets in a convenient way:

- The "List Files" dialog box will list the files Wireshark has recognized as being part of the current file set.
- **[Next File]** closes the current and opens the next file in the file set.
- **[Previous File]** closes the current and opens the previous file in the file set.

The "List Files" Dialog Box

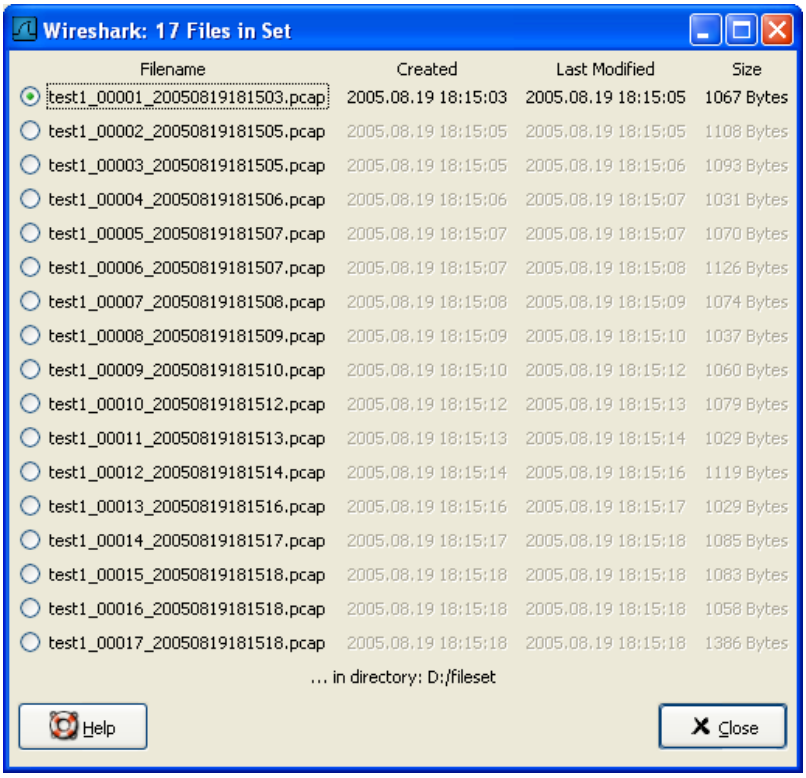


Figure 44. The “List Files” dialog box

Each line contains information about a file of the file set:

Filename

The name of the file. If you click on the filename (or the radio button left to it), the current file will be closed and the corresponding capture file will be opened.

Created

The creation time of the file.

Last Modified

The last time the file was modified.

Size

The size of the file.

The last line will contain info about the currently used directory where all of the files in the file set can be found.

The content of this dialog box is updated each time a capture file is opened/closed.

The **[Close]** button will, well, close the dialog box.

Exporting Data

Wireshark provides a variety of options for exporting packet data. This section describes general

ways to export data from the main Wireshark application. There are many other ways to export or extract data from capture files, including processing tshark output and customizing Wireshark and TShark using Lua scripts.

The “Export Specified Packets” Dialog Box

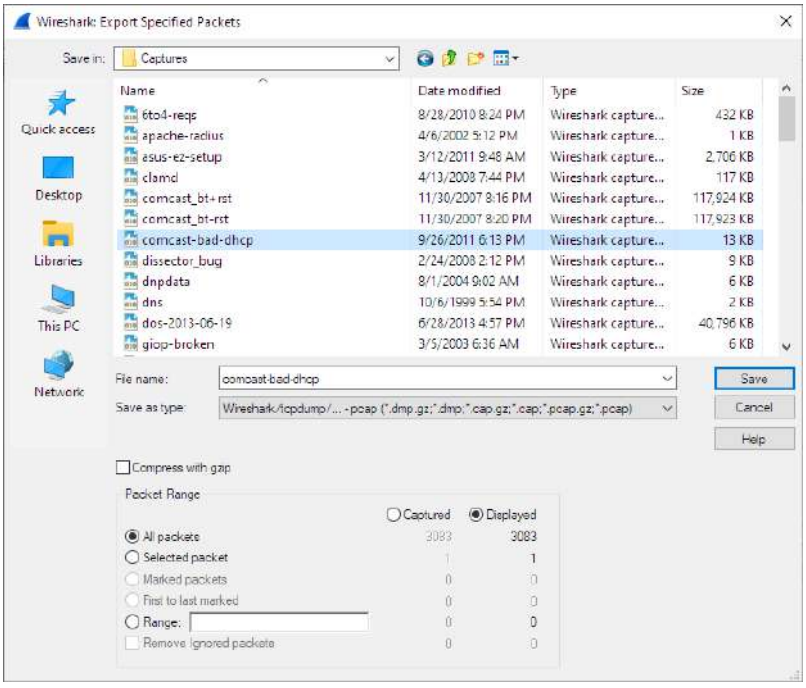


Figure 45. The “Export Specified Packets” dialog box

This is similar to the “Save” dialog box, but it lets you save specific packets. This can be useful for trimming irrelevant or unwanted packets from a capture file. See Packet Range for details on the range controls.

The “Export Packet Dissections” Dialog Box

This lets you save the packet list, packet details, and packet bytes as plain text, CSV, JSON, and other formats.

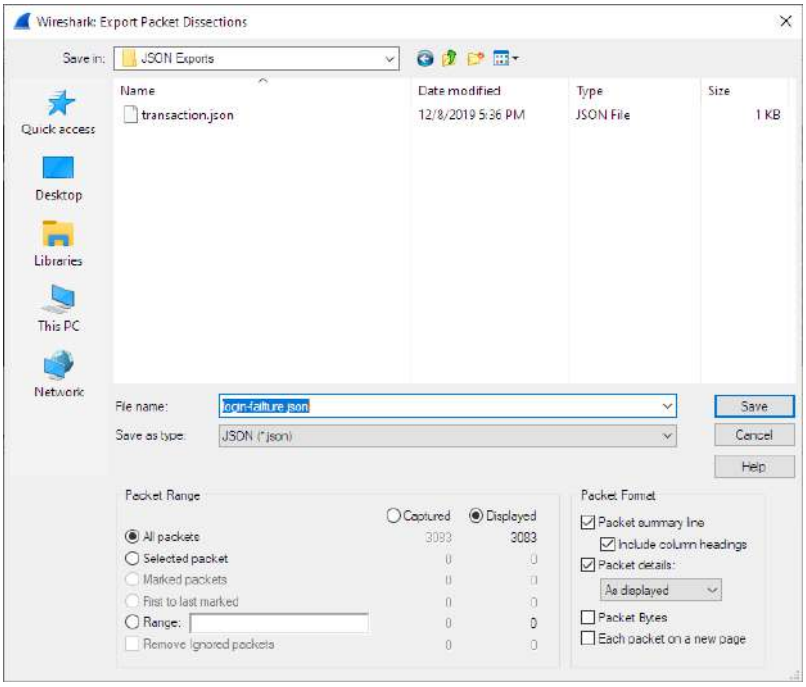


Figure 46. The “Export Packet Dissections” dialog box

The format can be selected from the “Export As” drop-down and further customized using the “Packet Range” and “Packet Format” controls. Some controls are unavailable for some formats, notably CSV and JSON. The following formats are supported:

- Plain text as shown in the main window
- Comma-separated values (CSV)
- C-compatible byte arrays
- PSML (summary XML)
- PDML (detailed XML)
- JavaScript Object Notation (JSON)

Here are some examples of exported data:

Plain text

```
No.     Time           Source                Destination           Protocol Length
SSID       Info
      1 0.000000       200.121.1.131         172.16.0.122          TCP      1454
10554 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=1400 [TCP segment of a reassembled PDU]

Frame 1: 1454 bytes on wire (11632 bits), 1454 bytes captured (11632 bits)
Ethernet II, Src: 00:50:56:c0:00:01, Dst: 00:0c:29:42:12:13
Internet Protocol Version 4, Src: 200.121.1.131 (200.121.1.131), Dst: 172.16.0.122
(172.16.0.122)
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1440
    Identification: 0x0141 (321)
    Flags: 0x0000
    ...0 0000 0000 0000 = Fragment offset: 0
    Time to live: 106
    Protocol: TCP (6)
    Header checksum: 0xd390 [validation disabled]
    [Header checksum status: Unverified]
    Source: 200.121.1.131 (200.121.1.131)
    Destination: 172.16.0.122 (172.16.0.122)
    [Source GeoIP: PE, ASN 6147, Telefonica del Peru S.A.A.]
Transmission Control Protocol, Src Port: 10554, Dst Port: 80, Seq: 1, Ack: 1, Len:
1400
```

TIP

If you would like to be able to import any previously exported packets from a plain text file it is recommended that you do the following:

- Add the “Absolute date and time” column.
- Temporarily hide all other columns.
- Disable the **Edit › Preferences › Protocols › Data** “Show not dissected data on new Packet Bytes pane” preference. More details are provided in Preferences
- Include the packet summary line.
- Exclude column headings.
- Exclude packet details.
- Include the packet bytes.

CSV

```
"No.","Time","Source","Destination","Protocol","Length","SSID","Info","Win Size"
"1","0.000000","200.121.1.131","172.16.0.122","TCP","1454","","10554  >  80 [ACK]
Seq=1 Ack=1 Win=65535 Len=1400 [TCP segment of a reassembled PDU]","65535"
"2","0.000011","172.16.0.122","200.121.1.131","TCP","54","","[TCP ACKed unseen
segment] 80  >  10554 [ACK] Seq=1 Ack=11201 Win=53200 Len=0","53200"
"3","0.025738","200.121.1.131","172.16.0.122","TCP","1454","","[TCP Spurious
Retransmission] 10554  >  80 [ACK] Seq=1401 Ack=1 Win=65535 Len=1400 [TCP segment of a
reassembled PDU]","65535"
"4","0.025749","172.16.0.122","200.121.1.131","TCP","54","","[TCP Window Update] [TCP
ACKed unseen segment] 80  >  10554 [ACK] Seq=1 Ack=11201 Win=63000 Len=0","63000"
"5","0.076967","200.121.1.131","172.16.0.122","TCP","1454","","[TCP Previous segment
not captured] [TCP Spurious Retransmission] 10554  >  80 [ACK] Seq=4201 Ack=1
Win=65535 Len=1400 [TCP segment of a reassembled PDU]","65535"
```

JSON


```
{
    "_index": "packets-2014-06-22",
    "_type": "doc",
    "_score": null,
    "_source": {
      "layers": {
        "frame": {
          "frame.encap_type": "1",
          "frame.time": "Jun 22, 2014 13:29:41.834477000 PDT",
          "frame.offset_shift": "0.000000000",
          "frame.time_epoch": "1403468981.834477000",
          "frame.time_delta": "0.450535000",
          "frame.time_delta_displayed": "0.450535000",
          "frame.time_relative": "0.450535000",
          "frame.number": "2",
          "frame.len": "86",
          "frame.cap_len": "86",
          "frame.marked": "0",
          "frame.ignored": "0",
          "frame.protocols": "eth:ethertype:ipv6:icmpv6",
          "frame.coloring_rule.name": "ICMP",
          "frame.coloring_rule.string": "icmp || icmpv6"
        },
        "eth": {
          "eth.dst": "33:33:ff:9e:e3:8e",
          "eth.dst_tree": {
            "eth.dst_resolved": "33:33:ff:9e:e3:8e",
            "eth.dst.oui": "3355647",
            "eth.addr": "33:33:ff:9e:e3:8e",
```

```
            "eth.addr_resolved": "33:33:ff:9e:e3:8e",
            "eth.addr.oui": "3355647",
            "eth.dst.lg": "1",
            "eth.lg": "1",
            "eth.dst.ig": "1",
            "eth.ig": "1"
          },
          "eth.src": "00:01:5c:62:8c:46",
          "eth.src_tree": {
            "eth.src_resolved": "00:01:5c:62:8c:46",
            "eth.src.oui": "348",
            "eth.src.oui_resolved": "Cadant Inc.",
            "eth.addr": "00:01:5c:62:8c:46",
            "eth.addr_resolved": "00:01:5c:62:8c:46",
            "eth.addr.oui": "348",
            "eth.addr.oui_resolved": "Cadant Inc.",
            "eth.src.lg": "0",
            "eth.lg": "0",
            "eth.src.ig": "0",
            "eth.ig": "0"
          },
          "eth.type": "0x000086dd"
        },
        "ipv6": {
          "ipv6.version": "6",
          "ip.version": "6",
          "ipv6.tclass": "0x00000000",
          "ipv6.tclass_tree": {
            "ipv6.tclass.dscp": "0",
            "ipv6.tclass.ecn": "0"
          },
          "ipv6.flow": "0x00000000",
          "ipv6.plen": "32",
          "ipv6.nxt": "58",
          "ipv6.hlim": "255",
          "ipv6.src": "2001:558:4080:16::1",
          "ipv6.addr": "2001:558:4080:16::1",
          "ipv6.src_host": "2001:558:4080:16::1",
          "ipv6.host": "2001:558:4080:16::1",
          "ipv6.dst": "ff02::1:ff9e:e38e",
          "ipv6.addr": "ff02::1:ff9e:e38e",
          "ipv6.dst_host": "ff02::1:ff9e:e38e",
          "ipv6.host": "ff02::1:ff9e:e38e",
          "ipv6.geoip.src_summary": "US, ASN 7922, Comcast Cable Communications, LLC",
          "ipv6.geoip.src_summary_tree": {
            "ipv6.geoip.src_country": "United States",
            "ipv6.geoip.country": "United States",
            "ipv6.geoip.src_country_iso": "US",
```

```
            "ipv6.geoip.country_iso": "US",
            "ipv6.geoip.src_asnum": "7922",
            "ipv6.geoip.asnum": "7922",
            "ipv6.geoip.src_org": "Comcast Cable Communications, LLC",
            "ipv6.geoip.org": "Comcast Cable Communications, LLC",
            "ipv6.geoip.src_lat": "37.751",
            "ipv6.geoip.lat": "37.751",
            "ipv6.geoip.src_lon": "-97.822",
            "ipv6.geoip.lon": "-97.822"
          }
        },
        "icmpv6": {
          "icmpv6.type": "135",
          "icmpv6.code": "0",
          "icmpv6.checksum": "0x00005b84",
          "icmpv6.checksum.status": "1",
          "icmpv6.reserved": "00:00:00:00",
          "icmpv6.nd.ns.target_address": "2001:558:4080:16:be36:e4ff:fe9e:e38e",
          "icmpv6.opt": {
            "icmpv6.opt.type": "1",
            "icmpv6.opt.length": "1",
            "icmpv6.opt.linkaddr": "00:01:5c:62:8c:46",
            "icmpv6.opt.src_linkaddr": "00:01:5c:62:8c:46"
          }
        }
      }
    }
  }
]
```


The “Export Selected Packet Bytes” Dialog Box

Export the bytes selected in the “Packet Bytes” pane into a raw binary file.

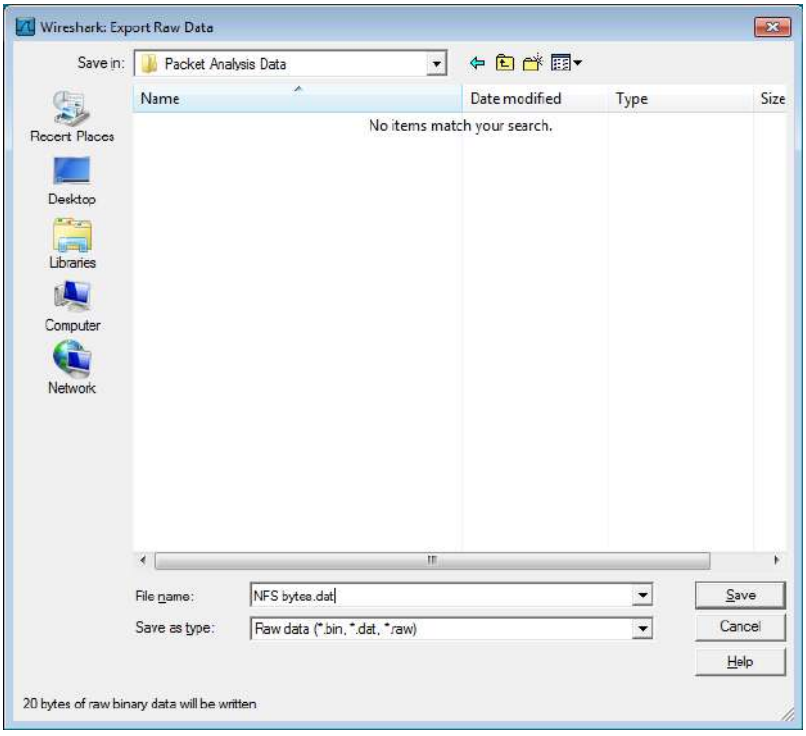


Figure 47. The “Export Selected Packet Bytes” dialog box

File name

The file name to export the packet data to.

Save as type

The file extension.

The “Export PDUs to File…” Dialog Box

The “Export PDUs to File…” dialog box allows you to filter the captured Protocol Data Units (PDUs) and export them into the file. It allows you to export reassembled PDUs avoiding lower layers such as HTTP without TCP, and decrypted PDUs without the lower protocols such as HTTP without TLS and TCP.

1. In the main menu select **File › Export PDUs to File…**. Wireshark will open a corresponding dialog Export PDUs to File window.

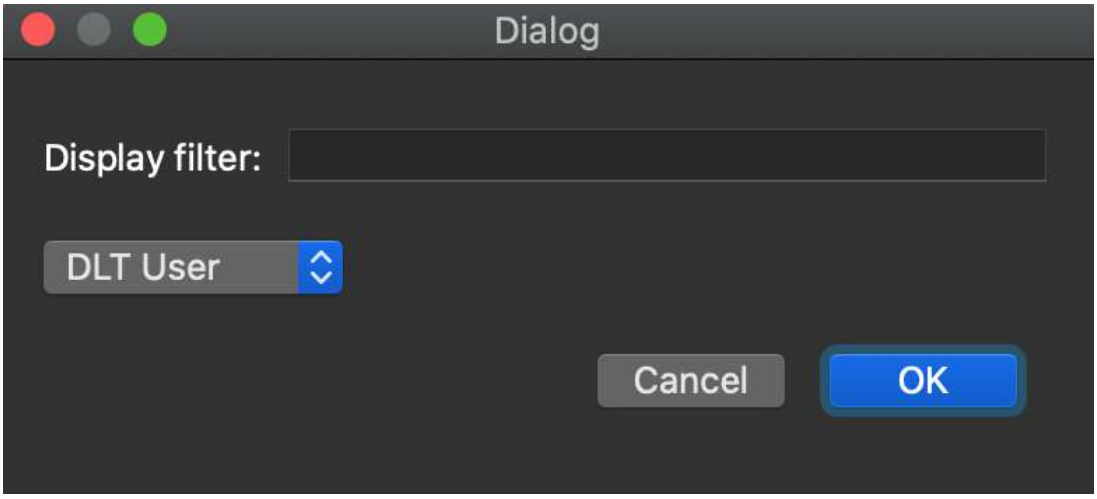


Figure 48. Export PDUs to File window

2. To select the data according to your needs, optionally type a filter value into the `Display Filter` field. For more information about filter syntax, see the Wireshark Filters man page.
3. In the field below the `Display Filter` field you can choose the level from which you want to export the PDUs to the file. There are seven levels:
   a. `DLT User`. You can export a protocol, which is framed in the user data link type table without the need to reconfigure the DLT user table. For more information, see the How to Dissect Anything page.
   b. `DVB-CI`. You can use it for the Digital Video Broadcasting (DVB) protocol.
   c. `Logcat` and `Logcat Text`. You can use them for the Android logs.
   d. `OSI layer 3`. You can use it to export PDUs encapsulated in the IPSec or SCTP protocols.
   e. `OSI layer 4`. You can use it to export PDUs encapsulated in the TCP or UDP protocols.
   f. `OSI layer 7`. You can use it to export the following protocols: CredSSP over TLS, Diameter, protocols encapsulated in TLS and DTLS, H.248, Megaco, RELOAD framing, SIP, SMPP.

NOTE	As a developer you can add any dissector to the existing list or define a new entry in the list by using the functions in `epan/exported_pdu.h`.

4. To finish exporting PDUs to file, click the **[OK]** button in the bottom-right corner. This will close the originally captured file and open the exported results instead as a temporary file in the main Wireshark window.
5. You may save the temporary file just like any captured file. See Saving Captured Packets for details.

NOTE	The file produced has a `Wireshark Upper PDU` encapsulation type that has somewhat limited support outside of Wireshark, but is very flexible and can contain PDUs for any protocol for which there is a Wireshark dissector.

The “Strip Headers…” Dialog Box

The “Strip Headers…” dialog box allows you to filter known encapsulation types on whatever protocol layer they appear and export them into a new capture file, removing lower-level protocols. It allows you to export reassembled packets and frames without lower layers such as GPF, GRE, GSE, GTP-U, MPLS, MPE, PPP, and more. If Wireshark has performed decryption, then you can export decrypted IP from protocols like IEEE 802.11 or IPSec without having to save encryption keys.

The procedure is similar to that of The “Export PDUs to File…” Dialog Box:

1. In the main menu select **File › Strip Headers…**. Wireshark will open a corresponding dialog.
2. To select the data according to your needs, optionally type a filter value into the `Display Filter` field. For more information about filter syntax, see the Wireshark Filters man page.
3. In the field below the `Display Filter` field you can choose the encapsulation type you want to find and export to the file. There are two encapsulations supported:
   a. `Ethernet`. You can use it to export Ethernet encapsulated in other protocols.
   b. `IP`. You can use it to export IPv4 and IPv6 encapsulated in other protocols.

   > **NOTE** As a developer you can add encapsulations to the list by using the functions in `epan/exported_pdu.h`.

4. To finish exporting to file, click the **[OK]** button in the bottom-right corner. This will close the originally captured file and open the exported results instead as a temporary file in the main Wireshark window.
5. You may save the temporary file just like any captured file. See Saving Captured Packets for details.

   > **NOTE** The new capture files produced have standard encapsulation types and can be read in nearly any tool.

The “Export TLS Session Keys…” Dialog Box

Transport Layer Security (TLS) encrypts the communication between a client and a server. The most common use for it is web browsing via HTTPS.

Decryption of TLS traffic requires TLS secrets. You can get them in the form of stored session keys in a "key log file", or by using an RSA private key file. For more details, see the TLS wiki page.

The **File › Export TLS Session Keys…** menu option generates a new "key log file" which contains TLS session secrets known by Wireshark. This feature is useful if you typically decrypt TLS sessions using the RSA private key file. The RSA private key is very sensitive because it can be used to decrypt other TLS sessions and impersonate the server. Session keys can be used only to decrypt

sessions from the packet capture file. However, session keys are the preferred mechanism for sharing data over the Internet.

To export captured TLS session keys, follow the steps below:

1. In the main menu select **File › Export TLS Session Keys...**. Wireshark will open a corresponding dialog Export TLS Session Keys window.

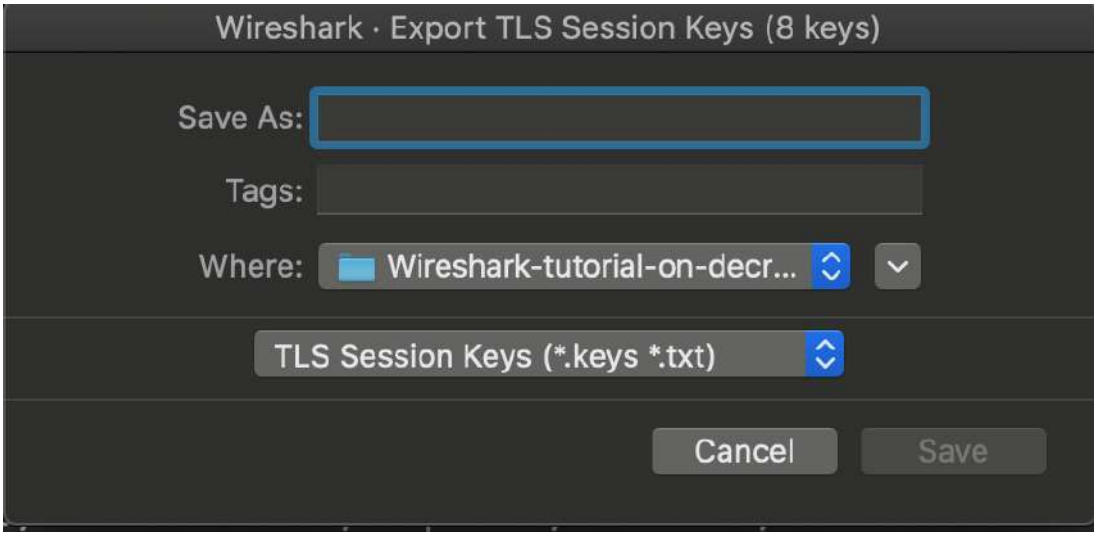


Figure 49. Export TLS Session Keys window

2. Type the desired file name in the `Save As` field.
3. Choose the destination folder for your file in the `Where` field.
4. Press the **[Save]** button to complete the export file procedure.

The "Export Objects" Dialog Box

This feature scans through the selected protocol's streams in the currently open capture file or running capture and allows the user to export reassembled objects to the disk. For example, if you select HTTP, you can export HTML documents, images, executables, and any other files transferred over HTTP to the disk. If you have a capture running, this list is automatically updated every few seconds with any new objects seen. The saved objects can then be opened or examined independently of Wireshark.

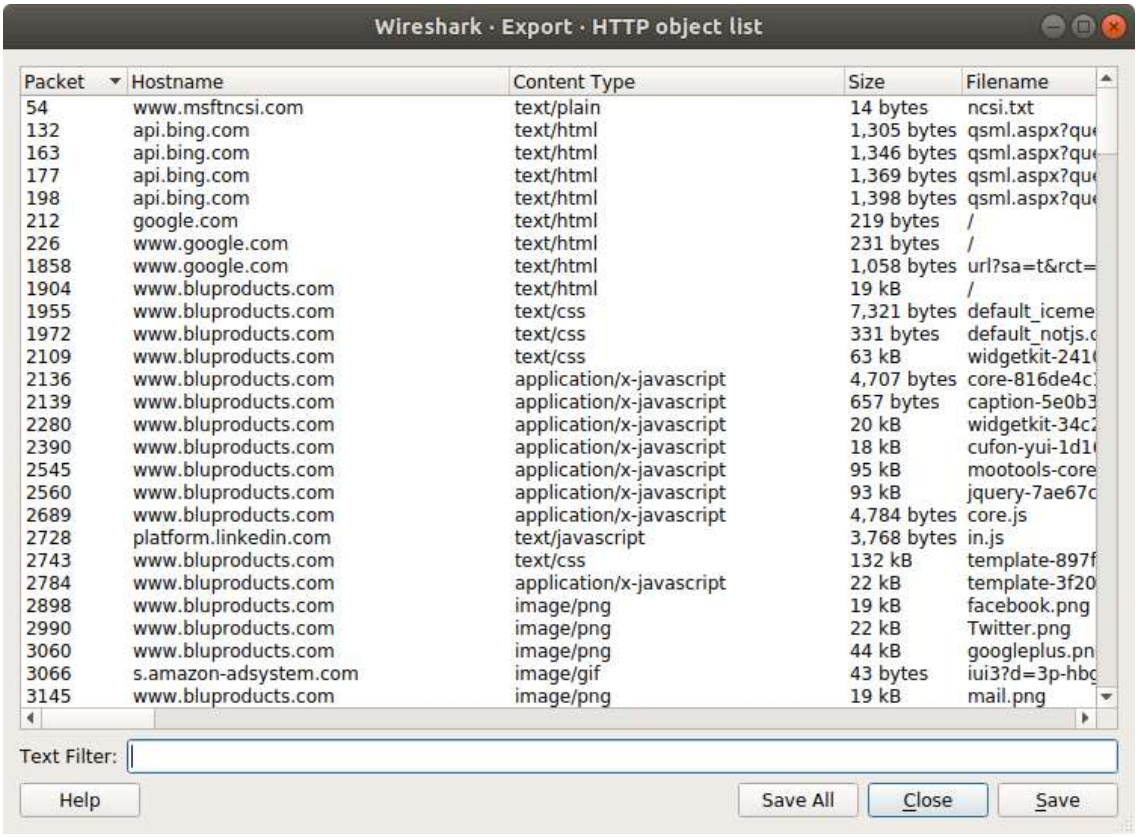


Figure 50. The "Export Objects" dialog box

Columns:

Packet

The packet number in which this object was found. In some cases, there can be multiple objects in the same packet.

Hostname

The hostname of the server that sent this object.

Content Type

The content type of this object.

Size

The size of this object in bytes.

Filename: The filename for this object. Each protocol generates the filename differently. For example, HTTP uses the final part of the URI and IMF uses the subject of the email.

Inputs:

Text Filter

Only displays objects containing the specified text string.

Help

Opens this section of the “User’s Guide”.

Save All

Saves all objects (including those not displayed) using the filename from the filename column. You will be asked what directory or folder to save them in.

Close

Closes the dialog without exporting.

Save

Saves the currently selected object as a filename you specify. The default filename to save as is taken from the filename column of the objects list.

Printing Packets

To print packets, select the **File › Print…** menu item. Wireshark will display the “Print” dialog box as shown below.

WARNING	*It’s easy to waste paper doing this* Printed output can contain lots of text, particularly if you print packet details and bytes.

The “Print” Dialog Box

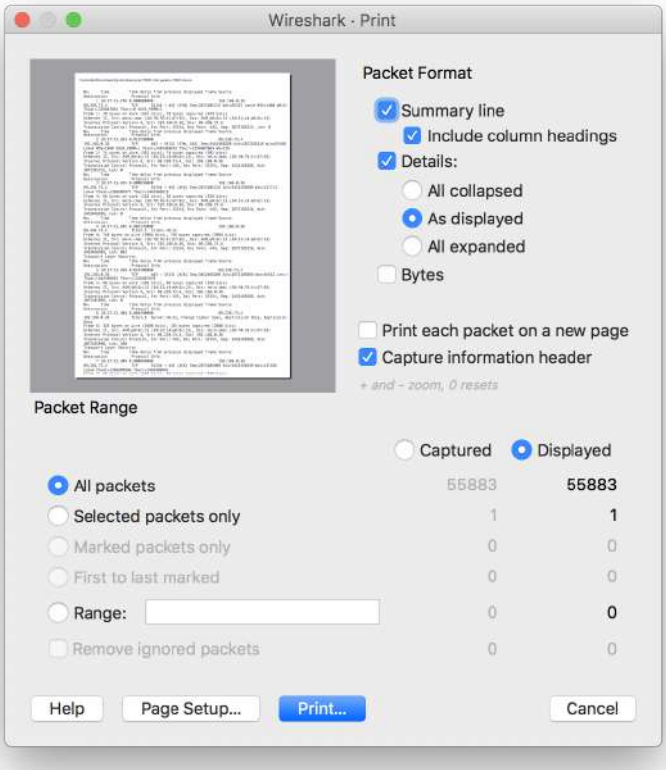


Figure 51. The "Print" dialog box

The "Print" dialog box shows a preview area which shows the result of changing the packet format settings. You can zoom in and out using the `+` and `-` keys and reset the zoom level using the `0` key. The following settings are available in the Print dialog box:

Packet Format

Lets you specify what gets printed. See The "Packet Format" frame for details.

Summary line

Include a summary line for each packet. The line will contain the same fields as the packet list.

Details

Print details for each packet.

Bytes

Print a hex dump of each packet.

Packet Range

Select the packets to be printed. See The "Packet Range" Frame for details.

[Page Setup...] lets you select the page size and orientation.

[Print...] prints to your default printer.

[Cancel] will close the dialog without printing.

[Help] will display this section of the "User's Guide".

The "Packet Range" Frame

The packet range frame is a part of the "Export Specified Packets," "Export Packet Dissections," and "Print" dialog boxes. You can use it to specify which packets will be exported or printed.

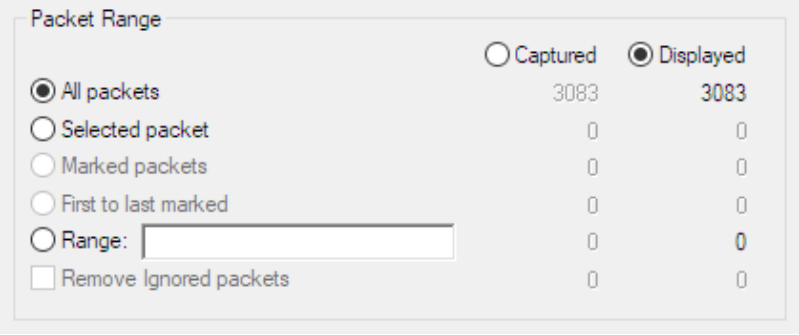


Figure 52. The "Packet Range" frame

By default, the **[Displayed]** button is set, which only exports or prints the packets that match the current display filter. Selecting **[Captured]** will export or print all packets. You can further limit what you export or print to the following:

All packets

All captured or displayed packets depending on the primary selection above.

Selected packet

Only the selected packet.

Marked packets

Only marked packets. See Marking Packets.

First to last marked

Lets you mark an inclusive range of packets.

Range

Lets you manually specify a range of packets, e.g., *5,10-15,20-* will process the packet number five, the packets from packet number ten to fifteen (inclusive) and every packet from number twenty to the end of the capture.

Remove ignored packets

Don't export or print ignored packets. See Ignoring Packets.

The Packet Format Frame

The packet format frame is also a part of the “Export Packet Dissections” and “Print” dialog boxes. You can use it to specify which parts of dissection are exported or printed.

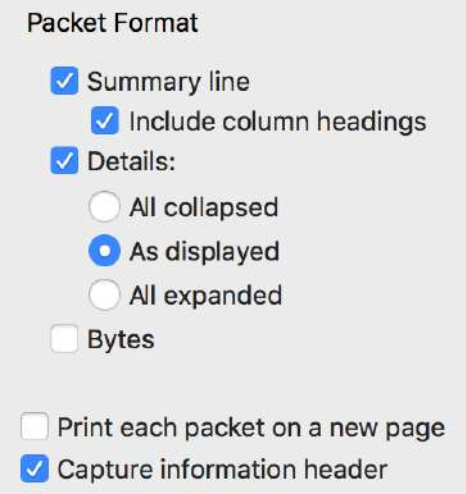


Figure 53. The “Packet Format” frame

Each of the settings below correspond to the packet list, packet detail, and packet bytes in the main window.

Packet summary line

Export or print each summary line as shown in the “Packet List” pane.

Packet details

Export or print the contents of the “Packet Details” tree.

All collapsed

Export or print as if the “Packet Details” tree is in the “all collapsed” state.

As displayed

Export or print as if the “Packet Details” tree is in the “as displayed” state.

All expanded

Export or print as if the “Packet Details” tree is in the “all expanded” state.

Packet Bytes

Export or print the contents of the “Packet Bytes” pane.

Each packet on a new page

For printing and some export formats, put each packet on a separate page. For example, when exporting to a text file this will put a form feed character between each packet.

Capture information header

Add a header to each page with capture filename and the number of total packets and shown packets.

Working With Captured Packets

Viewing Packets You Have Captured

Once you have captured some packets or you have opened a previously saved capture file, you can view the packets that are displayed in the packet list pane by simply clicking on a packet in the packet list pane, which will bring up the selected packet in the tree view and byte view panes.

You can then expand any part of the tree to view detailed information about each protocol in each packet. Clicking on an item in the tree will highlight the corresponding bytes in the byte view. An example with a TCP packet selected is shown in Wireshark with a TCP packet selected for viewing. It also has the Acknowledgment number in the TCP header selected, which shows up in the byte view as the selected bytes.

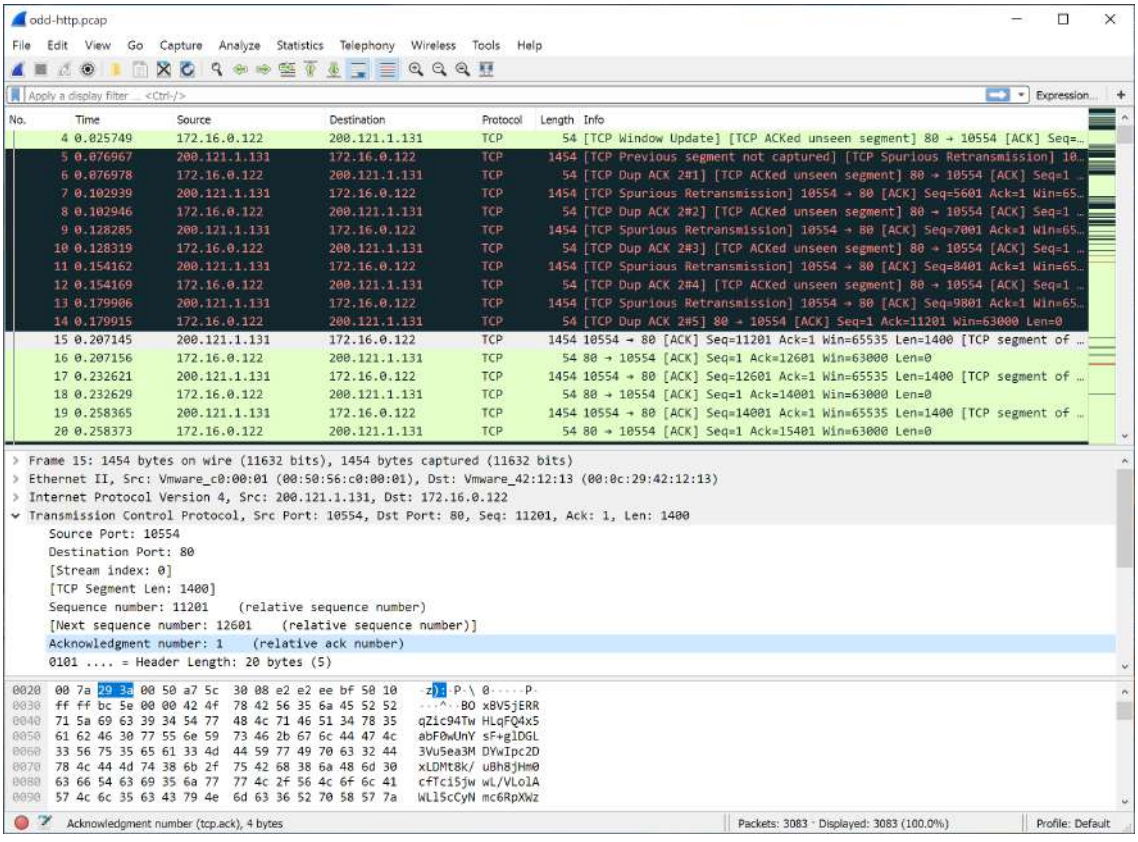


Figure 54. Wireshark with a TCP packet selected for viewing

You can also select and view packets the same way while Wireshark is capturing if you selected “Update list of packets in real time” in the “Capture Preferences” dialog box.

In addition you can view individual packets in a separate window as shown in Viewing a packet in a separate window. You can do this by double-clicking on an item in the packet list or by selecting the packet in which you are interested in the packet list pane and selecting **View › Show Packet in New Window**. This allows you to easily compare two or more packets, even across multiple files.

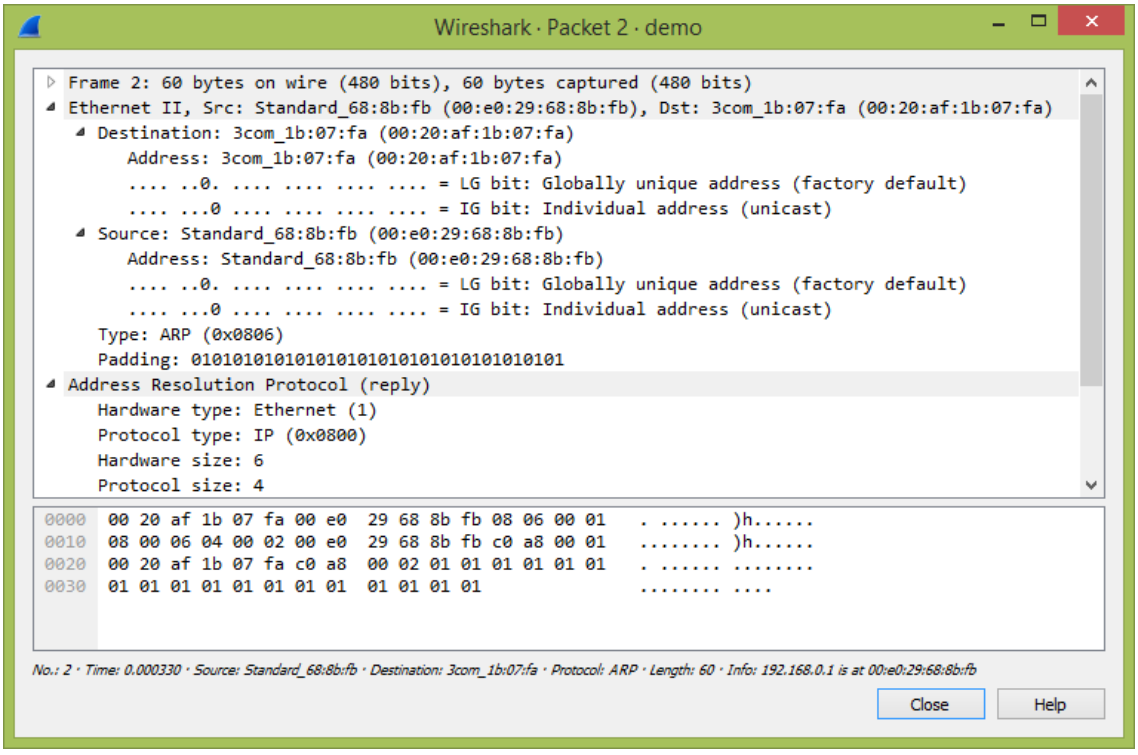


Figure 55. Viewing a packet in a separate window

Along with double-clicking the packet list and using the main menu there are a number of other ways to open a new packet window:

- Hold down the shift key and double-click on a frame link in the packet details.
- From The menu items of the “Packet List” pop-up menu.
- From The menu items of the “Packet Details” pop-up menu.

Pop-up Menus

You can open a pop-up menu over the “Packet List”, its column heading, “Packet Details”, or “Packet Bytes” by clicking your right mouse button on the corresponding item.

Pop-up Menu Of The “Packet List” Column Header

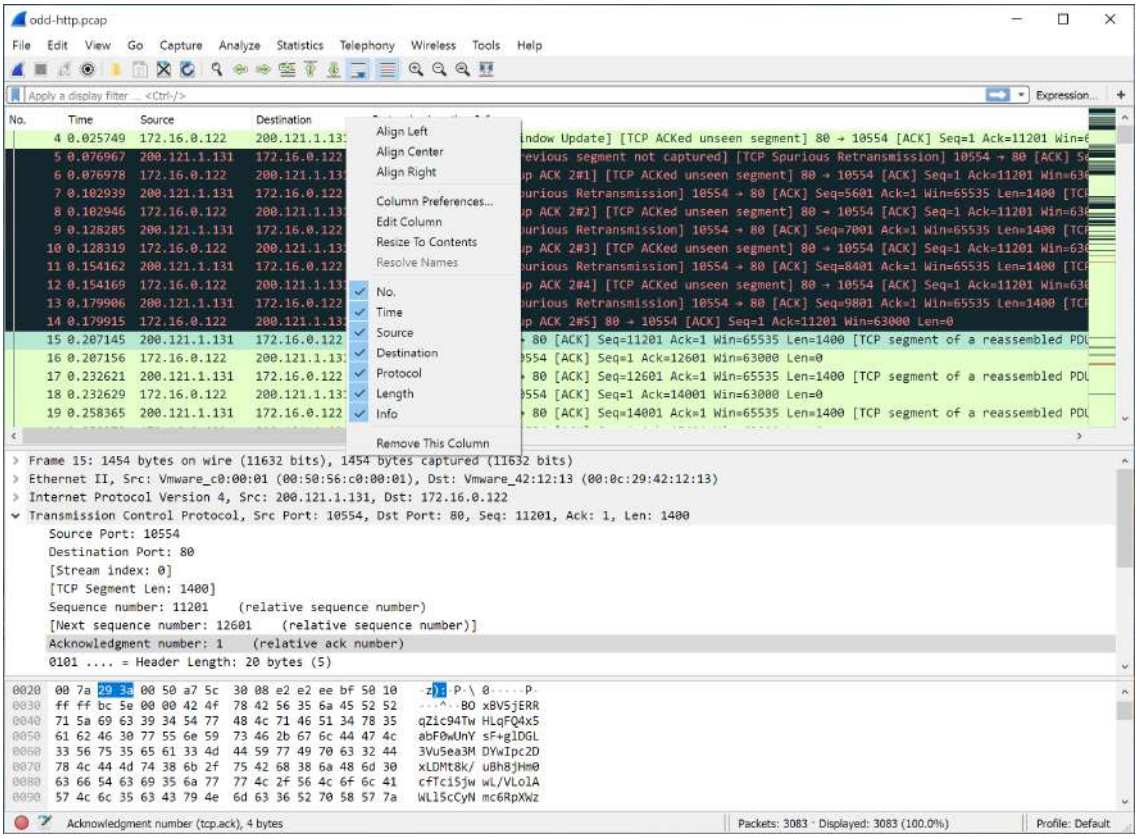


Figure 56. Pop-up menu of the “Packet List” column header

The following table gives an overview of which functions are available in this header, where to find the corresponding function in the main menu, and a description of each item.

Table 18. The menu items of the “Packet List” column header pop-up menu

Item	Description
Align Left	Left-align values in this column.
Align Center	Center-align values in this column.
Align Right	Right-align values in this column.
Column Preferences…	Open the “Preferences” dialog for this column.
Edit Column	Open the column editor toolbar for this column.
Resize To Contents	Resize the column to fit its values.
Resolve Names	If this column contains addresses, resolve them.
No., Time, Source, et al.	Show or hide a column by selecting its item.
Remove Column	Remove this column, similar to deleting it in the “Preferences” dialog.

Pop-up Menu Of The “Packet List” Pane

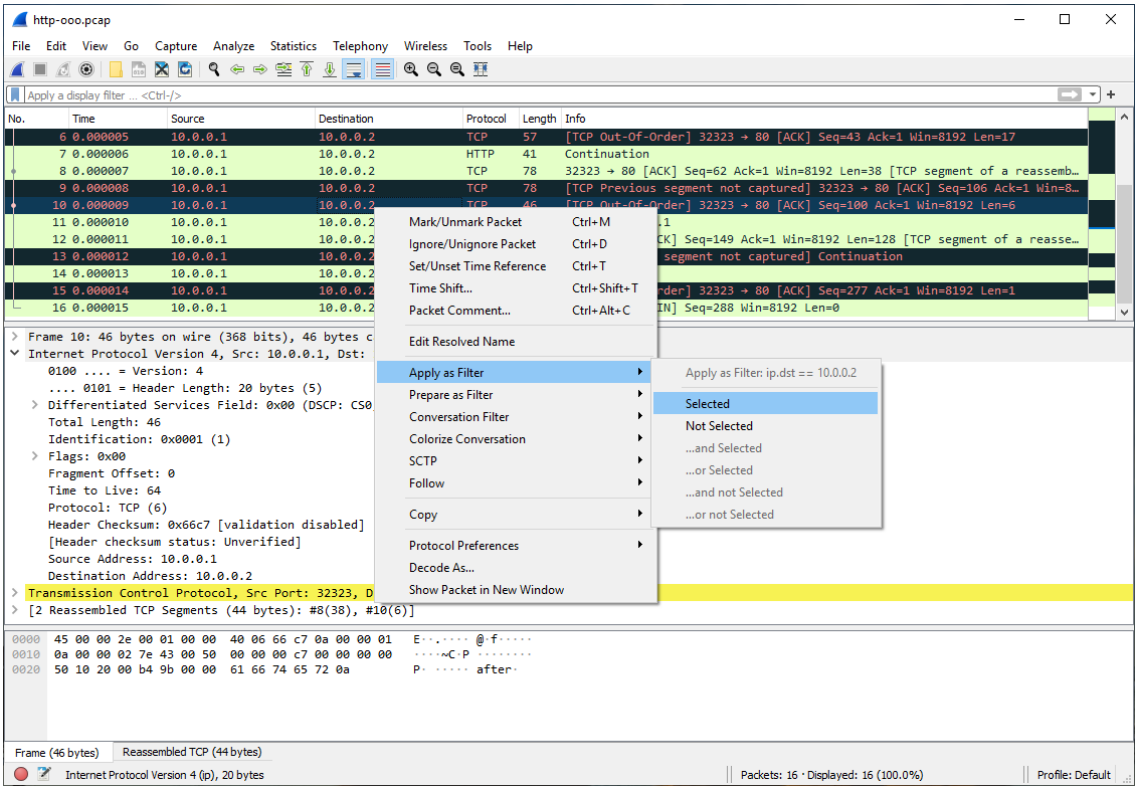


Figure 57. Pop-up menu of the “Packet List” pane

The following table gives an overview of which functions are available in this pane, where to find the corresponding function in the main menu, and a short description of each item.

Table 19. The menu items of the “Packet List” pop-up menu

Item	Corresponding main menu item	Description
Mark Packet (toggle)	**Edit**	Mark or unmark a packet.
Ignore Packet (toggle)	**Edit**	Ignore or inspect this packet while dissecting the capture file.
Set Time Reference (toggle)	**Edit**	Set or reset a time reference.
Time Shift	**Edit**	Opens the “Time Shift” dialog, which allows you to adjust the timestamps of some or all packets.
Packet Comment…	**Edit**	Opens the “Packet Comment” dialog, which lets you add a comment to a single packet. Note that the ability to save packet comments depends on your file format. E.g., pcapng supports comments, pcap does not.

Item	Corresponding main menu item	Description
Edit Resolved Name		Allows you to enter a name to resolve for the selected address.
Apply as Filter	**Analyze**	Immediately replace or append the current display filter based on the most recent packet list or packet details item selected. The first submenu item shows the filter and subsequent items show the different ways that the filter can be applied.
Prepare as Filter	**Analyze**	Change the current display filter based on the most recent packet list or packet details item selected, but don't apply it. The first submenu item shows the filter and subsequent items show the different ways that the filter can be changed.
Conversation Filter		Apply a display filter with the address information from the selected packet. For example, the IP menu entry will set a filter to show the traffic between the two IP addresses of the current packet.
Colorize Conversation		Create a new colorizing rule based on address information from the selected packet.
SCTP		Allows you to analyze and prepare a filter for this SCTP association.
Follow › TCP Stream	**Analyze**	Open a window that displays all the TCP segments captured that are on the same TCP connection as a selected packet. See Following Protocol Streams.
Follow › UDP Stream	**Analyze**	Same functionality as "Follow TCP Stream" but for UDP "streams".
Follow › DCCP Stream	**Analyze**	Same functionality as "Follow TCP Stream" but for DCCP streams.
Follow › TLS Stream	**Analyze**	Same functionality as "Follow TCP Stream" but for TLS or SSL streams. See the wiki page on SSL for instructions on providing TLS keys.
Follow › HTTP Stream	**Analyze**	Same functionality as "Follow TCP Stream" but for HTTP streams.
Copy › Summary as Text		Copy the summary fields as displayed to the clipboard as tab-separated text.

Item	Corresponding main menu item	Description
Copy › ...as CSV		Copy the summary fields as displayed to the clipboard as comma-separated text.
Copy › ...as YAML		Copy the summary fields as displayed to the clipboard as YAML data.
Copy › As Filter		Prepare a display filter based on the currently selected item and copy that filter to the clipboard.
Copy › Bytes as Hex + ASCII Dump		Copy the packet bytes to the clipboard in full “hexdump” format.
Copy › ...as Hex Dump		Copy the packet bytes to the clipboard in “hexdump” format without the ASCII portion.
Copy › ...as Printable Text		Copy the packet bytes to the clipboard as ASCII text, excluding non-printable characters.
Copy › ...as a Hex Stream		Copy the packet bytes to the clipboard as an unpunctuated list of hex digits.
Copy › ...as Raw Binary		Copy the packet bytes to the clipboard as raw binary. The data is stored in the clipboard using the MIME type “application/octet-stream”.
Protocol Preferences		Adjust the preferences for the selected protocol.
Decode As...	**Analyze**	Change or apply a new relation between two dissectors.
Show Packet in New Window	**View**	Shows the selected packet in a separate window. The separate window shows only the packet details and bytes. See Viewing a packet in a separate window for details.

Pop-up Menu Of The “Packet Details” Pane

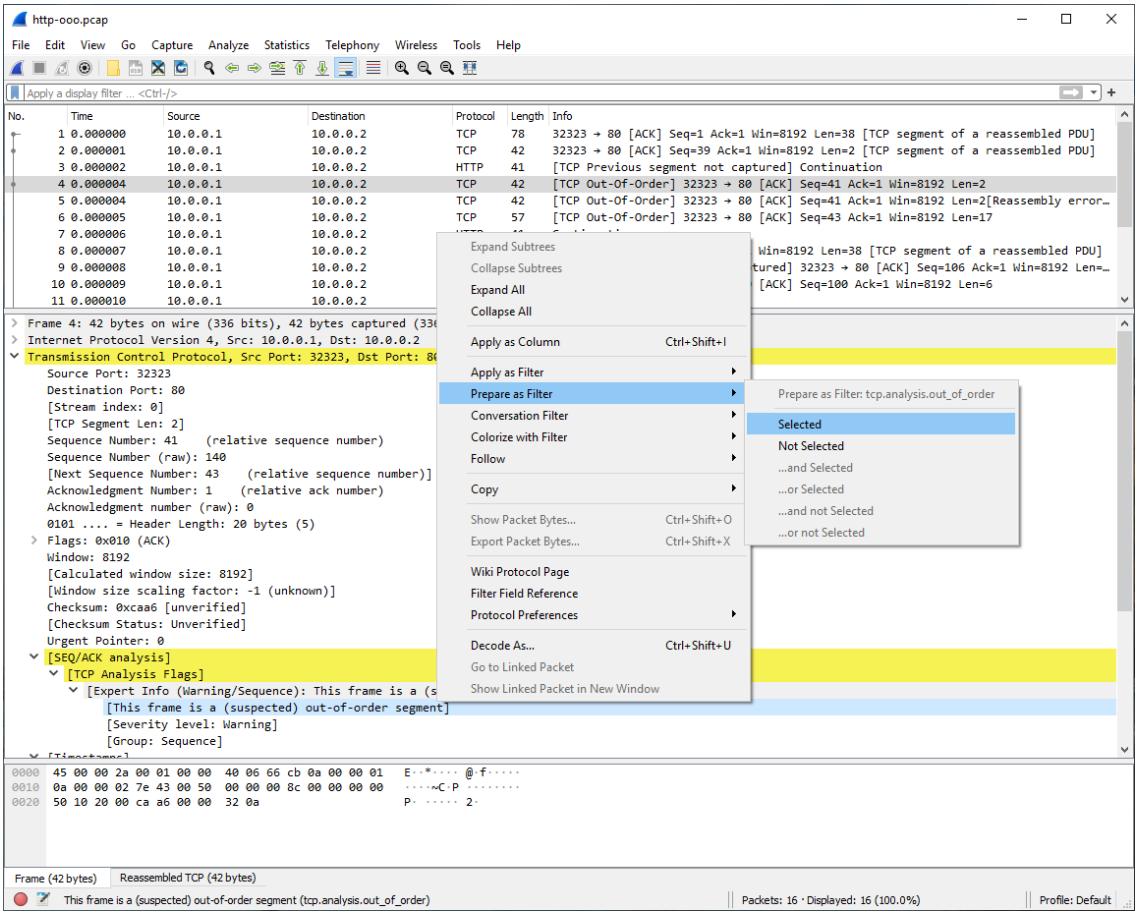


Figure 58. Pop-up menu of the “Packet Details” pane

The following table gives an overview of which functions are available in this pane, where to find the corresponding function in the main menu, and a short description of each item.

Table 20. The menu items of the “Packet Details” pop-up menu

Item	Corresponding main menu item	Description
Expand Subtrees	**View**	Expand the currently selected subtree.
Collapse Subtrees	**View**	Collapse the currently selected subtree.
Expand All	**View**	Expand all subtrees in all packets in the capture.
Collapse All	**View**	Wireshark keeps a list of all the protocol subtrees that are expanded, and uses it to ensure that the correct subtrees are expanded when you display a packet. This menu item collapses the tree view of all packets in the capture list.
Apply as Column		Use the selected protocol item to create a new column in the packet list.

Item	Corresponding main menu item	Description
Apply as Filter	**Analyze**	Immediately replace or append the current display filter based on the most recent packet list or packet details item selected. The first submenu item shows the filter and subsequent items show the different ways that the filter can be applied.
Prepare as Filter	**Analyze**	Change the current display filter based on the most recent packet list or packet details item selected, but don't apply it. The first submenu item shows the filter and subsequent items show the different ways that the filter can be changed.
Colorize with Filter		This menu item uses a display filter with the information from the selected protocol item to build a new colorizing rule.
Follow › TCP Stream	**Analyze**	Open a window that displays all the TCP segments captured that are on the same TCP connection as a selected packet. See Following Protocol Streams.
Follow › UDP Stream	**Analyze**	Same functionality as "Follow TCP Stream" but for UDP "streams".
Follow › TLS Stream	**Analyze**	Same functionality as "Follow TCP Stream" but for TLS or SSL streams. See the wiki page on SSL for instructions on providing TLS keys.
Follow › HTTP Stream	**Analyze**	Same functionality as "Follow TCP Stream" but for HTTP streams.
Copy › All Visible Items	**Edit**	Copy the packet details as displayed.
Copy › All Visible Selected Tree Items	**Edit**	Copy the selected packet detail and its children as displayed.
Copy › Description	**Edit**	Copy the displayed text of the selected field to the system clipboard.
Copy › Fieldname	**Edit**	Copy the name of the selected field to the system clipboard.
Copy › Value	**Edit**	Copy the value of the selected field to the system clipboard.
Copy › As Filter	**Edit**	Prepare a display filter based on the currently selected item and copy it to the clipboard.

Item	Corresponding main menu item	Description
Copy › Bytes as Hex + ASCII Dump		Copy the packet bytes to the clipboard in full “hexdump” format.
Copy › ...as Hex Dump		Copy the packet bytes to the clipboard in “hexdump” format without the ASCII portion.
Copy › ...as Printable Text		Copy the packet bytes to the clipboard as ASCII text, excluding non-printable characters.
Copy › ...as a Hex Stream		Copy the packet bytes to the clipboard as an unpunctuated list of hex digits.
Copy › ...as Raw Binary		Copy the packet bytes to the clipboard as raw binary. The data is stored in the clipboard using the MIME type “application/octet-stream”.
Copy › ...as Escaped String		Copy the packet bytes to the clipboard as C-style escape sequences.
Export Packet Bytes...	**File**	This menu item is the same as the File menu item of the same name. It allows you to export raw packet bytes to a binary file.
Wiki Protocol Page		Open the wiki page for the selected protocol in your web browser.
Filter Field Reference		Open the filter field reference web page for the selected protocol in your web browser.
Protocol Preferences		Adjust the preferences for the selected protocol.
Decode As...	**Analyze**	Change or apply a new relation between two dissectors.
Go to Linked Packet	**Go**	If the selected field has a corresponding packet such as the matching request for a DNS response, go to it.
Show Linked Packet in New Window	**Go**	If the selected field has a corresponding packet such as the matching request for a DNS response, show the selected packet in a separate window. See Viewing a packet in a separate window for details.

Pop-up Menu Of The “Packet Bytes” Pane

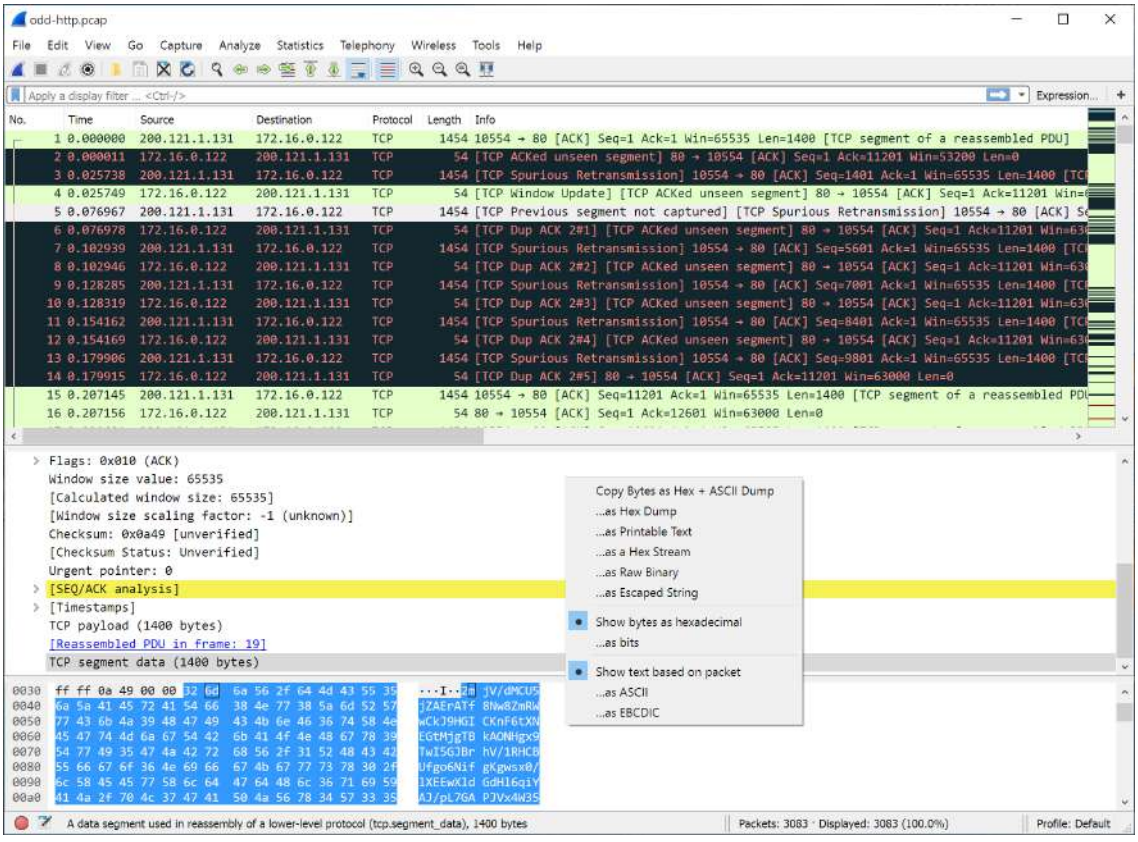


Figure 59. Pop-up menu of the “Packet Bytes” pane

The following table gives an overview of which functions are available in this pane along with a short description of each item.

Table 21. The menu items of the “Packet Bytes” pop-up menu

Item	Description
Copy Bytes as Hex + ASCII Dump	Copy the packet bytes to the clipboard in full “hexdump” format.
…as Hex Dump	Copy the packet bytes to the clipboard in “hexdump” format without the ASCII portion.
…as Printable Text	Copy the packet bytes to the clipboard as ASCII text, excluding non-printable characters.
…as a Hex Stream	Copy the packet bytes to the clipboard as an unpunctuated list of hex digits.
…as Raw Binary	Copy the packet bytes to the clipboard as raw binary. The data is stored in the clipboard using the MIME type “application/octet-stream”.
…as Escaped String	Copy the packet bytes to the clipboard as C-style escape sequences.
Show bytes as hexadecimal	Display the byte data as hexadecimal digits.
Show bytes as bits	Display the byte data as binary digits.

Item	Description
Show text based on packet	Show the “hexdump” data with text.
…as ASCII	Use ASCII encoding when displaying “hexdump” text.
…as EBCDIC	Use EBCDIC encoding when displaying “hexdump” text.

Pop-up Menu Of The “Packet Diagram” Pane

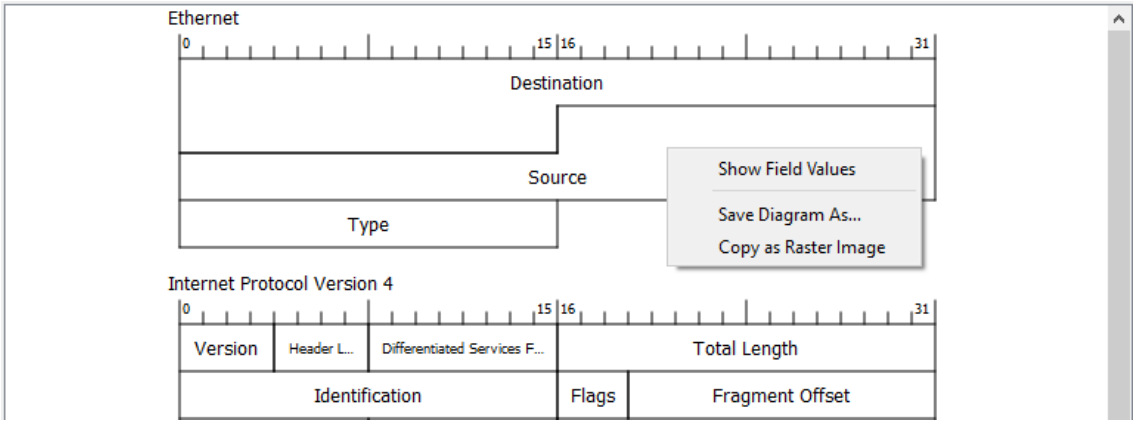


Figure 60. Pop-up menu of the “Packet Diagram” pane

The following table gives an overview of which functions are available in this pane along with a short description of each item.

Table 22. The menu items of the “Packet Diagram” pop-up menu

Item	Description
Show Field Values	Display current value for each field on the packet diagram.
Save Diagram As…	Save the packet diagram to an image file (PNG, BMP, JPEG).
Copy as Raster Image	Copy the packet diagram to the clipboard in raster (ARGB32) format.

Filtering Packets While Viewing

Wireshark has two filtering languages: *capture filters* and *display filters*. *Capture filters* are used for filtering when capturing packets and are discussed in Filtering while capturing. *Display filters* are used for filtering which packets are displayed and are discussed below. For more information about *display filter* syntax, see the wireshark-filter(4) man page.

Display filters allow you to concentrate on the packets you are interested in while hiding the currently uninteresting ones. They allow you to only display packets based on:

- Protocol
- The presence of a field
- The values of fields

- A comparison between fields
- ... and a lot more!

To only display packets containing a particular protocol, type the protocol name in the display filter toolbar of the Wireshark window and press enter to apply the filter. Filtering on the TCP protocol shows an example of what happens when you type *tcp* in the display filter toolbar.

NOTE	Protocol and field names are usually in lowercase.

NOTE	Don't forget to press enter or click on the apply display filter button after entering the filter expression.

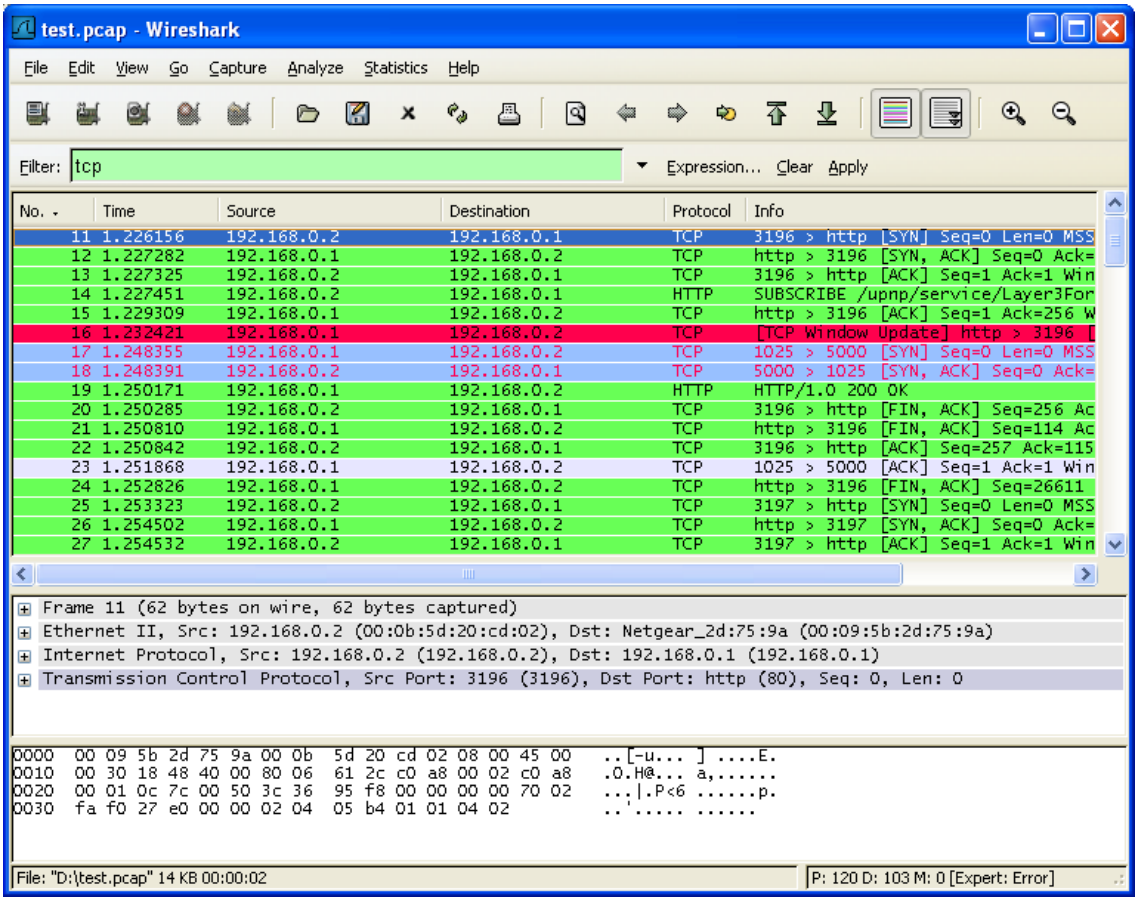


Figure 61. Filtering on the TCP protocol

As you may have noticed, only packets containing the TCP protocol are now displayed, so packets 1-10 are hidden and packet number 11 is the first packet displayed.

NOTE	When using a display filter, all packets remain in the capture file. The display filter only changes the display of the capture file but not its content!

To remove the filter, click on the **[Clear]** button to the right of the display filter field. All packets will become visible again.

Display filters can be very powerful and are discussed in further detail in Building Display Filter Expressions

It's also possible to create display filters with the *Display Filter Expression* dialog box. More information about the *Display Filter Expression* dialog box is available in The "Display Filter Expression" Dialog Box.

Building Display Filter Expressions

Wireshark provides a display filter language that enables you to precisely control which packets are displayed. They can be used to check for the presence of a protocol or field, the value of a field, or even compare two fields to each other. These comparisons can be combined with logical operators, like "and" and "or", and parentheses into complex expressions.

The following sections will go into the display filter functionality in more detail.

TIP There are many display filter examples on the *Wireshark Wiki Display Filter page* at: https://gitlab.com/wireshark/wireshark/wikis/DisplayFilters.

Display Filter Fields

The simplest display filter is one that displays a single protocol. To only display packets containing a particular protocol, type the protocol into Wireshark's display filter toolbar. For example, to only display TCP packets, type *tcp* into Wireshark's display filter toolbar. Similarly, to only display packets containing a particular field, type the field into Wireshark's display filter toolbar. For example, to only display HTTP requests, type *http.request* into Wireshark's display filter toolbar.

You can filter on any protocol that Wireshark supports. You can also filter on any field that a dissector adds to the tree view, if the dissector has added an abbreviation for that field. A full list of the available protocols and fields is available through the menu item **View › Internals › Supported Protocols**.

Comparing Values

You can build display filters that compare values using a number of different comparison operators. For example, to only display packets to or from the IP address 192.168.0.1, use `ip.addr==192.168.0.1`.

A complete list of available comparison operators is shown in Display Filter comparison operators.

TIP English and C-like operators are interchangeable and can be mixed within a filter string.

Table 23. Display Filter comparison operators

English	Alias	C-like	Description	Example
eq	any_eq	==	Equal (any if more than one)	`ip.src == 10.0.0.5`
ne	all_ne	!=	Not equal (all if more than one)	`ip.src != 10.0.0.5`
	all_eq	===	Equal (all if more than one)	`ip.src === 10.0.0.5`
	any_ne	!==	Not equal (any if more than one)	`ip.src !== 10.0.0.5`
gt		>	Greater than	`frame.len > 10`
lt		<	Less than	`frame.len < 128`
ge		>=	Greater than or equal to	`frame.len ge 0x100`
le		<=	Less than or equal to	`frame.len <= 0x20`
contains			Protocol, field or slice contains a value	`sip.To contains "a1762"`
matches		~	Protocol or text field matches a Perl-compatible regular expression	`http.host matches "acme\\.(org|com|net)"`

NOTE The meaning of != (all not equal) was changed in Wireshark 3.6. Before it used to mean "any not equal".

All protocol fields have a type. Display Filter Field Types provides a list of the types with examples of how to use them in display filters.

Display Filter Field Types

Unsigned integer

Can be 8, 16, 24, 32, or 64 bits. You can express integers in decimal, octal, hexadecimal or binary. The following display filters are equivalent:

```
ip.len le 1500
```

```
ip.len le 02734
```

```
ip.len le 0x5dc
```

```
ip.len le 0b10111011100
```

Signed integer

Can be 8, 16, 24, 32, or 64 bits. As with unsigned integers you can use decimal, octal, hexadecimal or binary.

Boolean

Can be 1 or "True" or "TRUE", 0 or "False" or "FALSE" (without quotes).

A Boolean field is present regardless if its value is true or false. For example, `tcp.flags.syn` is present in all TCP packets containing the flag, whether the SYN flag is 0 or 1. To only match TCP packets with the SYN flag set, you need to use `tcp.flags.syn == 1` or `tcp.flags.syn == True`.

Ethernet address

6 bytes separated by a colon (:), dot (.), or dash (-) with one or two bytes between separators:

```
eth.dst == ff:ff:ff:ff:ff:ff
```

```
eth.dst == ff-ff-ff-ff-ff-ff
```

```
eth.dst == ffff.ffff.ffff
```

IPv4 address

```
ip.addr == 192.168.0.1
```

Classless InterDomain Routing (CIDR) notation can be used to test if an IPv4 address is in a certain subnet. For example, this display filter will find all packets in the 129.111 Class-B network:

```
ip.addr == 129.111.0.0/16
```

IPv6 address

```
ipv6.addr == ::1
```

As with IPv4 addresses, IPv6 addresses can match a subnet.

Text string

```
http.request.uri == "https://www.wireshark.org/"
```

Strings are a sequence of bytes. Functions like `lower()` use ASCII, otherwise no particular encoding is assumed. String literals are specified with double quotes. Characters can also be specified using a byte escape sequence using hex \x*hh* or octal *ddd*, where *h* and *d* are hex and octal numerical digits respectively:

```
dns.qry.name contains "www.\x77\x69\x72\x65\x73\x68\x61\x72\x6b.org"
```

Alternatively, a raw string syntax can be used. Such strings are prefixed with `r` or `R` and treat backslash as a literal character.

```
http.user_agent matches r"\(X11;"
```

Date and time

```
frame.time == "Sep 26, 2004 23:18:04.954975"
```

```
ntp.xmt ge "2020-07-04 12:34:56"
```

The value of an absolute time field is expressed as a string, using one of the two formats above. Fractional seconds can be omitted or specified up to nanosecond precision; extra trailing zeros

are allowed but not other digits. The string cannot take a time zone suffix, and is always parsed as in the local time zone, even for fields that are displayed in UTC.

In the first format, the abbreviated month names must be in English regardless of locale. In the second format, any number of time fields may be omitted, in the order from least significant (seconds) to most, but at least the entire date must be specified:

`frame.time < "2022-01-01"`

In the second format, a `T` may appear between the date and time as in ISO 8601, but not when less significant times are dropped.

Some Examples

```
udp contains 81:60:03
```

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

```
sip.To contains "a1762"
```

The display filter above matches packets where the SIP To-header contains the string "a1762" anywhere in the header.

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

The display filter above matches HTTP packets where the HOST header contains acme.org, acme.com, or acme.net. Comparisons are case-insensitive.

```
tcp.flags & 0x02
```

That display filter will match all packets that contain the “tcp.flags” field with the 0x02 bit, i.e., the SYN bit, set.

Possible Pitfalls Using Regular Expressions

String literals containing regular expressions are parsed twice. Once by Wireshark’s display filter engine and again by the PCRE2 library. It’s important to keep this in mind when using the "matches" operator with regex escape sequences and special characters.

For example, the filter expression `frame matches "AB\x43"` uses the string `"ABC"` as input pattern to PCRE. However, the expression `frame matches "AB\\x43"` uses the string `"AB\x43"` as the pattern. In this case both expressions give the same result because Wireshark and PCRE both support the same

byte escape sequence (0x43 is the ASCII hex code for `C`).

An example where this fails badly is `foo matches "bar\x28"`. Because 0x28 is the ASCII code for `(` the pattern input to PCRE is `"bar("`. This regular expression is syntactically invalid (missing closing parenthesis). To match a literal parenthesis in a display filter regular expression it must be escaped (twice) with backslashes.

TIP	Using raw strings avoids most problem with the "matches" operator and double escape requirements.

Combining Expressions

You can combine filter expressions in Wireshark using the logical operators shown in Display Filter Logical Operations

Table 24. Display Filter Logical Operations

English	C-like	Description	Example
and	&&	Logical AND	`ip.src==10.0.0.5 and tcp.flags.fin`
or	||	Logical OR	`ip.src==10.0.0.5 or ip.src==192.1.1.1`
xor	^^	Logical XOR	`tr.dst[0:3] == 0.6.29 xor tr.src[0:3] == 0.6.29`
not	!	Logical NOT	`not llc`
[…]		Subsequence	See “Slice Operator” below.
in		Set Membership	http.request.method in {"HEAD", "GET"}. See “Membership Operator” below.

Slice Operator

Wireshark allows you to select a subsequence of a sequence in rather elaborate ways. After a label you can place a pair of brackets [] containing a comma separated list of range specifiers.

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

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

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

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

```
eth.src[:4] == 00:00:83:00
```

The example above uses the :m format, which takes everything from the beginning of a sequence to offset m. It is equivalent to 0:m

```
eth.src[4:] == 20:20
```

The example above uses the n: format, which takes everything from offset n to the end of the sequence.

```
eth.src[2] == 83
```

The example above uses the n format to specify a single range. In this case the element in the sequence at offset n is selected. This is equivalent to n:1.

```
eth.src[0:3,1-2,:4,4:,2] ==
00:00:83:00:83:00:00:83:00:20:20:83
```

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

Membership Operator

Wireshark allows you to test a field for membership in a set of values or fields. After the field name, use the `in` operator followed by the set items surrounded by braces {}. For example, to display packets with a TCP source or destination port of 80, 443, or 8080, you can use `tcp.port in {80, 443, 8080}`. Set elements must be separated by commas. The set of values can also contain ranges: `tcp.port in {443,4430..4434}`.

NOTE

The display filter

```
tcp.port in {80, 443, 8080}
```

is equivalent to

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

However, the display filter

```
tcp.port in {443, 4430..4434}
```

is not equivalent to

```
tcp.port == 443 || (tcp.port >= 4430 && tcp.port <= 4434)
```

This is because comparison operators are satisfied when *any* field matches the filter, so a packet with a source port of 56789 and destination port of port 80 would also match the second filter since `56789 >= 4430 && 80 <= 4434` is true. In contrast, the membership operator tests a single field against the range condition.

Sets are not just limited to numbers, other types can be used as well:

```
http.request.method in {"HEAD", "GET"}
ip.addr in {10.0.0.5 .. 10.0.0.9, 192.168.1.1..192.168.1.9}
frame.time_delta in {10 .. 10.5}
```

Arithmetic operators

You can perform the arithmetic operations on numeric fields shown in Display Filter Arithmetic Operations

Table 25. Display Filter Arithmetic Operations

Name	Syntax	Description
Unary minus	-A	Negation of A
Addition	A + B	Add B to A
Subtraction	A - B	Subtract B from A
Multiplication	A * B	Multiply A times B

Name	Syntax	Description
Division	A / B	Divide A by B
Modulo	A % B	Remainder of A divided by B
Bitwise AND	A & B	Bitwise AND of A and B

Arithmetic expressions can be grouped using curly braces.

Functions

The display filter language has a number of functions to convert fields, see Display Filter Functions.

Table 26. Display Filter Functions

Function	Description
upper	Converts a string field to uppercase.
lower	Converts a string field to lowercase.
len	Returns the byte length of a string or bytes field.
count	Returns the number of field occurrences in a frame.
string	Converts a non-string field to a string.
max	Return the maximum value for the arguments.
min	Return the minimum value for the arguments.
abs	Return the absolute value for the argument.

The `upper` and `lower` functions can used to force case-insensitive matches: `lower(http.server) contains "apache"`.

To find HTTP requests with long request URIs: `len(http.request.uri) > 100`. Note that the `len` function yields the string length in bytes rather than (multi-byte) characters.

Usually an IP frame has only two addresses (source and destination), but in case of ICMP errors or tunneling, a single packet might contain even more addresses. These packets can be found with `count(ip.addr) > 2`.

The `string` function converts a field value to a string, suitable for use with operators like "matches" or "contains". Integer fields are converted to their decimal representation. It can be used with IP/Ethernet addresses (as well as others), but not with string or byte fields.

For example, to match odd frame numbers:

```
string(frame.number) matches "[13579]$"
```

To match IP addresses ending in 255 in a block of subnets (172.16 to 172.31):

```
string(ip.dst) matches r"^172\.(1[6-9]|2[0-9]|3[0-1])\.[0-9]{1,3}\.255"
```

The functions max() and min() take any number of arguments of the same type and returns the largest/smallest respectively of the set.

```
max(tcp.srcport, tcp.dstport) <= 1024
```

Sometimes Fields Change Names

As protocols evolve they sometimes change names or are superseded by newer standards. For example, DHCP extends and has largely replaced BOOTP and TLS has replaced SSL. If a protocol dissector originally used the older names and fields for a protocol the Wireshark development team might update it to use the newer names and fields. In such cases they will add an alias from the old protocol name to the new one in order to make the transition easier.

For example, the DHCP dissector was originally developed for the BOOTP protocol but as of Wireshark 3.0 all of the “bootp” display filter fields have been renamed to their “dhcp” equivalents. You can still use the old filter names for the time being, e.g., “bootp.type” is equivalent to “dhcp.type” but Wireshark will show the warning “"bootp" is deprecated” when you use it. Support for the deprecated fields may be removed in the future.

Some protocol names can be ambiguous

In some particular cases relational expressions (equal, less than, etc.) can be ambiguous. The filter name of a protocol or protocol field can contain any letter and digit in any order, possibly separated by dots. That can be indistinguishable from a literal value (usually numerical values in hexadecimal). For example the semantic value of `fc` can be the protocol Fibre Channel or the number 0xFC in hexadecimal because the 0x prefix is optional for hexadecimal numbers.

Any value that matches a registered protocol or protocol field filter name is interpreted semantically as such. If it doesn’t match a protocol name the normal rules for parsing literal values apply.

So in the case of 'fc' the lexical token is interpreted as "Fibre Channel" and not 0xFC. In the case of 'fd' it would be interpreted as 0xFD because it is a well-formed hexadecimal literal value (according to the rules of display filter language syntax) and there is no protocol registered with the filter name 'fd'.

How ambiguous values are interpreted may change in the future. To avoid this problem and resolve the ambiguity there is additional syntax available. Values in-between angle brackets are always and only treated as literal values. Bytes arrays and numeric values can also be prefixed with a colon to force interpretation as a literal value. Values prefixed with a dot are always treated as a

protocol name. The dot stands for the root of the protocol namespace and is optional)

```
frame[10:] contains .fc or frame[10] == :fc and not frame contains <cafe.face>
```

If you are writing a script, or you think your expression may not be giving the expected results because of the syntactical ambiguity of some filter expression it is advisable to use the explicit syntax to indicate the correct meaning for that expression.

The “Display Filter Expression” Dialog Box

When you are accustomed to Wireshark’s filtering system and know what labels you wish to use in your filters it can be very quick to simply type a filter string. However, if you are new to Wireshark or are working with a slightly unfamiliar protocol it can be very confusing to try to figure out what to type. The “Display Filter Expression” dialog box helps with this.

TIP The “Display Filter Expression” dialog box is an excellent way to learn how to write Wireshark display filter strings.

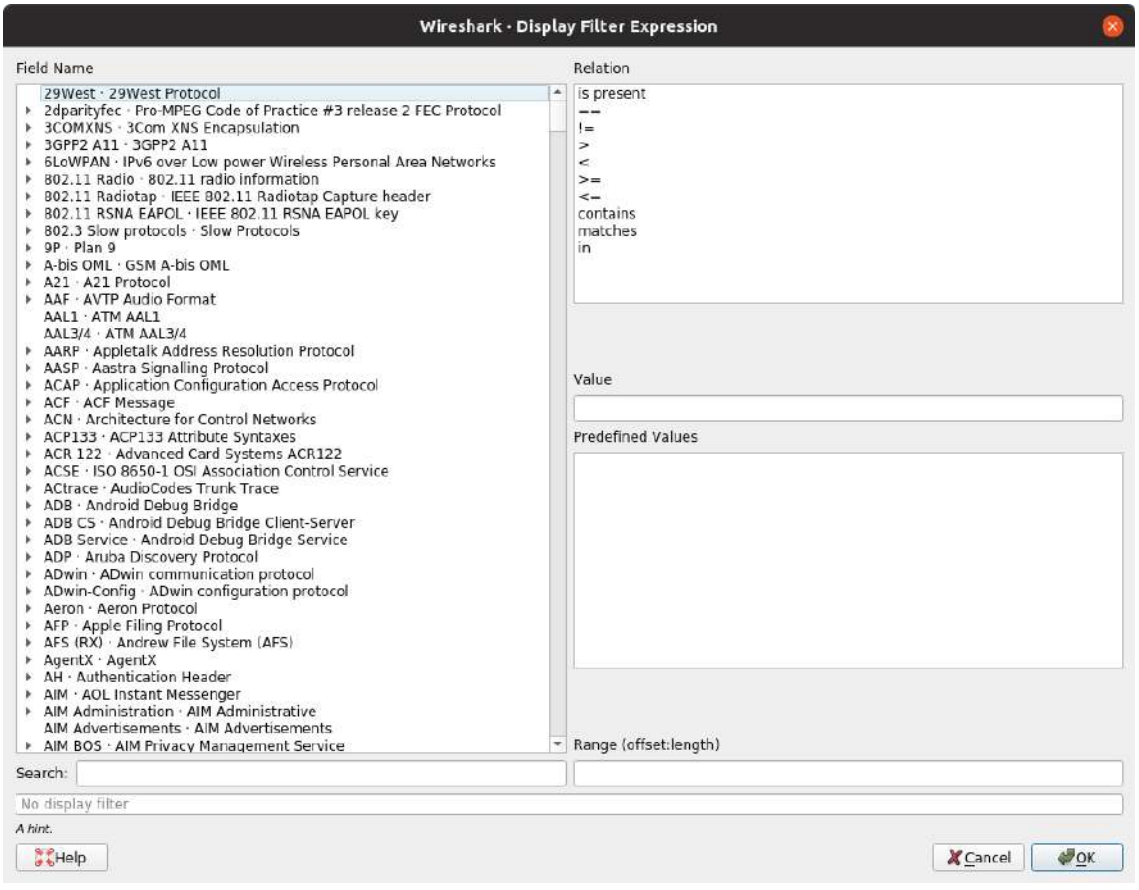


Figure 62. The “Display Filter Expression” dialog box

When you first bring up the Display Filter Expression dialog box you are shown a tree of field names, organized by protocol, and a box for selecting a relation.

Field Name

Select a protocol field from the protocol field tree. Every protocol with filterable fields is listed at the top level. You can search for a particular protocol entry by entering the first few letters of the protocol name. By expanding a protocol name you can get a list of the field names available for filtering for that protocol.

Relation

Select a relation from the list of available relation. The *is present* is a unary relation which is true if the selected field is present in a packet. All other listed relations are binary relations which require additional data (e.g. a *Value* to match) to complete.

When you select a field from the field name list and select a binary relation (such as the equality relation ==) you will be given the opportunity to enter a value, and possibly some range information.

Value

You may enter an appropriate value in the *Value* text box. The *Value* will also indicate the type of value for the *Field Name* you have selected (like character string).

Predefined Values

Some of the protocol fields have predefined values available, much like enumerations in C. If the selected protocol field has such values defined, you can choose one of them here.

Search

Lets you search for a full or partial field name or description. Regular expressions are supported. For example, searching for "tcp.*flag" shows the TCP flags fields supported by a wide variety of dissectors, while "^tcp.flag" shows only the TCP flags fields supported by the TCP dissector.

Range

A range of integers or a group of ranges, such as `1-12` or `39-42,98-2000`.

[Help]

Opens this section of the User's Guide.

[OK]

When you have built a satisfactory expression click **[OK]** and a filter string will be built for you.

[Cancel]

You can leave the "Add Expression..." dialog box without any effect by clicking the **[Cancel]** button.

Defining And Saving Filters

You create pre-defined filters that appear in the capture and display filter bookmark menus (). This can save time in remembering and retyping some of the more complex filters you use.

To create or edit capture filters, select **Manage Capture Filters** from the capture filter bookmark menu or **Capture › Capture Filters…** from the main menu. Display filters can be created or edited by selecting **Manage Display Filters** from the display filter bookmark menu or **Analyze › Display Filters…** from the main menu. Wireshark will open the corresponding dialog as shown in The "Capture Filters" and "Display Filters" dialog boxes. The two dialogs look and work similar to one another. Both are described here, and the differences are noted as needed.

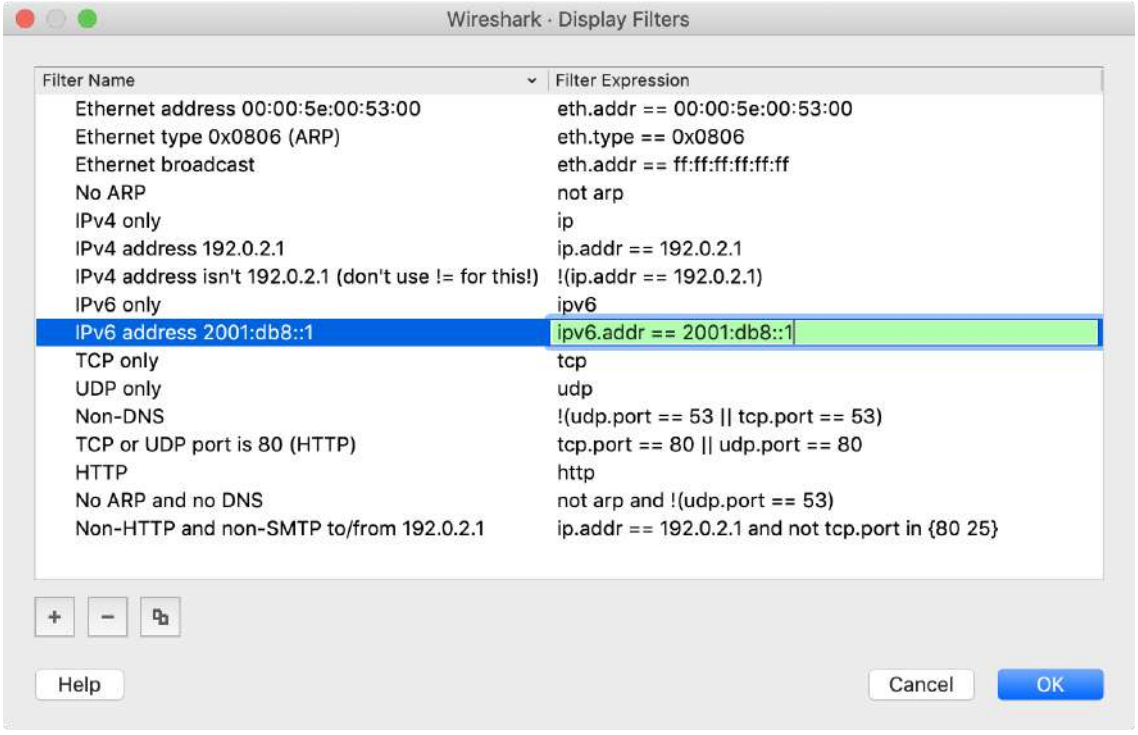


Figure 63. The "Capture Filters" and "Display Filters" dialog boxes

[+]

Adds a new filter to the list. You can edit the filter name or expression by double-clicking on it.

The filter name is used in this dialog to identify the filter for your convenience and is not used elsewhere. You can create multiple filters with the same name, but this is not very useful.

When typing in a filter string, the background color will change depending on the validity of the filter similar to the main capture and display filter toolbars.

[-]

Delete the selected filter. This will be greyed out if no filter is selected.

[Copy]

Copy the selected filter. This will be greyed out if no filter is selected.

[OK]

Saves the filter settings and closes the dialog.

[Cancel]

Closes the dialog without saving any changes.

Defining And Saving Filter Macros

You can define a filter macro with Wireshark and label it for later use. This can save time in remembering and retyping some of the more complex filters you use.

To define and save your own filter macros, follow the steps below:

1. In the main menu select **Analyze › Display Filter Macros...**. Wireshark will open a corresponding dialog Display Filter Macros window.

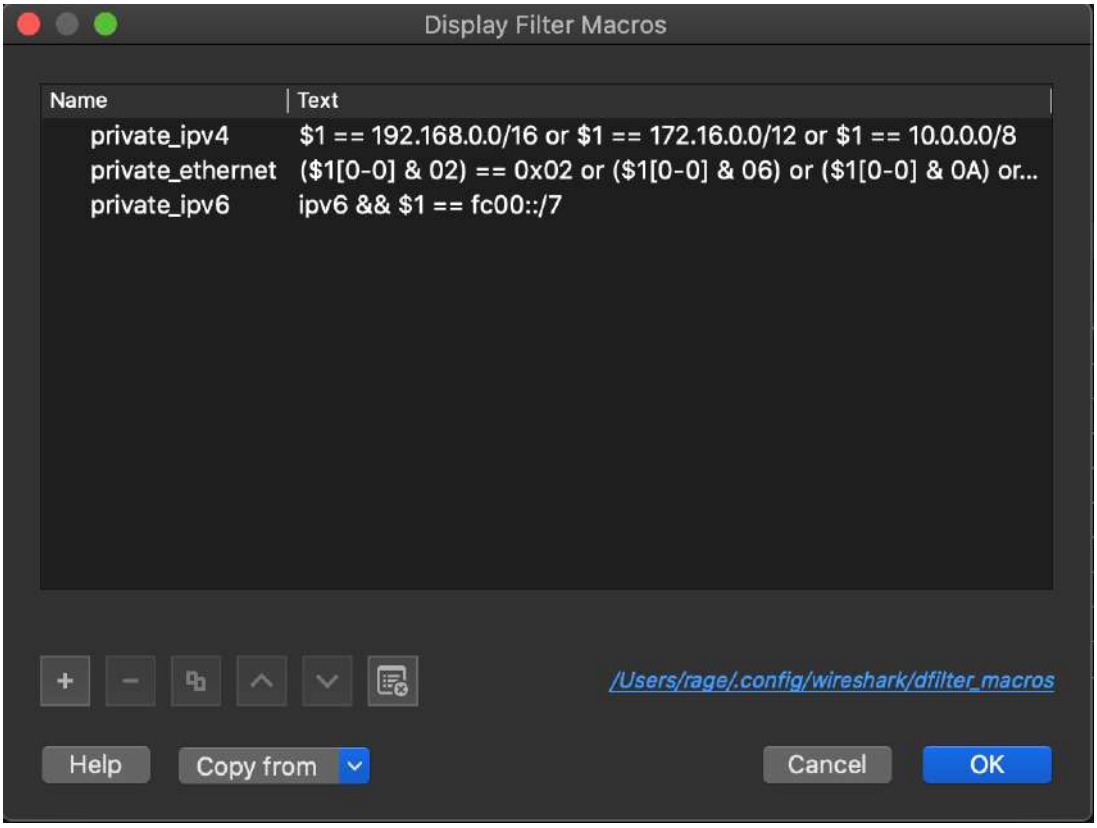


Figure 64. Display Filter Macros window

2. To add a new filter macro, click the **[+]** button in the bottom-left corner. A new row will appear in the Display Filter Macros table above.
3. Enter the name of your macro in the `Name` column. Enter your filter macro in the `Text` column.
4. To save your modifications, click the **[OK]** button in the bottom-right corner of the Display Filter Macros window.

To learn more about display filter macro syntax, see Display Filter Macros.

Finding Packets

You can easily find packets once you have captured some packets or have read in a previously saved capture file. Simply select **Edit › Find Packet...** in the main menu. Wireshark will open a toolbar between the main toolbar and the packet list shown in The “Find Packet” toolbar.

The “Find Packet” Toolbar

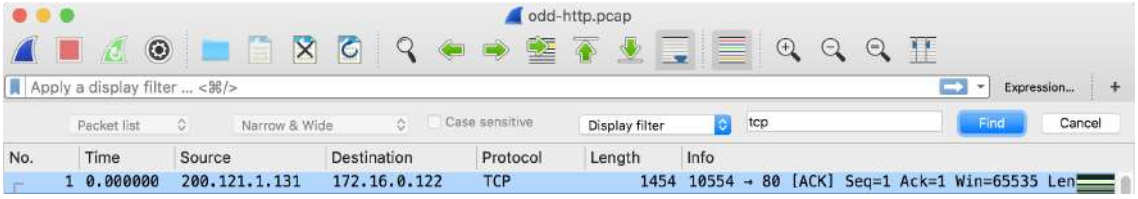


Figure 65. The “Find Packet” toolbar

You can search using the following criteria:

Display filter

Enter a display filter string into the text entry field and click the **[Find]** button. + For example, to find the three-way handshake for a connection from host 192.168.0.1, use the following filter string:

```
ip.src==192.168.0.1 and tcp.flags.syn==1
```

The value to be found will be syntax checked while you type it in. If the syntax check of your value succeeds, the background of the entry field will turn green, if it fails, it will turn red. For more details see Filtering Packets While Viewing

Hexadecimal Value

Search for a specific byte sequence in the packet data.

For example, use “ef:bb:bf” to find the next packet that contains the UTF-8 byte order mark.

String

Find a string in the packet data, with various options.

Regular Expression

Search the packet data using Perl-compatible regular expressions. PCRE patterns are beyond the scope of this document, but typing “pcre test” into your favorite search engine should return a number of sites that will help you test and explore your expressions.

Go To A Specific Packet

You can easily jump to specific packets with one of the menu items in the **Go** menu.

The “Go Back” Command

Go back in the packet history, works much like the page history in most web browsers.

The “Go Forward” Command

Go forward in the packet history, works much like the page history in most web browsers.

The “Go to Packet” Toolbar

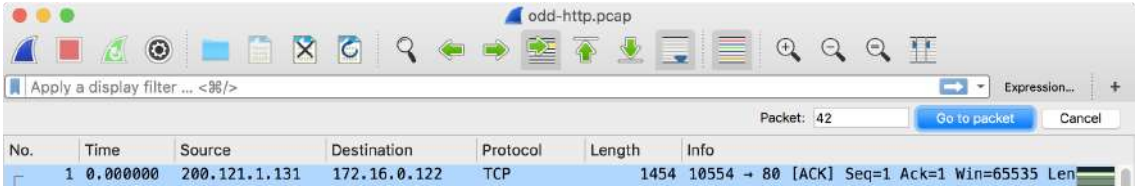


Figure 66. The “Go To Packet” toolbar

This toolbar can be opened by selecting **Go › Go to packet...** from the main menu. It appears between the main toolbar and the packet list, similar to the ”Find Packet” toolbar.

When you enter a packet number and press **[Go to packet]** Wireshark will jump to that packet.

The “Go to Corresponding Packet” Command

If a protocol field is selected which points to another packet in the capture file, this command will jump to that packet.

As these protocol fields now work like links (just as in your Web browser), it’s easier to simply double-click on the field to jump to the corresponding field.

The “Go to First Packet” Command

This command will jump to the first packet displayed.

The “Go to Last Packet” Command

This command will jump to the last packet displayed.

Marking Packets

You can mark packets in the “Packet List” pane. A marked packet will be shown with black background, regardless of the coloring rules set. Marking a packet can be useful to find it later while analyzing in a large capture file.

Marked packet information is not stored in the capture file or anywhere else. It will be lost when the capture file is closed.

You can use packet marking to control the output of packets when saving, exporting, or printing. To do so, an option in the packet range is available, see The “Packet Range” Frame.

There are several ways to mark and unmark packets. From the **Edit** menu you can select from the following:

- **Mark/Unmark Packet** toggles the marked state of a single packet. This option is also available in the packet list context menu.
- **Mark All Displayed** set the mark state of all displayed packets.
- **Unmark All Displayed** reset the mark state of all packets.

You can also mark and unmark a packet by clicking on it in the packet list with the middle mouse button.

Ignoring Packets

You can ignore packets in the “Packet List” pane. Wireshark will then pretend that they not exist in the capture file. An ignored packet will be shown with white background and grey foreground, regardless of the coloring rules set.

Ignored packet information is not stored in the capture file or anywhere else. It will be lost when the capture file is closed.

There are several ways to ignore and unignore packets. From the **Edit** menu you can select from the following:

- **Ignore/Unignore Packet** toggles the ignored state of a single packet. This option is also available in the packet list context menu.
- **Ignore All Displayed** set the ignored state of all displayed packets.
- **Unignore All Displayed** reset the ignored state of all packets.

Time Display Formats And Time References

While packets are captured, each packet is timestamped. These timestamps will be saved to the capture file, so they will be available for later analysis.

A detailed description of timestamps, timezones and alike can be found at: Time Stamps.

The timestamp presentation format and the precision in the packet list can be chosen using the View menu, see The “View” Menu.

The available presentation formats are:

- **Date and Time of Day: 1970-01-01 01:02:03.123456** The absolute date and time of the day when the packet was captured.

- **Time of Day: 01:02:03.123456** The absolute time of the day when the packet was captured.
- **Seconds Since First Captured Packet: 123.123456** The time relative to the start of the capture file or the first "Time Reference" before this packet (see Packet Time Referencing).
- **Seconds Since Previous Captured Packet: 1.123456** The time relative to the previous captured packet.
- **Seconds Since Previous Displayed Packet: 1.123456** The time relative to the previous displayed packet.
- **Seconds Since Epoch (1970-01-01): 1234567890.123456** The time relative to epoch (midnight UTC of January 1, 1970).

The available precisions (aka. the number of displayed decimal places) are:

- **Automatic (from capture file)** The timestamp precision of the loaded capture file format will be used (the default).
- **Seconds**, **Tenths of a second**, **Hundredths of a second**, **Milliseconds**, **Microseconds** or **Nanoseconds** The timestamp precision will be forced to the given setting. If the actually available precision is smaller, zeros will be appended. If the precision is larger, the remaining decimal places will be cut off.

Precision example: If you have a timestamp and it's displayed using, "Seconds Since Previous Packet" the value might be 1.123456. This will be displayed using the "Automatic" setting for libpcap files (which is microseconds). If you use Seconds it would show simply 1 and if you use Nanoseconds it shows 1.123456000.

Packet Time Referencing

The user can set time references to packets. A time reference is the starting point for all subsequent packet time calculations. It will be useful, if you want to see the time values relative to a special packet, e.g., the start of a new request. It's possible to set multiple time references in the capture file.

The time references will not be saved permanently and will be lost when you close the capture file.

Time referencing will only be useful if the time display format is set to "Seconds Since First Captured Packet". If one of the other time display formats are used, time referencing will have no effect (and will make no sense either).

To work with time references, choose one of the **Time Reference** items in the menu:[Edit] menu or from the pop-up menu of the "Packet List" pane. See The "Edit" Menu.

- **Set Time Reference (toggle)** Toggles the time reference state of the currently selected packet to on or off.
- **Find Next** Find the next time referenced packet in the "Packet List" pane.
- **Find Previous** Find the previous time referenced packet in the "Packet List" pane.

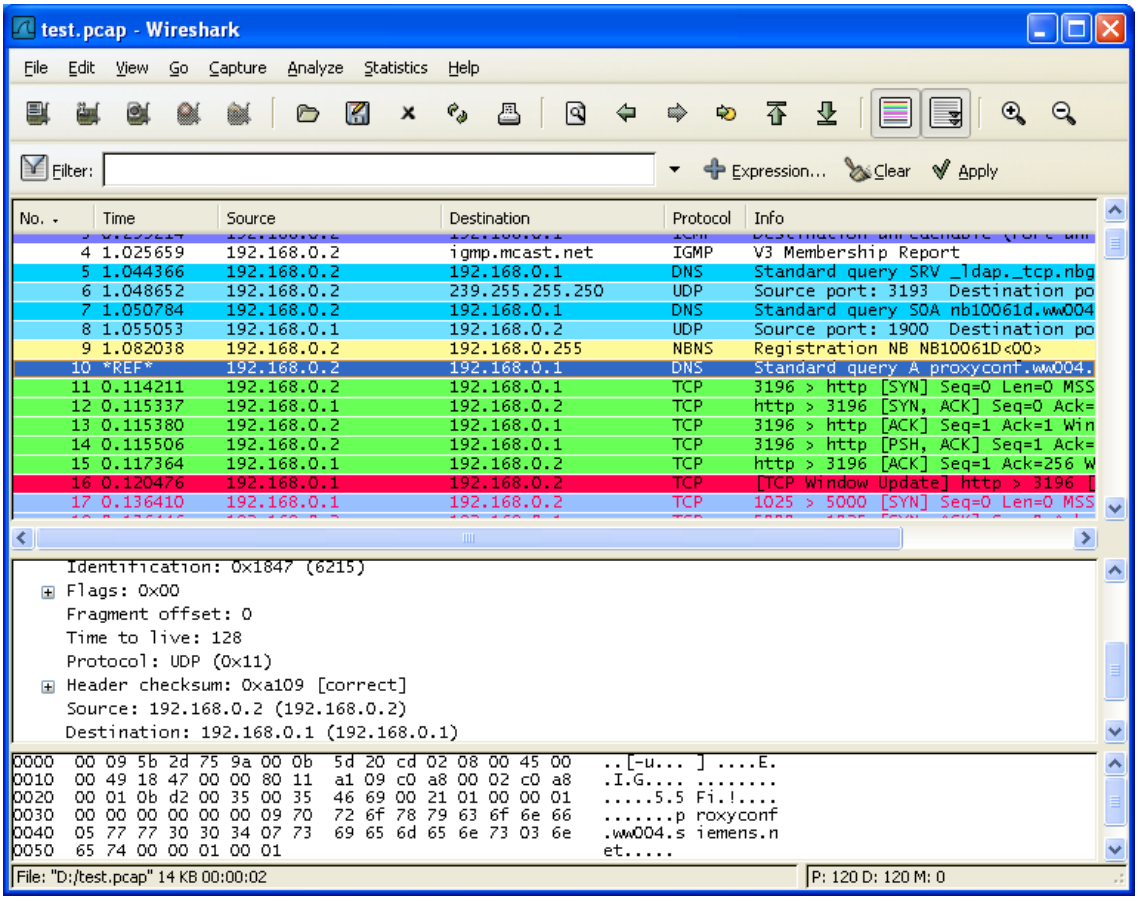


Figure 67. Wireshark showing a time referenced packet

A time referenced packet will be marked with the string *REF* in the Time column (see packet number 10). All subsequent packets will show the time since the last time reference.

Advanced Topics

Introduction

This chapter will describe some of Wireshark's advanced features.

Following Protocol Streams

It can be very helpful to see a protocol in the way that the application layer sees it. Perhaps you are looking for passwords in a Telnet stream, or you are trying to make sense of a data stream. Maybe you just need a display filter to show only the packets in a TLS or SSL stream. If so, Wireshark's ability to follow protocol streams will be useful to you.

To filter to a particular stream, select a TCP, UDP, DCCP, TLS, HTTP, HTTP/2, QUIC or SIP packet in the packet list of the stream/connection you are interested in and then select the menu item **Analyze › Follow › TCP Stream** (or use the context menu in the packet list). Wireshark will set an appropriate display filter and display a dialog box with the data from the stream laid out, as shown in The "Follow TCP Stream" dialog box.

TIP

Following a protocol stream applies a display filter which selects all the packets in the current stream. Some people open the "Follow TCP Stream" dialog and immediately close it as a quick way to isolate a particular stream. Closing the dialog with the "Back" button will reset the display filter if this behavior is not desired.

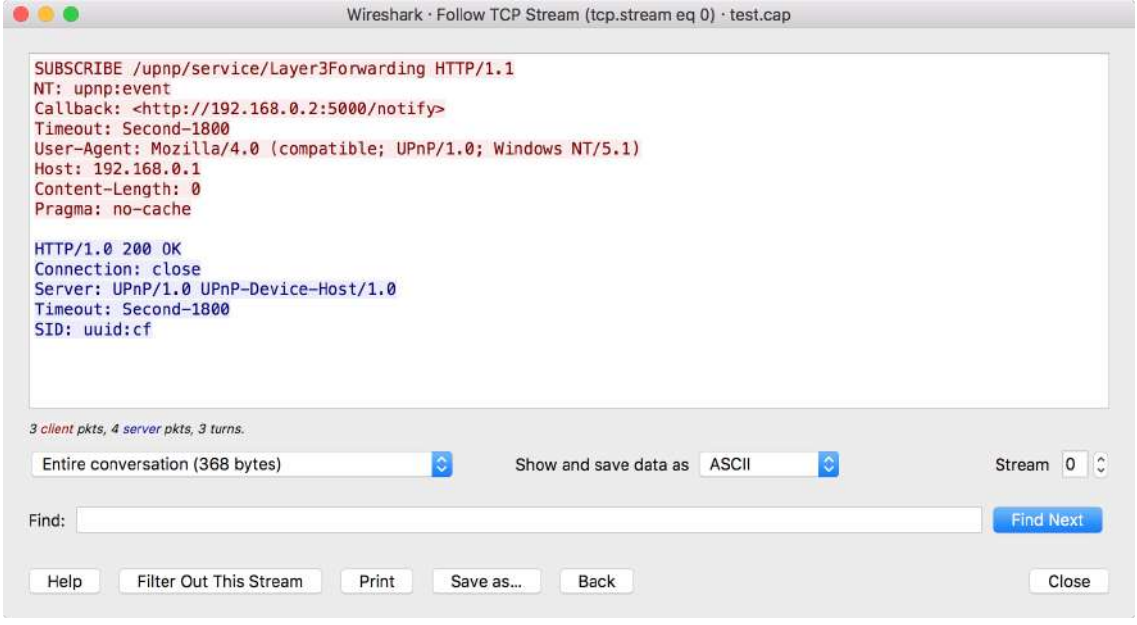


Figure 68. The "Follow TCP Stream" dialog box

The stream content is displayed in the same sequence as it appeared on the network. Non-printable characters are replaced by dots. Traffic from the client to the server is colored red, while traffic

from the server to the client is colored blue. These colors can be changed by opening **Edit › Preferences** and under **Appearance › Font and Colors**, selecting different colors for the **[Sample "Follow Stream" client text]** and **[Sample "Follow Stream" server text]** options.

The stream content won't be updated while doing a live capture. To get the latest content you'll have to reopen the dialog.

You can choose from the following actions:

[Help]

Show this help.

[Filter out this stream]

Apply a display filter removing the current stream data from the display.

[Print]

Print the stream data in the currently selected format.

[Save as...]

Save the stream data in the currently selected format.

[Back]

Close this dialog box and restore the previous display filter.

[Close]

Close this dialog box, leaving the current display filter in effect.

By default, Wireshark displays both client and server data. You can select the **Entire conversation** to switch between both, client to server, or server to client data.

You can choose to view the data in one of the following formats:

ASCII

In this view you see the data from each direction in ASCII. Obviously best for ASCII based protocols, e.g., HTTP.

C Arrays

This allows you to import the stream data into your own C program.

EBCDIC

For the big-iron freaks out there.

HEX Dump

This allows you to see all the data. This will require a lot of screen space and is best used with binary protocols.

UTF-8

Like ASCII, but decode the data as UTF-8.

UTF-16

Like ASCII, but decode the data as UTF-16.

YAML

This allows you to load the stream as YAML.

The YAML output is divided into 2 main sections:

- The `peers` section where for each `peer` you found the peer index, the `host` address and the `port` number.
- The `packets` section where for each `packet` you found the packet number in the original capture, the `peer` index, the packet `index` for this peer, the `timestamp` in seconds and the `data` in base64 encoding.

Example 3. Follow Stream YAML output

```
peers:
  - peer: 0
    host: 127.0.0.1
    port: 54048
  - peer: 1
    host: 127.0.10.1
    port: 5000
packets:
  - packet: 1
    peer: 0
    index: 0
    timestamp: 1599485409.693955274
    data: !!binary |
      aGVsbG8K
  - packet: 3
    peer: 1
    index: 0
    timestamp: 1599485423.885866692
    data: !!binary |
      Ym9uam91cgo=
```

The same example but in old YAML format (before version 3.5):

```
# Packet 1
peer0_0: !!binary |
  aGVsbG8K
# Packet 3
peer1_0: !!binary |
  Ym9uam91cgo=
```

How the old format data can be found in the new format:

New YAML format	Old YAML format	

`...`
`packets:`
`  - packet: AAA`
`  peer: BBB`
`  index: CCC`
`  data: !!binary |`
`  DDD`
	`# Packet AAA`
	`peerBBB_CCC !!binary |`
	`  DDD`
		`AAA: packet number in the original capture`
		`BBB: peer index`
		`CCC: packet index for this peer`
		`DDD: data in base64 encoding`

Raw

This allows you to load the unaltered stream data into a different program for further examination. The display will look the same as the ASCII setting, but “Save As” will result in a binary file.

You can switch between streams using the “Stream” selector.

You can search for text by entering it in the “Find” entry box and pressing **[Find Next]**.

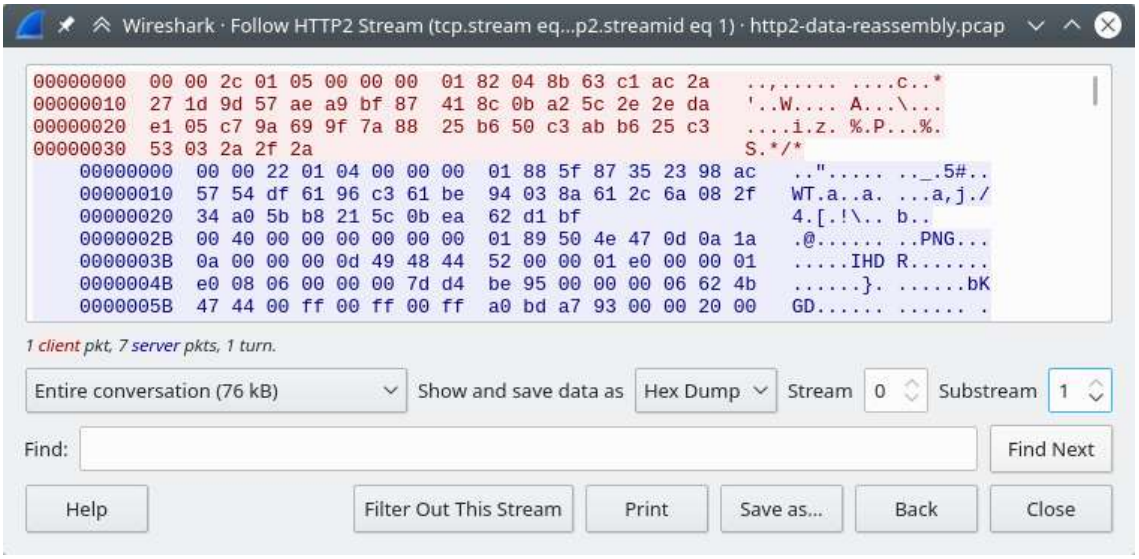


Figure 69. The “Follow HTTP/2 Stream” dialog box

The HTTP/2 Stream dialog is similar to the "Follow TCP Stream" dialog, except for an additional "Substream" dialog field. HTTP/2 Streams are identified by a HTTP/2 Stream Index (field name `http2.streamid`) which are unique within a TCP connection. The “Stream” selector determines the TCP connection whereas the “Substream” selector is used to pick the HTTP/2 Stream ID.

The QUIC protocol is similar, the first number selects the QUIC connection number while the "Substream" field selects the QUIC Stream ID.

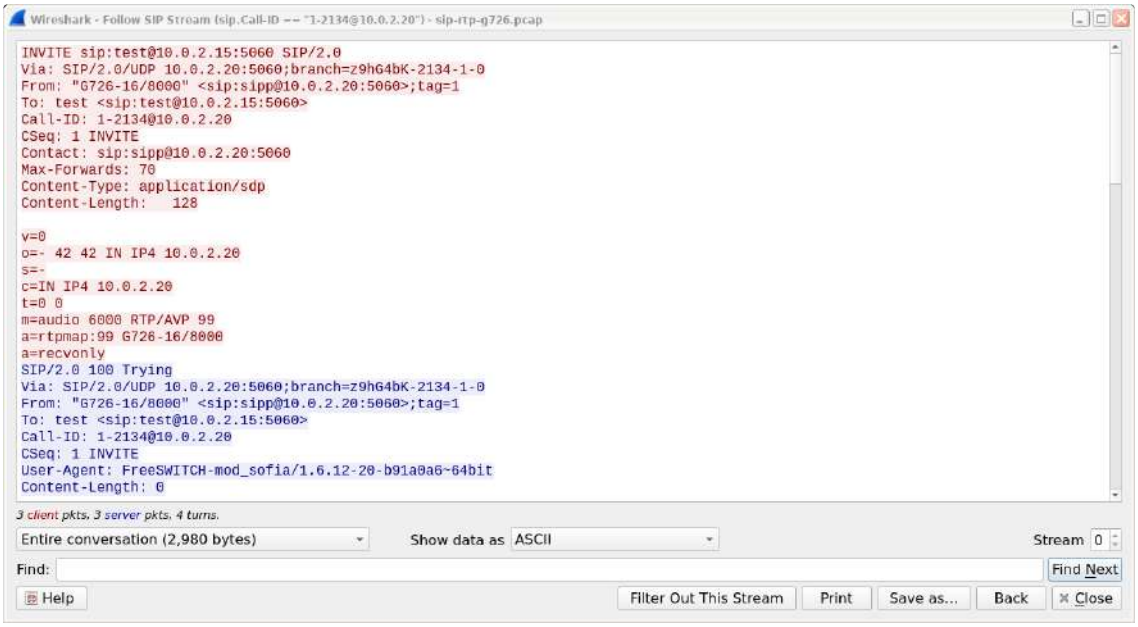


Figure 70. The “Follow SIP Call” dialog box

The SIP call is shown with same dialog, just filter is based on sip.Call-ID field. Count of streams is fixed to 0 and the field is disabled.

Show Packet Bytes

If a selected packet field does not show all the bytes (i.e., they are truncated when displayed) or if they are shown as bytes rather than string or if they require more formatting because they contain an image or HTML then this dialog can be used.

This dialog can also be used to decode field bytes from base64, zlib compressed or quoted-printable and show the decoded bytes as configurable output. It’s also possible to select a subset of bytes setting the start byte and end byte.

You can choose from the following actions:

[Help]

Show this help.

[Print]

Print the bytes in the currently selected format.

[Copy]

Copy the bytes to the clipboard in the currently selected format.

[Save As]

Save the bytes in the currently selected format.

[Close]

Close this dialog box.

You can choose to decode the data from one of the following formats:

None

This is the default which does not decode anything.

Base64

This will decode from Base64.

Compressed

This will decompress the buffer using zlib.

Hex Digits

This will decode from a string of hex digits. Non-hex characters are skipped.

Quoted-Printable

This will decode from a Quoted-Printable string.

ROT-13

This will decode ROT-13 encoded text.

You can choose to view the data in one of the following formats:

ASCII

In this view you see the bytes as ASCII. All control characters and non-ASCII bytes are replaced by dot.

ASCII & Control

In this view all control characters are shown using a UTF-8 symbol and all non-ASCII bytes are replaced by dot.

C Array

This allows you to import the field data into your own C program.

EBCDIC

For the big-iron freaks out there.

Hex Dump

This allows you to see all the data. This will require a lot of screen space and is best used with binary protocols.

HTML

This allows you to see all the data formatted as a HTML document. The HTML supported is what's supported by the Qt QTextEdit class.

Image

This will try to convert the bytes into an image. Most popular formats are supported including PNG, JPEG, GIF, and BMP.

ISO 8859-1

In this view you see the bytes as ISO 8859-1.

Raw

This allows you to load the unaltered stream data into a different program for further examination. The display will show HEX data, but "Save As" will result in a binary file.

UTF-8

In this view you see the bytes as UTF-8.

UTF-16

In this view you see the bytes as UTF-16.

YAML

This will show the bytes as a YAML binary dump.

You can search for text by entering it in the "Find" entry box and pressing **[Find Next]**.

Expert Information

Wireshark keeps track of any anomalies and other items of interest it finds in a capture file and shows them in the Expert Information dialog. The goal is to give you a better idea of uncommon or notable network behavior and to let novice and expert users find network problems faster than manually scanning through the packet list.

WARNING

Expert information is only a hint

Expert information is the starting point for investigation, not the stopping point. Every network is different, and it's up to you to verify that Wireshark's expert information applies to your particular situation. The presence of expert information doesn't necessarily indicate a problem and absence of expert information doesn't necessarily mean everything is OK.

The amount of expert information largely depends on the protocol being used. While dissectors for some common protocols like TCP and IP will show detailed information, other dissectors will show little or none.

The following describes the components of a single expert information entry along with the expert user interface.

Expert Information Entries

Expert information entries are grouped by severity level (described below) and contain the following:

Table 27. Example expert information items

Packet #	Summary	Group	Protocol
592	TCP: [TCP Out-Of-Order] …	Malformed	TCP
1202	DNS: Standard query response …	Protocol	DNS
443	TCP: 80 → 59322 [RST] Seq=12761 Win=0 Len=0	Sequence	TCP

Severity

Every expert information item has a severity level. The following levels are used, from lowest to highest. Wireshark marks them using different colors, which are shown in parentheses:

Chat (blue)
: Information about usual workflow, e.g., a TCP packet with the SYN flag set.

Note (cyan)
: Notable events, e.g., an application returned a common error code such as HTTP 404.

Warn (yellow)
: Warnings, e.g., application returned an unusual error code like a connection problem.

Error (red)
: Serious problems, such as malformed packets.

Summary

Short explanatory text for each expert information item.

Group

Along with severity levels, expert information items are categorized by group. The following groups are currently implemented:

Assumption

The protocol field has incomplete data and was dissected based on assumed value.

Checksum

A checksum was invalid.

Comment

Packet comment.

Debug

Debugging information. You shouldn't see this group in release versions of Wireshark.

Decryption

A decryption issue.

Deprecated

The protocol field has been deprecated.

Malformed

Malformed packet or dissector has a bug. Dissection of this packet aborted.

Protocol

Violation of a protocol's specification (e.g., invalid field values or illegal lengths). Dissection of this packet probably continued.

Reassemble

Problems while reassembling, e.g., not all fragments were available or an exception happened during reassembly.

Request Code

An application request (e.g., File Handle == *x*). Usually assigned the Chat severity level.

Response Code

An application response code indicates a potential problem, e.g., HTTP 404 page not found.

Security

A security problem, e.g., an insecure implementation.

Sequence

A protocol sequence number was suspicious, e.g., it wasn't continuous or a retransmission was detected.

Undecoded

Dissection incomplete or data can't be decoded for other reasons.

It's possible that more groups will be added in the future.

Protocol

The protocol dissector that created the expert information item.

The "Expert Information" Dialog

You can open the expert info dialog by selecting **Analyze** › **Expert Info** or by clicking the expert level indicator in the main status bar.

Right-clicking on an item will allow you to apply or prepare a filter based on the item, copy its summary text, and other tasks.

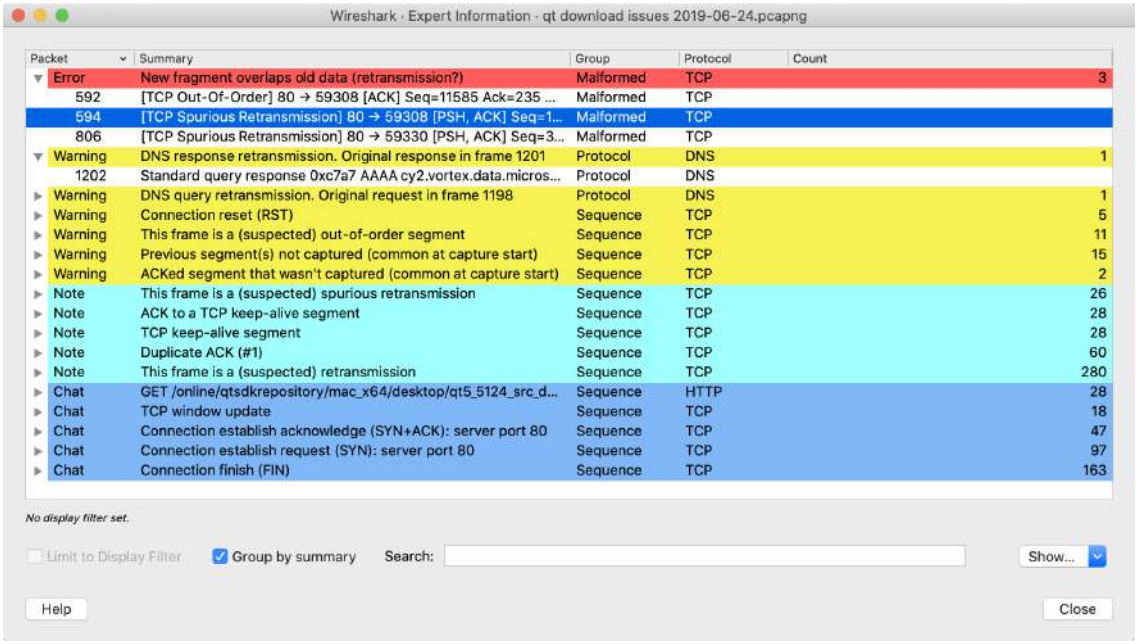


Figure 71. The "Expert Information" dialog box

You can choose from the following actions:

Limit to display filter

Only show expert information items present in packets that match the current display filter.

Group by summary

Group items by their summary instead of the groups described above.

Search

Only show items that match the search string, such as "dns". Regular expressions are supported.

Show...

Lets you show or hide each severity level. For example, you can deselect Chat and Note severities if desired.

[Help]

Takes you to this section of the User's Guide.

[Close]

Closes the dialog

"Colorized" Protocol Details Tree

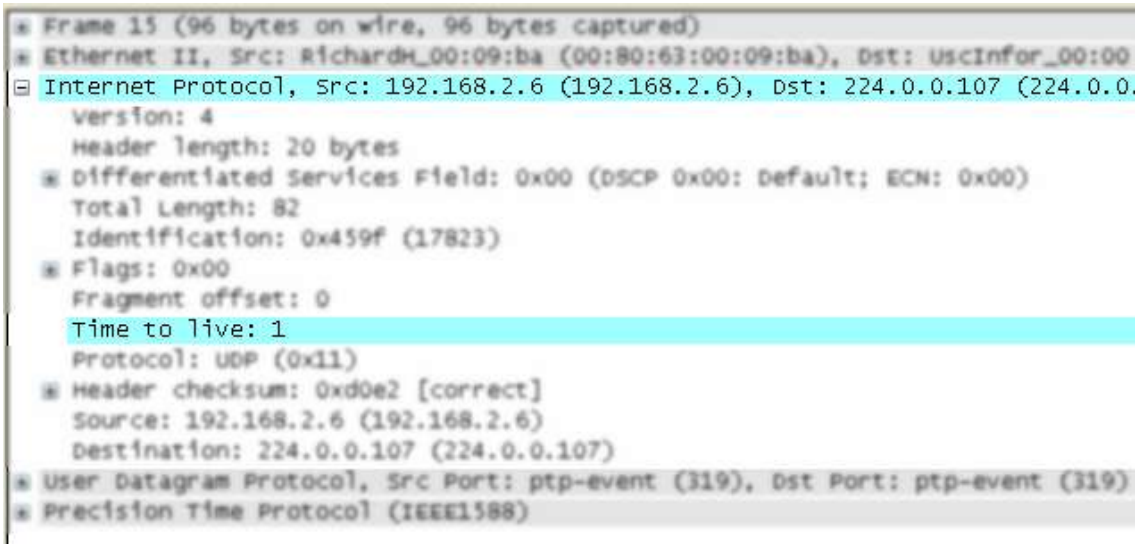


Figure 72. The "Colorized" protocol details tree

The packet detail tree marks fields with expert information based on their severity level color, e.g., "Warning" severities have a yellow background. This color is propagated to the top-level protocol item in the tree in order to make it easy to find the field that created the expert information.

For the example screenshot above, the IP "Time to live" value is very low (only 1), so the corresponding protocol field is marked with a cyan background. To make it easier find that item in the packet tree, the IP protocol toplevel item is marked cyan as well.

"Expert" Packet List Column (Optional)

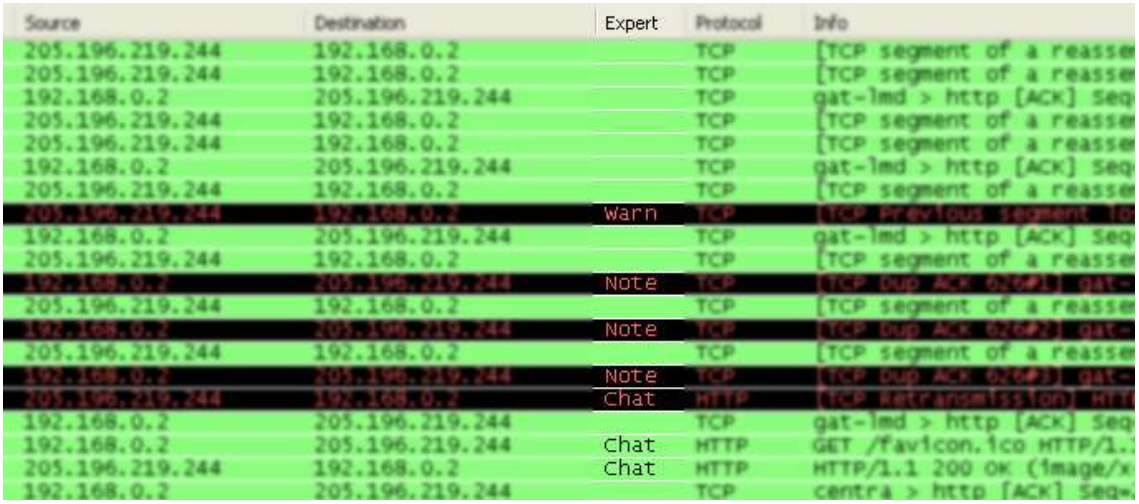


Figure 73. The "Expert" packet list column

An optional "Expert Info Severity" packet list column is available that displays the most significant severity of a packet or stays empty if everything seems OK. This column is not displayed by default but can be easily added using the Preferences Columns page described in Preferences.

TCP Analysis

By default, Wireshark's TCP dissector tracks the state of each TCP session and provides additional information when problems or potential problems are detected. Analysis is done once for each TCP packet when a capture file is first opened. Packets are processed in the order in which they appear in the packet list. You can enable or disable this feature via the "Analyze TCP sequence numbers" TCP dissector preference.

For analysis of data or protocols layered on top of TCP (such as HTTP), see TCP Reassembly.

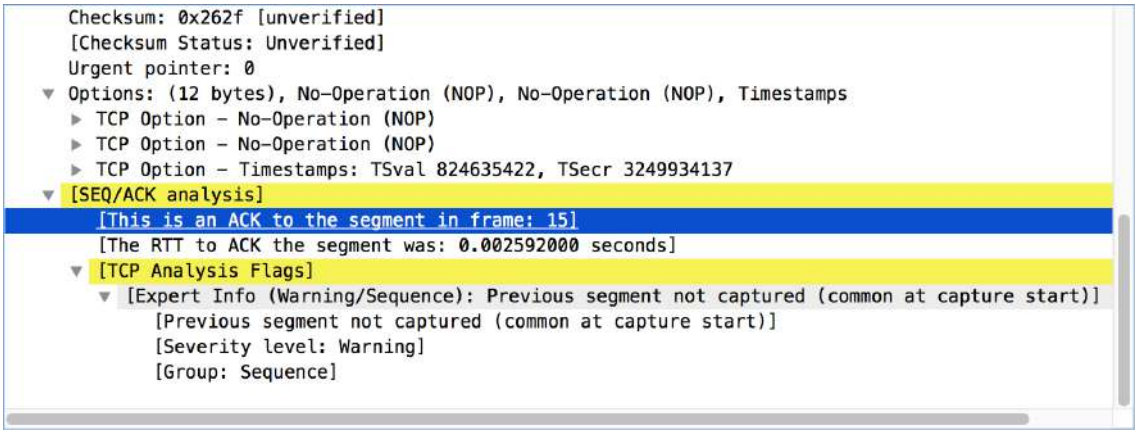


Figure 74. "TCP Analysis" packet detail items

TCP Analysis flags are added to the TCP protocol tree under "SEQ/ACK analysis". Each flag is described below. Terms such as "next expected sequence number" and "next expected acknowledgement number" refer to the following":

Next expected sequence number

The last-seen sequence number plus segment length. Set when there are no analysis flags and for zero window probes. This is initially zero and calculated based on the previous packet in the same TCP flow. Note that this may not be the same as the tcp.nxtseq protocol field.

Next expected acknowledgement number

The last-seen sequence number for segments. Set when there are no analysis flags and for zero window probes.

Last-seen acknowledgment number

Always set. Note that this is not the same as the next expected acknowledgment number.

Last-seen acknowledgment number

Always updated for each packet. Note that this is not the same as the next expected acknowledgment number.

TCP ACKed unseen segment

Set when the expected next acknowledgement number is set for the reverse direction and it's less than the current acknowledgement number.

TCP Dup ACK *<frame>#<acknowledgement number>*

Set when all of the following are true:

- The segment size is zero.
- The window size is non-zero and hasn't changed.
- The next expected sequence number and last-seen acknowledgment number are non-zero (i.e., the connection has been established).
- SYN, FIN, and RST are not set.

TCP Fast Retransmission

Set when all of the following are true:

- This is not a keepalive packet.
- In the forward direction, the segment size is greater than zero or the SYN or FIN is set.
- The next expected sequence number is greater than the current sequence number.
- We have more than two duplicate ACKs in the reverse direction.
- The current sequence number equals the next expected acknowledgement number.
- We saw the last acknowledgement less than 20ms ago.

Supersedes "Out-Of-Order" and "Retransmission".

TCP Keep-Alive

Set when the segment size is zero or one, the current sequence number is one byte less than the next expected sequence number, and none of SYN, FIN, or RST are set.

Supersedes “Fast Retransmission”, “Out-Of-Order”, “Spurious Retransmission”, and “Retransmission”.

TCP Keep-Alive ACK

Set when all of the following are true:

- The segment size is zero.
- The window size is non-zero and hasn’t changed.
- The current sequence number is the same as the next expected sequence number.
- The current acknowledgement number is the same as the last-seen acknowledgement number.
- The most recently seen packet in the reverse direction was a keepalive.
- The packet is not a SYN, FIN, or RST.

Supersedes “Dup ACK” and “ZeroWindowProbeAck”.

TCP Out-Of-Order

Set when all of the following are true:

- This is not a keepalive packet.
- In the forward direction, the segment length is greater than zero or the SYN or FIN is set.
- The next expected sequence number is greater than the current sequence number.
- The next expected sequence number and the next sequence number differ.
- The last segment arrived within the Out-Of-Order RTT threshold. The threshold is either the value shown in the “iRTT” (tcp.analysis.initial_rtt) field under “SEQ/ACK analysis” if it is present, or the default value of 3ms if it is not.

Supersedes “Retransmission”.

TCP Port numbers reused

Set when the SYN flag is set (not SYN+ACK), we have an existing conversation using the same addresses and ports, and the sequence number is different than the existing conversation’s initial sequence number.

TCP Previous segment not captured

Set when the current sequence number is greater than the next expected sequence number.

TCP Spurious Retransmission

Checks for a retransmission based on analysis data in the reverse direction. Set when all of the following are true:

- The SYN or FIN flag is set.
- This is not a keepalive packet.
- The segment length is greater than zero.
- Data for this flow has been acknowledged. That is, the last-seen acknowledgement number has been set.
- The next sequence number is less than or equal to the last-seen acknowledgement number.

Supersedes “Fast Retransmission”, “Out-Of-Order”, and “Retransmission”.

TCP Retransmission

Set when all of the following are true:

- This is not a keepalive packet.
- In the forward direction, the segment length is greater than zero or the SYN or FIN flag is set.
- The next expected sequence number is greater than the current sequence number.

TCP Window Full

Set when the segment size is non-zero, we know the window size in the reverse direction, and our segment size exceeds the window size in the reverse direction.

TCP Window Update

Set when the all of the following are true:

- The segment size is zero.
- The window size is non-zero and not equal to the last-seen window size.
- The sequence number is equal to the next expected sequence number.
- The acknowledgement number is equal to the last-seen acknowledgement number.
- None of SYN, FIN, or RST are set.

TCP ZeroWindow

Set when the receive window size is zero and none of SYN, FIN, or RST are set.

The *window* field in each TCP header advertises the amount of data a receiver can accept. If the receiver can't accept any more data it will set the window value to zero, which tells the sender to pause its transmission. In some specific cases this is normal — for example, a printer might use a zero window to pause the transmission of a print job while it loads or reverses a sheet of paper. However, in most cases this indicates a performance or capacity problem on the receiving end. It might take a long time (sometimes several minutes) to resume a paused connection, even if the underlying condition that caused the zero window clears up quickly.

TCP ZeroWindowProbe

Set when the sequence number is equal to the next expected sequence number, the segment size is one, and last-seen window size in the reverse direction was zero.

If the single data byte from a Zero Window Probe is dropped by the receiver (not ACKed), then a subsequent segment should not be flagged as retransmission if all of the following conditions are true for that segment: * The segment size is larger than one. * The next expected sequence number is one less than the current sequence number.

This affects "Fast Retransmission", "Out-Of-Order", or "Retransmission".

TCP ZeroWindowProbeAck

Set when the all of the following are true:

- The segment size is zero.
- The window size is zero.
- The sequence number is equal to the next expected sequence number.
- The acknowledgement number is equal to the last-seen acknowledgement number.
- The last-seen packet in the reverse direction was a zero window probe.

Supersedes "TCP Dup ACK".

TCP Ambiguous Interpretations

Some captures are quite difficult to analyze automatically, particularly when the time frame may cover both Fast Retransmission and Out-Of-Order packets. A TCP preference allows to switch the precedence of these two interpretations at the protocol level.

TCP Conversation Completeness

TCP conversations are said to be complete when they have both opening and closing handshakes, independently of any data transfer. However, we might be interested in identifying complete conversations with some data sent, and we are using the following bit values to build a filter value on the tcp.completeness field :

- 1 : SYN
- 2 : SYN-ACK
- 4 : ACK
- 8 : DATA
- 16 : FIN
- 32 : RST

For example, a conversation containing only a three-way handshake will be found with the filter 'tcp.completeness==7' (1+2+4) while a complete conversation with data transfer will be found with a longer filter as closing a connection can be associated with FIN or RST packets, or even both : 'tcp.completeness==31 or tcp.completeness==47 or tcp.completeness==63'

Time Stamps

Time stamps, their precisions and all that can be quite confusing. This section will provide you with information about what's going on while Wireshark processes time stamps.

While packets are captured, each packet is time stamped as it comes in. These time stamps will be saved to the capture file, so they also will be available for (later) analysis.

So where do these time stamps come from? While capturing, Wireshark gets the time stamps from the libpcap (Npcap) library, which in turn gets them from the operating system kernel. If the capture data is loaded from a capture file, Wireshark obviously gets the data from that file.

Wireshark Internals

The internal format that Wireshark uses to keep a packet time stamp consists of the date (in days since 1.1.1970) and the time of day (in nanoseconds since midnight). You can adjust the way Wireshark displays the time stamp data in the packet list, see the "Time Display Format" item in the The "View" Menu for details.

While reading or writing capture files, Wireshark converts the time stamp data between the capture file format and the internal format as required.

While capturing, Wireshark uses the libpcap (Npcap) capture library which supports microsecond resolution. Unless you are working with specialized capturing hardware, this resolution should be adequate.

Capture File Formats

Every capture file format that Wireshark knows supports time stamps. The time stamp precision supported by a specific capture file format differs widely and varies from one second "0" to one nanosecond "0.123456789". Most file formats store the time stamps with a fixed precision (e.g., microseconds), while some file formats are even capable of storing the time stamp precision itself

(whatever the benefit may be).

The common libpcap capture file format that is used by Wireshark (and a lot of other tools) supports a fixed microsecond resolution “0.123456” only.

Writing data into a capture file format that doesn’t provide the capability to store the actual precision will lead to loss of information. For example, if you load a capture file with nanosecond resolution and store the capture data in a libpcap file (with microsecond resolution) Wireshark obviously must reduce the precision from nanosecond to microsecond.

Accuracy

People often ask “Which time stamp accuracy is provided by Wireshark?”. Well, Wireshark doesn’t create any time stamps itself but simply gets them from “somewhere else” and displays them. So accuracy will depend on the capture system (operating system, performance, etc.) that you use. Because of this, the above question is difficult to answer in a general way.

NOTE	USB connected network adapters often provide a very bad time stamp accuracy. The incoming packets have to take “a long and winding road” to travel through the USB cable until they actually reach the kernel. As the incoming packets are time stamped when they are processed by the kernel, this time stamping mechanism becomes very inaccurate. Don’t use USB connected NICs when you need precise time stamp accuracy.

Time Zones

If you travel across the planet, time zones can be confusing. If you get a capture file from somewhere around the world time zones can even be a lot more confusing ;-)

First of all, there are two reasons why you may not need to think about time zones at all:

- You are only interested in the time differences between the packet time stamps and don’t need to know the exact date and time of the captured packets (which is often the case).
- You don’t get capture files from different time zones than your own, so there are simply no time zone problems. For example, everyone in your team is working in the same time zone as yourself.

What are time zones?

People expect that the time reflects the sunset. Dawn should be in the morning maybe around 06:00 and dusk in the evening maybe at 20:00. These times will obviously vary depending on the season. It would be very confusing if everyone on earth would use the same global time as this would correspond to the sunset only at a small part of the world.

For that reason, the earth is split into several different time zones, each zone with a local time that corresponds to the local sunset.

The time zone's base time is UTC (Coordinated Universal Time) or Zulu Time (military and aviation). The older term GMT (Greenwich Mean Time) shouldn't be used as it is slightly incorrect (up to 0.9 seconds difference to UTC). The UTC base time equals to 0 (based at Greenwich, England) and all time zones have an offset to UTC between -12 to +14 hours!

For example: If you live in Berlin, you are in a time zone one hour earlier than UTC, so you are in time zone "+1" (time difference in hours compared to UTC). If it's 3 o'clock in Berlin it's 2 o'clock in UTC "at the same moment".

Be aware that at a few places on earth don't use time zones with even hour offsets (e.g., New Delhi uses UTC+05:30)!

Further information can be found at: https://en.wikipedia.org/wiki/Time_zone and https://en.wikipedia.org/wiki/Coordinated_Universal_Time.

What is daylight saving time (DST)?

Daylight Saving Time (DST), also known as Summer Time is intended to "save" some daylight during the summer months. To do this, a lot of countries (but not all!) add a DST hour to the already existing UTC offset. So you may need to take another hour (or in very rare cases even two hours!) difference into your "time zone calculations".

Unfortunately, the date at which DST actually takes effect is different throughout the world. You may also note, that the northern and southern hemispheres have opposite DST's (e.g., while it's summer in Europe it's winter in Australia).

Keep in mind: UTC remains the same all year around, regardless of DST!

Further information can be found at https://en.wikipedia.org/wiki/Daylight_saving.

Further time zone and DST information can be found at https://wwp.greenwichmeantime.com/ and https://www.timeanddate.com/worldclock/.

Set your computer's time correctly!

If you work with people around the world it's very helpful to set your computer's time and time zone right.

You should set your computers time and time zone in the correct sequence:

1. Set your time zone to your current location
2. Set your computer's clock to the local time

This way you will tell your computer both the local time and also the time offset to UTC. Many organizations simply set the time zone on their servers and networking gear to UTC in order to make coordination and troubleshooting easier.

TIP	If you travel around the world, it's an often-made mistake to adjust the hours of your computer clock to the local time. Don't adjust the hours but your time zone setting instead! For your computer, the time is essentially the same as before, you are simply in a different time zone with a different local time.

You can use the Network Time Protocol (NTP) to automatically adjust your computer to the correct time, by synchronizing it to Internet NTP clock servers. NTP clients are available for all operating systems that Wireshark supports (and for a lot more), for examples see http://www.ntp.org/.

Wireshark and Time Zones

So what's the relationship between Wireshark and time zones anyway?

Wireshark's native capture file format (libpcap format), and some other capture file formats, such as the Windows Sniffer, *Peek, Sun snoop formats, and newer versions of the Microsoft Network Monitor and Network Instruments/Viavi Observer formats, save the arrival time of packets as UTC values. UN*X systems, and "Windows NT based" systems represent time internally as UTC. When Wireshark is capturing, no conversion is necessary. However, if the system time zone is not set correctly, the system's UTC time might not be correctly set even if the system clock appears to display correct local time. When capturing, Npcap has to convert the time to UTC before supplying it to Wireshark. If the system's time zone is not set correctly, that conversion will not be done correctly.

Other capture file formats, such as the OOS-based Sniffer format and older versions of the Microsoft Network Monitor and Network Instruments/Viavi Observer formats, save the arrival time of packets as local time values.

Internally to Wireshark, time stamps are represented in UTC. This means that when reading capture files that save the arrival time of packets as local time values, Wireshark must convert those local time values to UTC values.

Wireshark in turn will display the time stamps always in local time. The displaying computer will convert them from UTC to local time and displays this (local) time. For capture files saving the arrival time of packets as UTC values, this means that the arrival time will be displayed as the local

time in your time zone, which might not be the same as the arrival time in the time zone in which the packet was captured. For capture files saving the arrival time of packets as local time values, the conversion to UTC will be done using your time zone's offset from UTC and DST rules, which means the conversion will not be done correctly; the conversion back to local time for display might undo this correctly, in which case the arrival time will be displayed as the arrival time in which the packet was captured.

Table 28. Time zone examples for UTC arrival times (without DST)

	Los Angeles	**New York**	**Madrid**	**London**	**Berlin**	**Tokyo**
Capture File (UTC)	10:00	10:00	10:00	10:00	10:00	10:00
Local Offset to UTC	-8	-5	-1	0	+1	+9
Displayed Time (Local Time)	02:00	05:00	09:00	10:00	11:00	19:00

For example, let's assume that someone in Los Angeles captured a packet with Wireshark at exactly 2 o'clock local time and sends you this capture file. The capture file's time stamp will be represented in UTC as 10 o'clock. You are located in Berlin and will see 11 o'clock on your Wireshark display.

Now you have a phone call, video conference or Internet meeting with that one to talk about that capture file. As you are both looking at the displayed time on your local computers, the one in Los Angeles still sees 2 o'clock but you in Berlin will see 11 o'clock. The time displays are different as both Wireshark displays will show the (different) local times at the same point in time.

Conclusion: You may not bother about the date/time of the time stamp you currently look at unless you must make sure that the date/time is as expected. So, if you get a capture file from a different time zone and/or DST, you'll have to find out the time zone/DST difference between the two local times and "mentally adjust" the time stamps accordingly. In any case, make sure that every computer in question has the correct time and time zone setting.

Packet Reassembly

What Is It?

Network protocols often need to transport large chunks of data which are complete in themselves, e.g., when transferring a file. The underlying protocol might not be able to handle that chunk size (e.g., limitation of the network packet size), or is stream-based like TCP, which doesn't know data chunks at all.

In that case the network protocol has to handle the chunk boundaries itself and (if required) spread the data over multiple packets. It obviously also needs a mechanism to determine the chunk

boundaries on the receiving side.

Wireshark calls this mechanism reassembly, although a specific protocol specification might use a different term for this (e.g., desegmentation, defragmentation, etc.).

How Wireshark Handles It

For some of the network protocols Wireshark knows of, a mechanism is implemented to find, decode and display these chunks of data. Wireshark will try to find the corresponding packets of this chunk, and will show the combined data as additional tabs in the “Packet Bytes” pane (for information about this pane. See The “Packet Bytes” Pane).

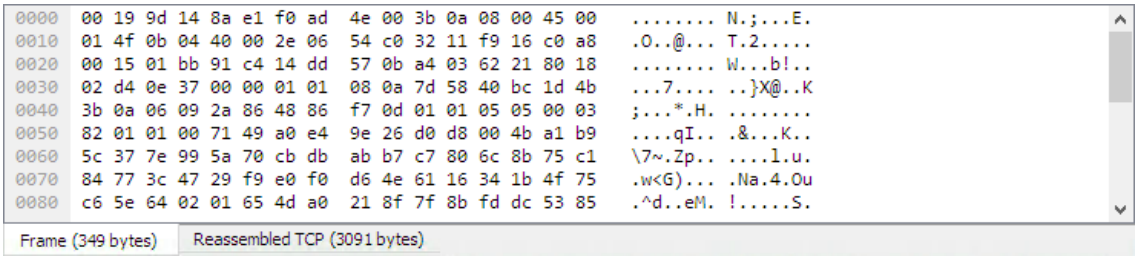


Figure 75. The “Packet Bytes” pane with a reassembled tab

Reassembly might take place at several protocol layers, so it’s possible that multiple tabs in the “Packet Bytes” pane appear.

NOTE	You will find the reassembled data in the last packet of the chunk.

For example, in a *HTTP* GET response, the requested data (e.g., an HTML page) is returned. Wireshark will show the hex dump of the data in a new tab “Uncompressed entity body” in the “Packet Bytes” pane.

Reassembly is enabled in the preferences by default but can be disabled in the preferences for the protocol in question. Enabling or disabling reassembly settings for a protocol typically requires two things:

1. The lower-level protocol (e.g., TCP) must support reassembly. Often this reassembly can be enabled or disabled via the protocol preferences.
2. The higher-level protocol (e.g., HTTP) must use the reassembly mechanism to reassemble fragmented protocol data. This too can often be enabled or disabled via the protocol preferences.

The tooltip of the higher-level protocol setting will notify you if and which lower-level protocol setting also has to be considered.

TCP Reassembly

Protocols such as HTTP or TLS are likely to span multiple TCP segments. The TCP protocol preference “Allow subdissector to reassemble TCP streams” (enabled by default) makes it possible

for Wireshark to collect a contiguous sequence of TCP segments and hand them over to the higher-level protocol (for example, to reconstruct a full HTTP message). All but the final segment will be marked with "[TCP segment of a reassembled PDU]" in the packet list.

Disable this preference to reduce memory and processing overhead if you are only interested in TCP sequence number analysis (TCP Analysis). Keep in mind, though, that higher-level protocols might be wrongly dissected. For example, HTTP messages could be shown as "Continuation" and TLS records could be shown as "Ignored Unknown Record". Such results can also be observed if you start capturing while a TCP connection was already started or when TCP segments are lost or delivered out-of-order.

To reassemble of out-of-order TCP segments, the TCP protocol preference "Reassemble out-of-order segments" (currently disabled by default) must be enabled in addition to the previous preference. If all packets are received in-order, this preference will not have any effect. Otherwise (if missing segments are encountered while sequentially processing a packet capture), it is assuming that the new and missing segments belong to the same PDU. Caveats:

- Lost packets are assumed to be received out-of-order or retransmitted later. Applications usually retransmit segments until these are acknowledged, but if the packet capture drops packets, then Wireshark will not be able to reconstruct the TCP stream. In such cases, you can try to disable this preference and hopefully have a partial dissection instead of seeing just "[TCP segment of a reassembled PDU]" for every TCP segment.
- When doing a capture in monitor mode (IEEE 802.11), packets are more likely to get lost due to signal reception issues. In that case it is recommended to disable the option.
- If the new and missing segments are in fact part of different PDUs, then processing is currently delayed until no more segments are missing, even if the begin of the missing segments completed a PDU. For example, assume six segments forming two PDUs `ABC` and `DEF`. When received as `ABECDF`, an application can start processing the first PDU after receiving `ABEC`. Wireshark however requires the missing segment `D` to be received as well. This issue will be addressed in the future.
- In the GUI and during a two-pass dissection (`tshark -2`), the previous scenario will display both PDUs in the packet with last segment (`F`) rather than displaying it in the first packet that has the final missing segment of a PDU. This issue will be addressed in the future.
- When enabled, fields such as the SMB "Time from request" (`smb.time`) might be smaller if the request follows other out-of-order segments (this reflects application behavior). If the previous scenario however occurs, then the time of the request is based on the frame where all missing segments are received.

Regardless of the setting of these two reassembly-related preferences, you can always use the "Follow TCP Stream" option (Following Protocol Streams) which displays segments in the expected order.

Name Resolution

Name resolution tries to convert some of the numerical address values into a human readable format. There are two possible ways to do these conversions, depending on the resolution to be done: calling system/network services (like the gethostname() function) and/or resolve from Wireshark specific configuration files. For details about the configuration files Wireshark uses for name resolution and alike, see Files and Folders.

The name resolution feature can be enabled individually for the protocol layers listed in the following sections.

Name Resolution Drawbacks

Name resolution can be invaluable while working with Wireshark and may even save you hours of work. Unfortunately, it also has its drawbacks.

- *Name resolution can often fail.* The name to be resolved might simply be unknown by the name servers asked, or the servers are just not available and the name is also not found in Wireshark's configuration files.
- *Resolved names might not be available.* Wireshark obtains name resolution information from a variety of sources, including DNS servers, the capture file itself (e.g., for a pcapng file), and the *hosts* files on your system and in your profile directory. The resolved names might not be available if you open the capture file later or on a different machine. As a result, each time you or someone else opens a particular capture file it may look slightly different due to changing environments.
- *DNS may add additional packets to your capture file.* You might run into the observer effect if the extra traffic from Wireshark's DNS queries and responses affects the problem you're trying to troubleshoot or any subsequent analysis.

  The same sort of thing can happen when capturing over a remote connection, e.g., SSH or RDP.

- *Resolved DNS names are cached by Wireshark.* This is required for acceptable performance. However, if the name resolution information should change while Wireshark is running, Wireshark won't notice a change in the name resolution information once it gets cached. If this information changes while Wireshark is running, e.g., a new DHCP lease takes effect, Wireshark won't notice it.

Name resolution in the packet list is done while the list is filled. If a name can be resolved after a packet is added to the list, its former entry won't be changed. As the name resolution results are cached, you can use **View › Reload** to rebuild the packet list with the correctly resolved names. However, this isn't possible while a capture is in progress.

Ethernet Name Resolution (MAC Layer)

Try to resolve an Ethernet MAC address (e.g., 00:09:5b:01:02:03) to a human readable name.

ARP name resolution (system service): Wireshark will ask the operating system to convert an Ethernet address to the corresponding IP address (e.g. 00:09:5b:01:02:03 → 192.168.0.1).

Ethernet codes (ethers file): If the ARP name resolution failed, Wireshark tries to convert the Ethernet address to a known device name, which has been assigned by the user using an *ethers* file (e.g., 00:09:5b:01:02:03 → homerouter).

Ethernet manufacturer codes (manuf file): If neither ARP or ethers returns a result, Wireshark tries to convert the first 3 bytes of an ethernet address to an abbreviated manufacturer name, which has been assigned by the IEEE (e.g. 00:09:5b:01:02:03 → Netgear_01:02:03).

IP Name Resolution (Network Layer)

Try to resolve an IP address (e.g., 216.239.37.99) to a human readable name.

DNS name resolution (system/library service): Wireshark will use a name resolver to convert an IP address to the hostname associated with it (e.g., 216.239.37.99 → www.1.google.com).

Most applications use synchronously DNS name resolution. For example, your web browser must resolve the host name portion of a URL before it can connect to the server. Capture file analysis is different. A given file might have hundreds, thousands, or millions of IP addresses so for usability and performance reasons Wireshark uses asynchronous resolution. Both mechanisms convert IP addresses to human readable (domain) names and typically use different sources such as the system hosts file (*/etc/hosts*) and any configured DNS servers.

Since Wireshark doesn't wait for DNS responses, the host name for a given address might be missing from a given packet when you view it the first time but be present when you view it subsequent times.

You can adjust name resolution behavior in the Name Resolution section in the Preferences Dialog. You can control resolution itself by adding a *hosts* file to your personal configuration directory. You can also edit your system *hosts* file, but that isn't generally recommended.

TCP/UDP Port Name Resolution (Transport Layer)

Try to resolve a TCP/UDP port (e.g., 80) to a human readable name.

TCP/UDP port conversion (system service): Wireshark will ask the operating system to convert a TCP or UDP port to its well-known name (e.g., 80 → http).

VLAN ID Resolution

To get a descriptive name for a VLAN tag ID a vlans file can be used.

SS7 Point Code Resolution

To get a node name for a SS7 point code a ss7pcs file can be used.

Checksums

Several network protocols use checksums to ensure data integrity. Applying checksums as described here is also known as *redundancy checking*.

What are checksums for?

Checksums are used to ensure the integrity of data portions for data transmission or storage. A checksum is basically a calculated summary of such a data portion.

Network data transmissions often produce errors, such as toggled, missing or duplicated bits. As a result, the data received might not be identical to the data transmitted, which is obviously a bad thing.

Because of these transmission errors, network protocols very often use checksums to detect such errors. The transmitter will calculate a checksum of the data and transmits the data together with the checksum. The receiver will calculate the checksum of the received data with the same algorithm as the transmitter. If the received and calculated checksums don't match a transmission error has occurred.

Some checksum algorithms are able to recover (simple) errors by calculating where the expected error must be and repairing it.

If there are errors that cannot be recovered, the receiving side throws away the packet. Depending on the network protocol, this data loss is simply ignored or the sending side needs to detect this loss somehow and retransmits the required packet(s).

Using a checksum drastically reduces the number of undetected transmission errors. However, the usual checksum algorithms cannot guarantee an error detection of 100%, so a very small number of transmission errors may remain undetected.

There are several different kinds of checksum algorithms; an example of an often used checksum algorithm is CRC32. The checksum algorithm actually chosen for a specific network protocol will depend on the expected error rate of the network medium, the importance of error detection, the processor load to perform the calculation, the performance needed and many other things.

Further information about checksums can be found at: https://en.wikipedia.org/wiki/Checksum.

Wireshark Checksum Validation

Wireshark will validate the checksums of many protocols, e.g., IP, TCP, UDP, etc.

It will do the same calculation as a “normal receiver” would do, and shows the checksum fields in the packet details with a comment, e.g., [correct] or [invalid, must be 0x12345678].

Checksum validation can be switched off for various protocols in the Wireshark protocol preferences, e.g., to (very slightly) increase performance.

If the checksum validation is enabled and it detected an invalid checksum, features like packet reassembly won’t be processed. This is avoided as incorrect connection data could “confuse” the internal database.

Checksum Offloading

The checksum calculation might be done by the network driver, protocol driver or even in hardware.

For example: The Ethernet transmitting hardware calculates the Ethernet CRC32 checksum and the receiving hardware validates this checksum. If the received checksum is wrong Wireshark won’t even see the packet, as the Ethernet hardware internally throws away the packet.

Higher-level checksums are “traditionally” calculated by the protocol implementation and the completed packet is then handed over to the hardware.

Recent network hardware can perform advanced features such as IP checksum calculation, also known as checksum offloading. The network driver won’t calculate the checksum itself but will simply hand over an empty (zero or garbage filled) checksum field to the hardware.

NOTE	Checksum offloading often causes confusion as the network packets to be transmitted are handed over to Wireshark before the checksums are actually calculated. Wireshark gets these “empty” checksums and displays them as invalid, even though the packets will contain valid checksums when they leave the network hardware later.

Checksum offloading can be confusing and having a lot of [invalid] messages on the screen can be quite annoying. As mentioned above, invalid checksums may lead to unreassembled packets, making the analysis of the packet data much harder.

You can do two things to avoid this checksum offloading problem:

- Turn off the checksum offloading in the network driver, if this option is available.
- Turn off checksum validation of the specific protocol in the Wireshark preferences. Recent releases of Wireshark disable checksum validation by default due to the prevalence of offloading in modern hardware and operating systems.

Statistics

Introduction

Wireshark provides a wide range of network statistics which can be accessed via the **Statistics** menu.

These statistics range from general information about the loaded capture file (like the number of captured packets), to statistics about specific protocols (e.g., statistics about the number of HTTP requests and responses captured).

General statistics

- **Capture File Properties** about the capture file.
- **Protocol Hierarchy** of the captured packets.
- **Conversations** e.g., traffic between specific IP addresses.
- **Endpoints** e.g., traffic to and from IP addresses.
- **I/O Graphs** visualizing the number of packets (or similar) in time.

Protocol specific statistics

- **Service Response Time** between request and response of some protocols.
- Various other protocol specific statistics.

NOTE	The protocol specific statistics require detailed knowledge about the specific protocol. Unless you are familiar with that protocol, statistics about it may be difficult to understand.

Wireshark has many other statistics windows that display detailed information about specific protocols and might be described in a later version of this document.

Some of these statistics are described at https://gitlab.com/wireshark/wireshark/wikis/Statistics.

The “Capture File Properties” Dialog

General information about the current capture file.

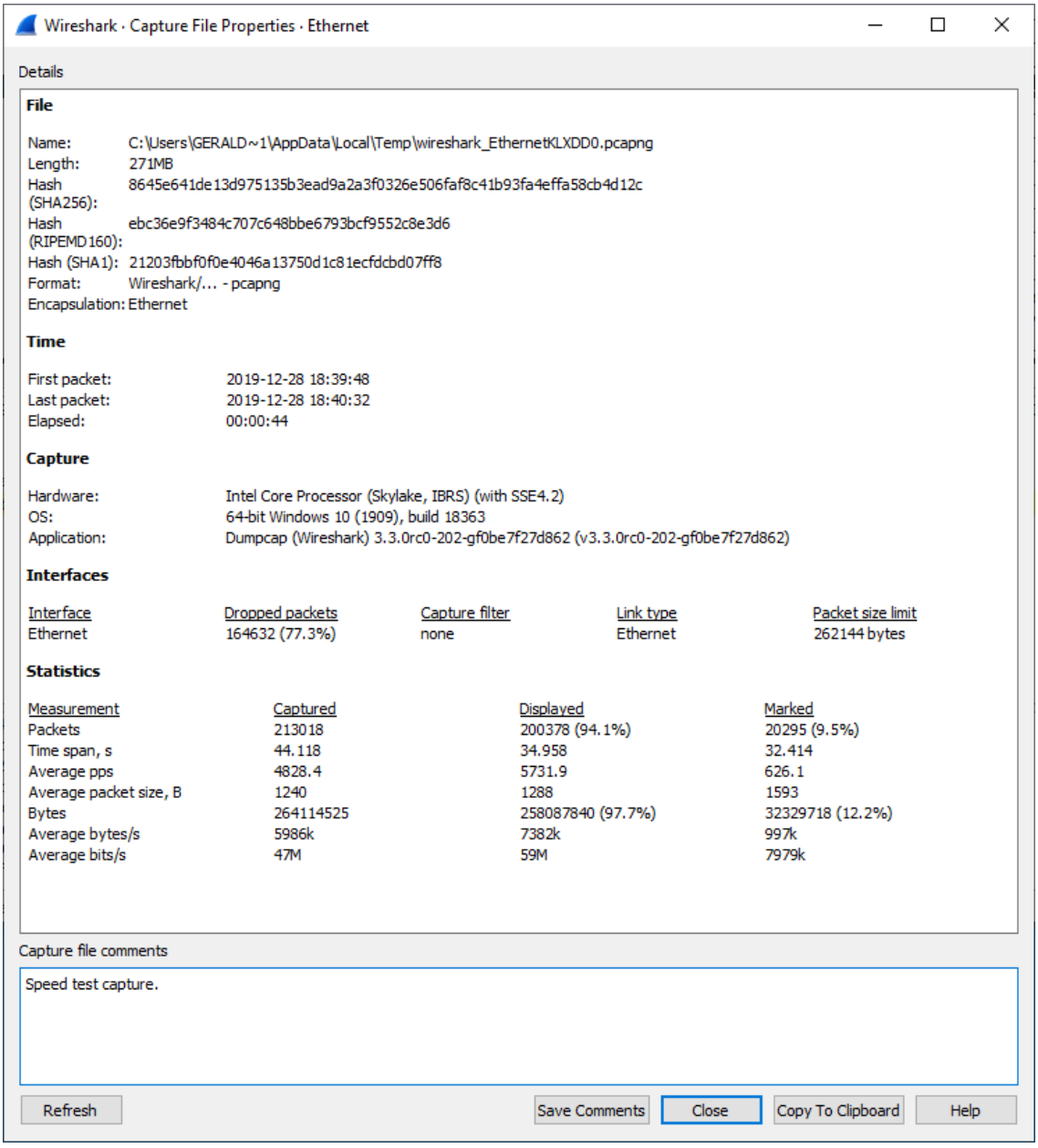


Figure 76. The “Capture File Properties” dialog

This dialog shows the following information:

Details

Notable information about the capture file.

File

General information about the capture file, including its full path, size, cryptographic hashes, file format, and encapsulation.

Time

The timestamps of the first and the last packet in the file along with their difference.

Capture

Information about the capture environment. This will only be shown for live captures or if this information is present in a saved capture file. The pcapng format supports this, while pcap doesn't.

Interfaces

Information about the capture interface or interfaces.

Statistics

A statistical summary of the capture file. If a display filter is set, you will see values in the *Captured* column, and if any packets are marked, you will see values in the *Marked* column. The values in the *Captured* column will remain the same as before, while the values in the *Displayed* column will reflect the values corresponding to the packets shown in the display. The values in the *Marked* column will reflect the values corresponding to the marked packages.

Capture file comments

Some capture file formats (notably pcapng) allow a text comment for the entire file. You can view and edit this comment here.

[Refresh]

Updates the information in the dialog.

[Save Comments]

Saves the contents of the "Capture file comments" text entry.

[Close]

Closes the dialog

[Copy To Clipboard]

Copies the "Details" information to the clipboard.

[Help]

Opens this section of the User's Guide.

Resolved Addresses

The Resolved Addresses window shows the list of resolved addresses and their host names. Users can choose the `Hosts` field to display IPv4 and IPv6 addresses only. In this case, the dialog displays host names for each IP address in a capture file with a known host. This host is typically taken from DNS answers in a capture file. In case of an unknown host name, users can populate it based on a reverse DNS lookup. To do so, follow these steps:

1. Enable `Resolve Network Addresses` in the **View › Name Resolution** menu as this option is disabled by default.

2. Select `Use an external network name resolver` in the **Preferences › Name Resolution** menu. This option is enabled by default.

NOTE	The resolved addresses are not updated automatically after a user changes the settings. To display newly available names, the user has to reopen the dialog.

The `Ports` tab shows the list of service names, ports and types.

Wireshark reads the entries for port mappings from the `hosts` service configuration files. See Configuration Files section for more information.

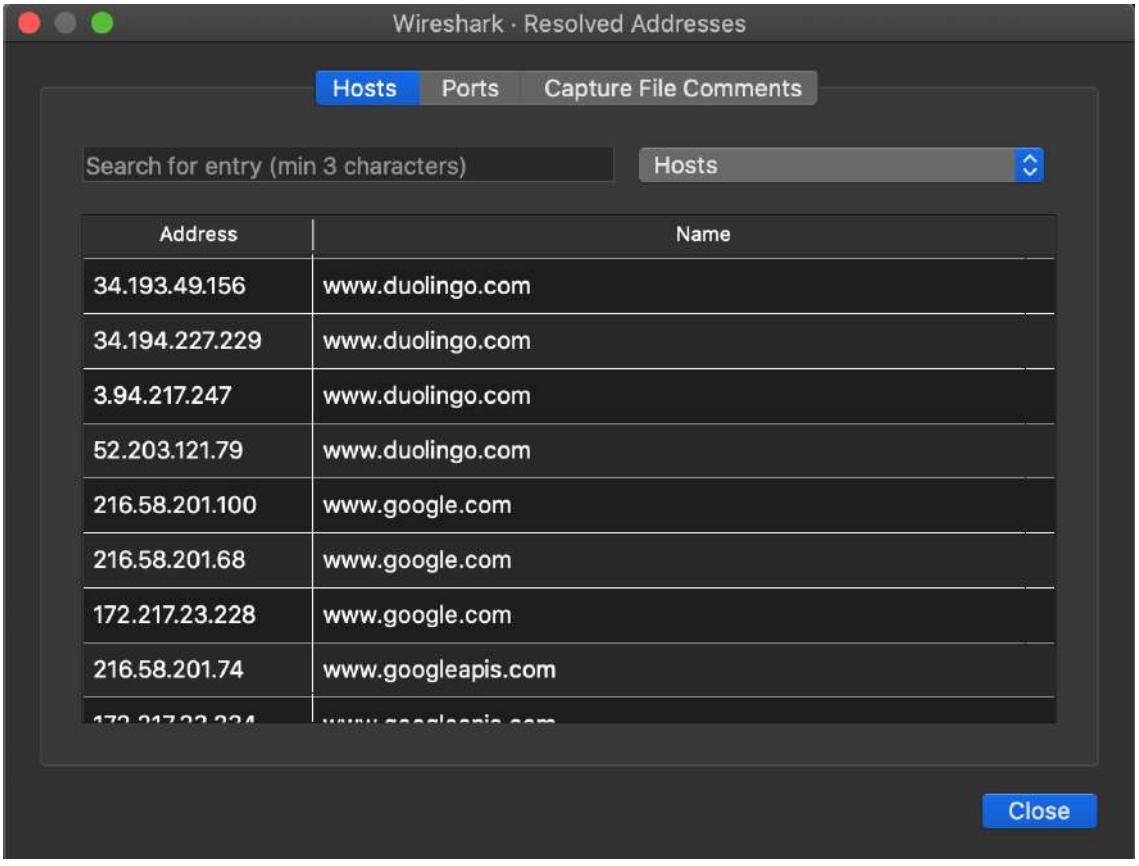


Figure 77. Resolved Addresses window

The “Protocol Hierarchy” Window

The protocol hierarchy of the captured packets.

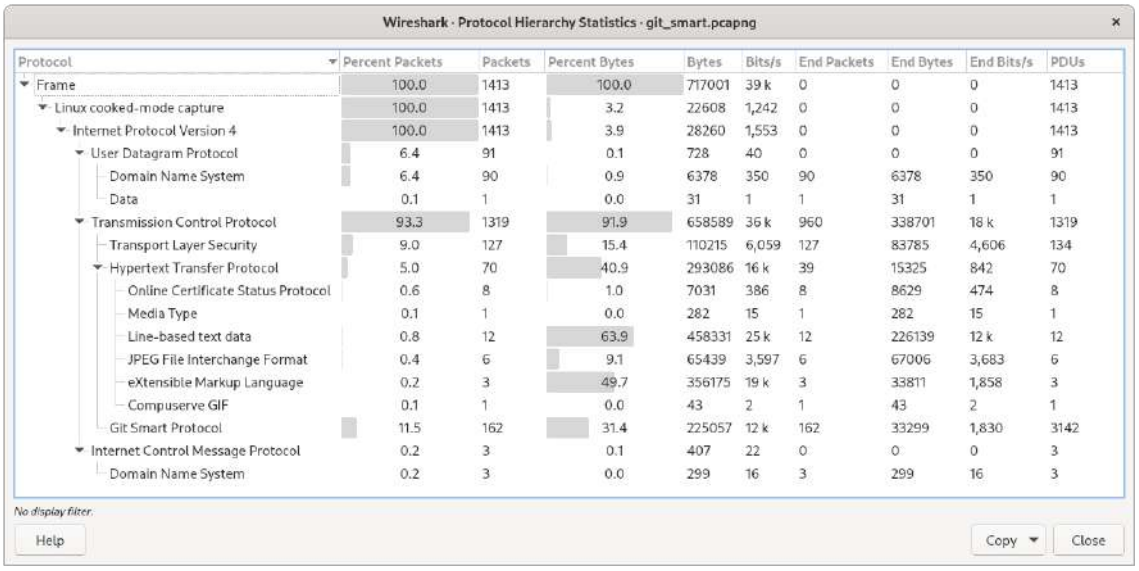


Figure 78. The "Protocol Hierarchy" Window

This is a tree of all the protocols in the capture. Each row contains the statistical values of one protocol. Two of the columns (*Percent Packets* and *Percent Bytes*) serve double duty as bar graphs. If a display filter is set it will be shown at the bottom.

The **[Copy]** button will let you copy the window contents as CSV or YAML.

Protocol hierarchy columns

Protocol

This protocol's name.

Percent Packets

The percentage of protocol packets relative to all packets in the capture.

Packets

The total number of packets that contain this protocol.

Percent Bytes

The percentage of protocol bytes relative to the total bytes in the capture.

Bytes

The total number of bytes of this protocol.

Bits/s

The bandwidth of this protocol relative to the capture time.

End Packets

The absolute number of packets of this protocol where it was the highest protocol in the stack (last dissected).

End Bytes

The absolute number of bytes of this protocol where it was the highest protocol in the stack (last dissected).

End Bits/s

The bandwidth of this protocol relative to the capture time where was the highest protocol in the stack (last dissected).

PDUs

The total number of PDUs of this protocol.

Packets usually contain multiple protocols. As a result, more than one protocol will be counted for each packet. Example: In the screenshot 100% of packets are IP and 99.3% are TCP (which is together much more than 100%).

Protocol layers can consist of packets that won't contain any higher layer protocol, so the sum of all higher layer packets may not sum to the protocol's packet count. This can be caused by segments and fragments reassembled in other frames, TCP protocol overhead, and other undissected data. Example: In the screenshot 99.3% of the packets are TCP but the sum of the subprotocols (TLS, HTTP, Git, etc.) is much less.

A single packet can contain the same protocol more than once. In this case, the entry in the `PDUs` column will be greater than that of `Packets`. Example: In the screenshot there are many more TLS and Git PDUs than there are packets.

Conversations

A network conversation is the traffic between two specific endpoints. For example, an IP conversation is all the traffic between two IP addresses. The description of the known endpoint types can be found in Endpoints.

The "Conversations" Window

The conversations window is similar to the endpoint Window. See The "Endpoints" Window for a description of their common features. Along with addresses, packet counters, and byte counters the conversation window adds four columns: the start time of the conversation ("Rel Start") or ("Abs Start"), the duration of the conversation in seconds, and the average bits (not bytes) per second in each direction. A timeline graph is also drawn across the "Rel Start" / "Abs Start" and "Duration" columns. Additionally

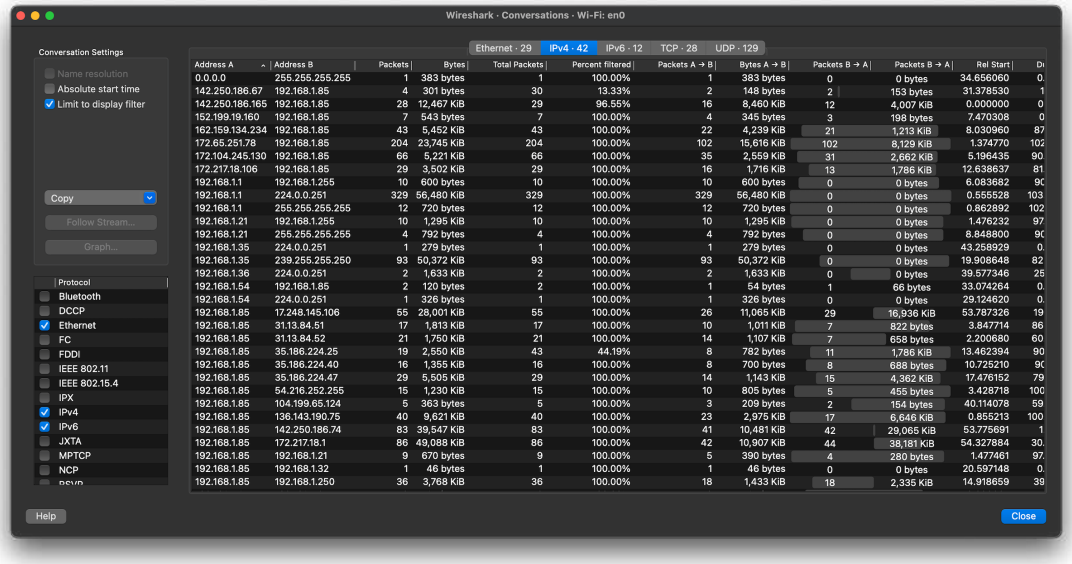


Figure 79. The "Conversations" window

Each row in the list shows the statistical values for exactly one conversation.

Name resolution will be done if selected in the window and if it is active for the specific protocol layer (MAC layer for the selected Ethernet endpoints page). *Limit to display filter* will only show conversations matching the current display filter. *Absolute start time* switches the start time column between relative ("Rel Start") and absolute ("Abs Start") times. Relative start times match the "Seconds Since First Captured Packet" time display format in the packet list and absolute start times match the "Time of Day" display format.

If a display filter had been applied before the dialog is opened, *Limit to display filter* will be set automatically. Additionally, after a display filter had been applied, two columns ("Total Packets") and ("Percent Filtered") show the number of unfiltered total packets and the percentage of packets in this filter display.

The **[Copy]** button will copy the list values to the clipboard in CSV (Comma Separated Values), YAML format or JSON format. The numbers are generally exported without special formatting, but this can be enabled if needed.

The **[Follow Stream...]** button will show the stream contents as described in The "Follow TCP Stream" dialog box dialog. The **[Graph...]** button will show a graph as described in The "I/O Graphs" Window.

[Conversation Types] lets you choose which traffic type tabs are shown. See Endpoints for a list of endpoint types. The enabled types are saved in your profile settings.

TIP	This window will be updated frequently so it will be useful even if you open it before (or while) you are doing a live capture.

Endpoints

A network endpoint is the logical endpoint of separate protocol traffic of a specific protocol layer. The endpoint statistics of Wireshark will take the following endpoints into account:

> **TIP** If you are looking for a feature other network tools call a *hostlist*, here is the right place to look. The list of Ethernet or IP endpoints is usually what you're looking for.

Endpoint and Conversation types

Bluetooth

A MAC-48 address similar to Ethernet.

Ethernet

Identical to the Ethernet device's MAC-48 identifier.

Fibre Channel

A MAC-48 address similar to Ethernet.

IEEE 802.11

A MAC-48 address similar to Ethernet.

FDDI

Identical to the FDDI MAC-48 address.

IPv4

Identical to the 32-bit IPv4 address.

IPv6

Identical to the 128-bit IPv6 address.

IPX

A concatenation of a 32-bit network number and 48-bit node address, by default the Ethernet interface's MAC-48 address.

JXTA

A 160-bit SHA-1 URN.

NCP

Similar to IPX.

RSVP

A combination of various RSVP session attributes and IPv4 addresses.

SCTP

A combination of the host IP addresses (plural) and the SCTP port used. So different SCTP ports on the same IP address are different SCTP endpoints, but the same SCTP port on different IP addresses of the same host are still the same endpoint.

TCP

A combination of the IP address and the TCP port used. Different TCP ports on the same IP address are different TCP endpoints.

Token Ring

Identical to the Token Ring MAC-48 address.

UDP

A combination of the IP address and the UDP port used, so different UDP ports on the same IP address are different UDP endpoints.

USB

Identical to the 7-bit USB address.

NOTE	*Broadcast and multicast endpoints* Broadcast and multicast traffic will be shown separately as additional endpoints. Of course, as these aren't physical endpoints the real traffic will be received by some or all of the listed unicast endpoints.

The "Endpoints" Window

This window shows statistics about the endpoints captured.

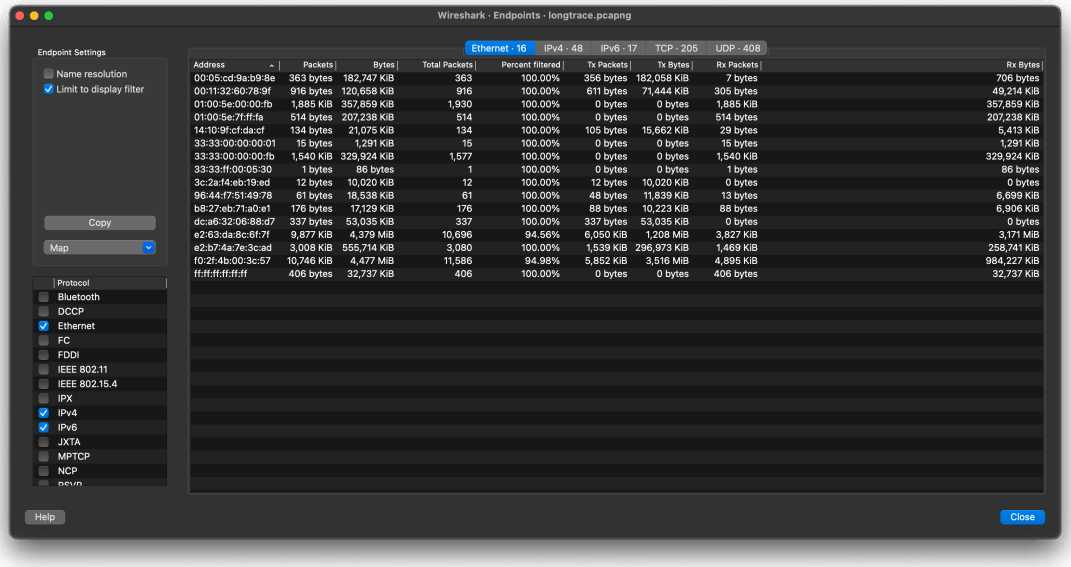


Figure 80. The "Endpoints" window

For each supported protocol, a tab is shown in this window. Each tab label shows the number of endpoints captured (e.g., the tab label “Ethernet · 4” tells you that four ethernet endpoints have been captured). If no endpoints of a specific protocol were captured, the tab label will be greyed out (although the related page can still be selected).

Each row in the list shows the statistical values for exactly one endpoint.

Name resolution will be done if selected in the window and if it is active for the specific protocol layer (MAC layer for the selected Ethernet endpoints page). *Limit to display filter* will only show conversations matching the current display filter. Note that in this example we have MaxMind DB configured which gives us extra geographic columns. See MaxMind Database Paths for more information.

If a display filter had been applied before the dialog is opened, *Limit to display filter* will be set automatically. Additionally, after a display filter had been applied, two columns ("Total Packets") and ("Percent Filtered") show the number of unfiltered total packets and the percentage of packets in this filter display.

The **[Copy]** button will copy the list values to the clipboard in CSV (Comma Separated Values), YAML format or JSON format. The numbers are generally exported without special formatting, but this can be enabled if needed. The **[Map]** button will show the endpoints mapped in your web browser.

[Endpoint Types] lets you choose which traffic type tabs are shown. See Endpoints above for a list of endpoint types. The enabled types are saved in your profile settings.

TIP	This window will be updated frequently, so it will be useful even if you open it before (or while) you are doing a live capture.

Packet Lengths

Shows the distribution of packet lengths and related information.

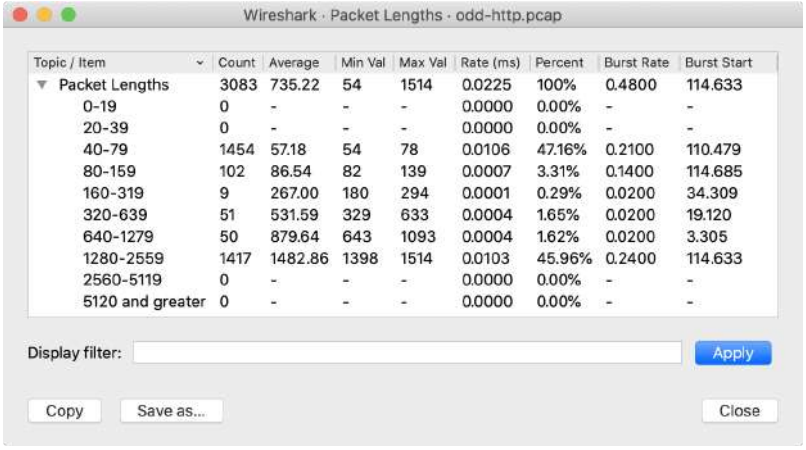


Figure 81. The “Packet Lengths” window

Information is broken down by packet length ranges as shown above.

Packet Lengths

The range of packet lengths.

Ranges can be configured in the “Statistics → Stats Tree” section of the Preferences Dialog.

Count

The number of packets that fall into this range.

Average

The arithmetic mean of the packet lengths in this range.

Min Val, Max Val

The minimum and maximum lengths in this range.

Rate (ms)

The average packets per millisecond for the packets in this range.

Percent

The percentage of packets in this range, by count.

Burst Rate

Packet bursts are detected by counting the number of packets in a given time interval and comparing that count to the intervals across a window of time. Statistics for the interval with the maximum number of packets are shown. By default, bursts are detected across 5 millisecond intervals and intervals are compared across 100 millisecond windows.

These calculations can be adjusted in the “Statistics” section of the Preferences Dialog.

Burst Start

The start time, in seconds from the beginning of the capture, for the interval with the maximum number of packets.

You can show statistics for a portion of the capture by entering a display filter into the *Display filter* entry and pressing **[Apply]**.

[Copy] copies the statistics to the clipboard. **[Save as…]** lets you save the data as text, CSV, YAML, or XML.

The “I/O Graphs” Window

Lets you plot packet and protocol data in a variety of ways.

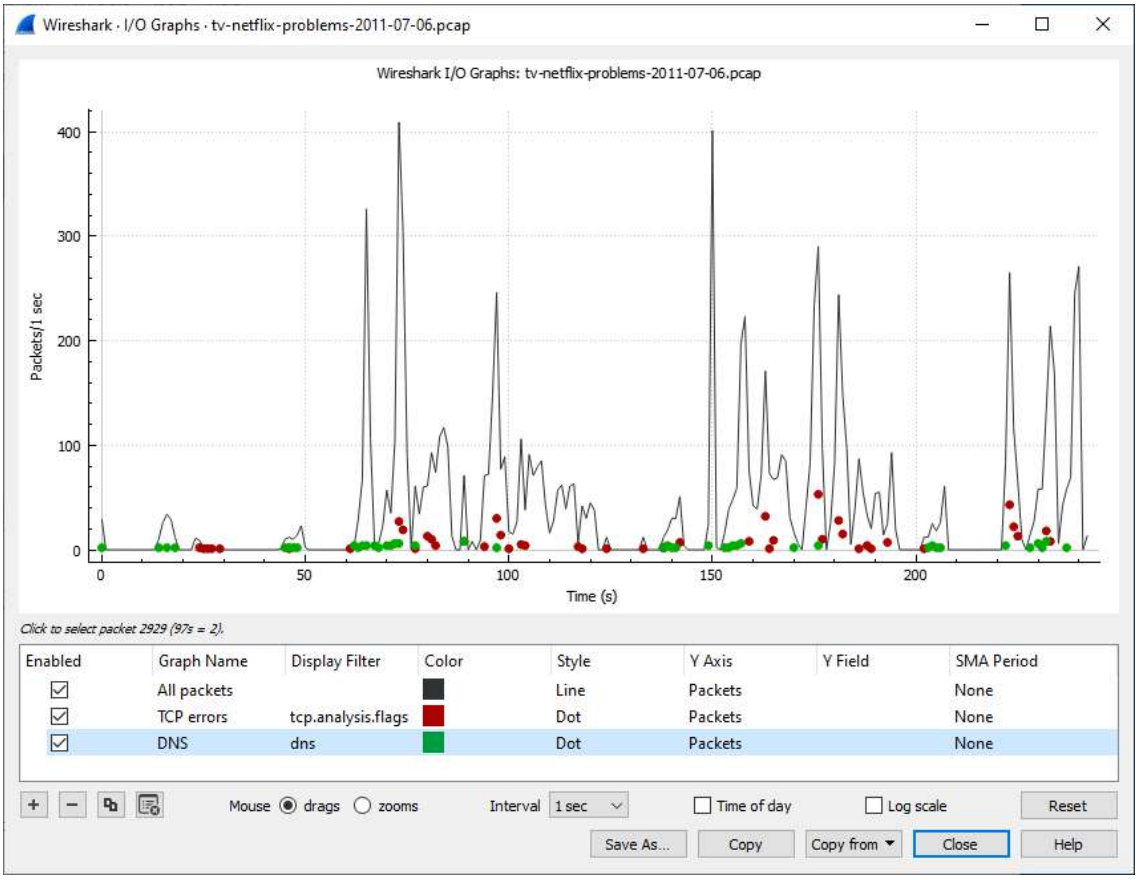


Figure 82. The "I/O Graphs" window

As shown above, this window contains a chart drawing area along with a customizable list of graphs. Graphs are saved in your current profile. They are divided into time intervals, which can be set as described below. Hovering over the graph shows the last packet in each interval except as noted below. Clicking on the graph takes you to the associated packet in the packet list. Individual graphs can be configured using the following options:

Enabled

Draw or don't draw this graph.

Graph Name

The name of this graph.

Display Filter

Limits the graph to packets that match this filter.

Color

The color to use for plotting the graph's lines, bars, or points.

Style

How to visually represent the graph's data, e.g., by drawing a line, bar, circle, plus, etc.

Y Axis

The value to use for the graph's Y axis. Can be one of:

Packets, Bytes, or Bits

The total number of packets, packet bytes, or packet bits that match the graph's display filter per interval. Zero values are omitted in some cases.

SUM(Y Field)

The sum of the values of the field specified in "Y Field" per interval.

COUNT FRAMES(Y Field)

The number of frames that contain the field specified in "Y Field" per interval. Unlike the plain "Packets" graph, this always displays zero values.

COUNT FIELDS(Y Field)

The number of instances of the field specified in "Y Field" per interval. Some fields, such as *dns.resp.name*, can show up multiple times in a packet.

MAX(Y Field), MIN(Y Field), AVG(Y Field)

The maximum, minimum, and arithmetic mean values of the specified "Y Field" per interval. For MAX and MIN values, hovering and clicking the graph will show and take you to the packet with the MAX or MIN value in the interval instead of the most recent packet.

LOAD(Y Field)

If the "Y Field" is a relative time value, this is the sum of the "Y Field" values divided by the interval time. This can be useful for tracking response times.

Y Field

The display filter field from which to extract values for the Y axis calculations listed above.

SMA Period

Show an average of values over a specified period of intervals.

The chart as a whole can be configured using the controls under the graph list:

[+]

Add a new graph.

[-]

Add a new graph.

[Copy]

Copy the selected graph.

[Clear]

Remove all graphs.

Mouse drags / zooms

When using the mouse inside the graph area, either drag the graph contents or select a zoom area.

Interval

Set the interval period for the graph.

Time of day

Switch between showing the absolute time of day or the time relative from the start of capture in the X axis.

Log scale

Switch between a logarithmic or linear Y axis.

The main dialog buttons along the bottom let you do the following:

The **[Help]** button will take you to this section of the User's Guide.

The **[Copy]** button will copy values from selected graphs to the clipboard in CSV (Comma Separated Values) format.

[Copy from] will let you copy graphs from another profile.

[Close] will close this dialog.

[Save As…] will save the currently displayed graph as an image or CSV data.

TIP	You can see a list of useful keyboard shortcuts by right-clicking on the graph.

Missing Values Are Zero

Wireshark's I/O Graph window doesn't distinguish between missing and zero values. For scatter plots it is assumed that zero values indicate missing data, and those values are omitted. Zero values are shown in line graphs, and bar charts.

Service Response Time

The service response time is the time between a request and the corresponding response. This information is available for many protocols, including the following:

- AFP
- CAMEL

- DCE-RPC
- Diameter
- Fibre Channel
- GTP
- H.225 RAS
- LDAP
- MEGACO
- MGCP
- NCP
- ONC-RPC
- RADIUS
- SCSI
- SMB
- SMB2
- SNMP

As an example, the SMB2 service response time is described below in more detail. The other Service Response Time windows will show statistics specific to their respective protocols, but will offer the same menu options.

The “SMB2 Service Response Time Statistics” Window

This window shows the number of transactions for each SMB2 opcode present in the capture file along with various response time statistics. Right-clicking on a row will let you apply or prepare filters for, search for, or colorize a specific opcode. You can also copy all of the response time information or save it in a variety of formats.

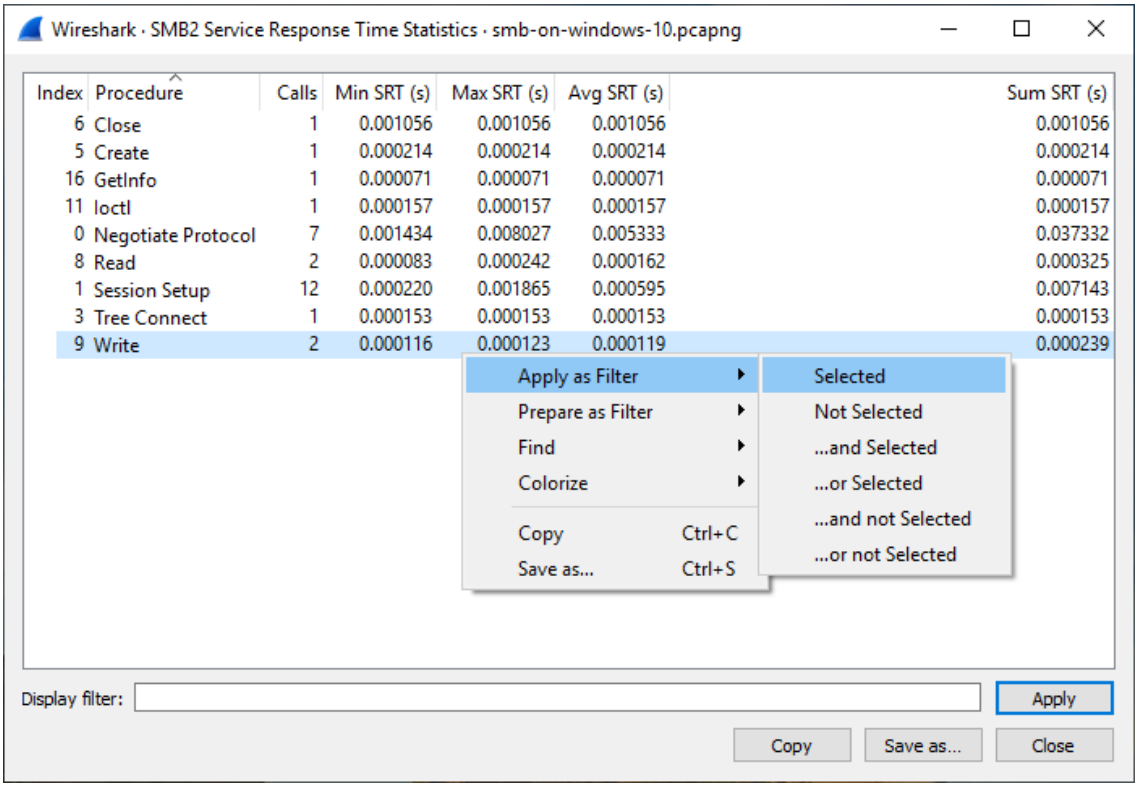


Figure 83. The "SMB2 Service Response Time Statistics" window

You can optionally apply a display filter in order to limit the statistics to a specific set of packets.

The main dialog buttons along the bottom let you do the following:

The **[Copy]** button will copy the response time information as text.

[Save As...] will save the response time information in various formats.

[Close] will close this dialog.

DHCP (BOOTP) Statistics

The Dynamic Host Configuration Protocol (DHCP) is an option of the Bootstrap Protocol (BOOTP). It dynamically assigns IP addresses and other parameters to a DHCP client. The DHCP (BOOTP) Statistics window displays a table over the number of occurrences of a DHCP message type. The user can filter, copy or save the data into a file.

NetPerfMeter Statistics

The NetPerfMeter Protocol (NPMP) is the control and data transfer protocol of NetPerfMeter, the transport protocol performance testing tool. It transmits data streams over TCP, SCTP, UDP and DCCP with given parameters, such as frame rate, frame size, saturated flows, etc.

With these statistics you can:

- Observed number of messages and bytes per message type.
- The share of messages and bytes for each message type.
- See the first and last occurrence of each message type.
- See the interval between first and last occurrence of each message type (if there are at least 2 messages of the corresponding type).
- See the message and byte rate within the interval for each message type (if there are at least 2 messages of the corresponding type).

See NetPerfMeter – A TCP/MPTCP/UDP/SCTP/DCCP Network Performance Meter Tool and Section 6.3 of Evaluation and Optimisation of Multi-Path Transport using the Stream Control Transmission Protocol for more details about NetPerfMeter and the NetPerfMeter Protocol.

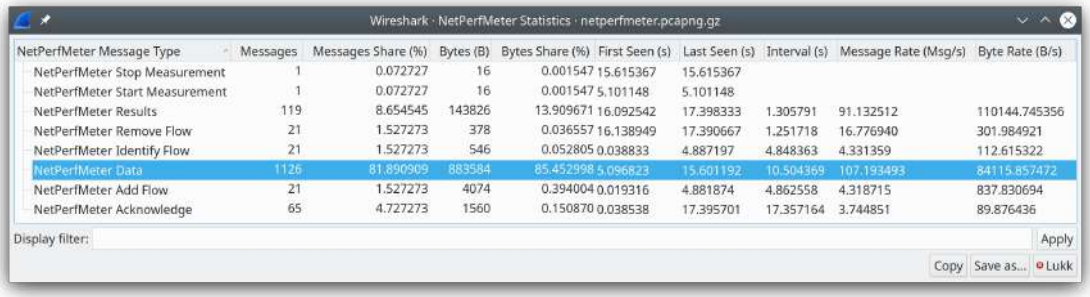


Figure 84. NetPerfMeter Statistics window

ONC-RPC Programs

Open Network Computing (ONC) Remote Procedure Call (RPC) uses TCP or UDP protocols to map a program number to a specific port on a remote machine and call a required service at that port. The ONC-RPC Programs window shows the description for captured program calls, such as program name, its number, version, and other data.

29West

The 29West technology now refers to Ultra-Low Latency Messaging (ULLM) technology. It allows sending and receiving a high number of messages per second with microsecond delivery times for zero-latency data delivery.

The **Statistics** › **29West** shows:

The `Topics` submenu shows counters for:	• Advertisement by Topic • Advertisement by Source • Advertisement by Transport • Queries by Topic • Queries by Receiver • Wildcard Queries by Pattern • Wildcard Queries by Receiver
The `Queues` submenu shows counters for:	• Advertisement by Queue • Advertisement by Source • Queries by Queue • Queries by Receiver
The `UIM` submenu shows `Streams`:	Each stream is provided by Endpoints, Messages, Bytes, and the First and Last Frame statistics.
The `LBT-RM` submenu	The LBT-RM Transport Statistics window shows the Sources and Receivers sequence numbers for transport and other data.
The `LBT-RU` submenu	The LBT-Ru Transport Statistics window shows the Sources and Receivers sequence numbers for transport and other data.

ANCP

The Access Node Control Protocol (ANCP) is an TCP based protocol, which operates between an Access Node and Network Access Server. The Wireshark ANCP dissector supports the listed below messages:

- Adjacency Message
- Topology Discovery Extensions, such as Port-Up and Port-Down Messages
- Operation And Maintenance (OAM) Extension, such as Port Management Message.

The ANCP window shows the related statistical data. The user can filter, copy or save the data into a file.

BACnet

Building Automation and Control Networks (BACnet) is a communication protocol which provides control for various building automated facilities, such as light control, fire alarm control, and

others. Wireshark provides the BACnet statistics which is a packet counter. You can sort packets by instance ID, IP address, object type or service.

Collectd

Collectd is a system statistics collection daemon. It collects various statistics from your system and converts it for the network use. The Collectd statistics window shows counts for values, which split into type, plugin, and host as well as total packets counter. You can filter, copy or save the data to a file.

DNS

The Domain Name System (DNS) associates different information, such as IP addresses, with domain names. DNS returns different codes, request-response and counters for various aggregations. The DNS statistics window enlists a total count of DNS messages, which are divided into groups by request types (opcodes), response code (rcode), query type, and others.

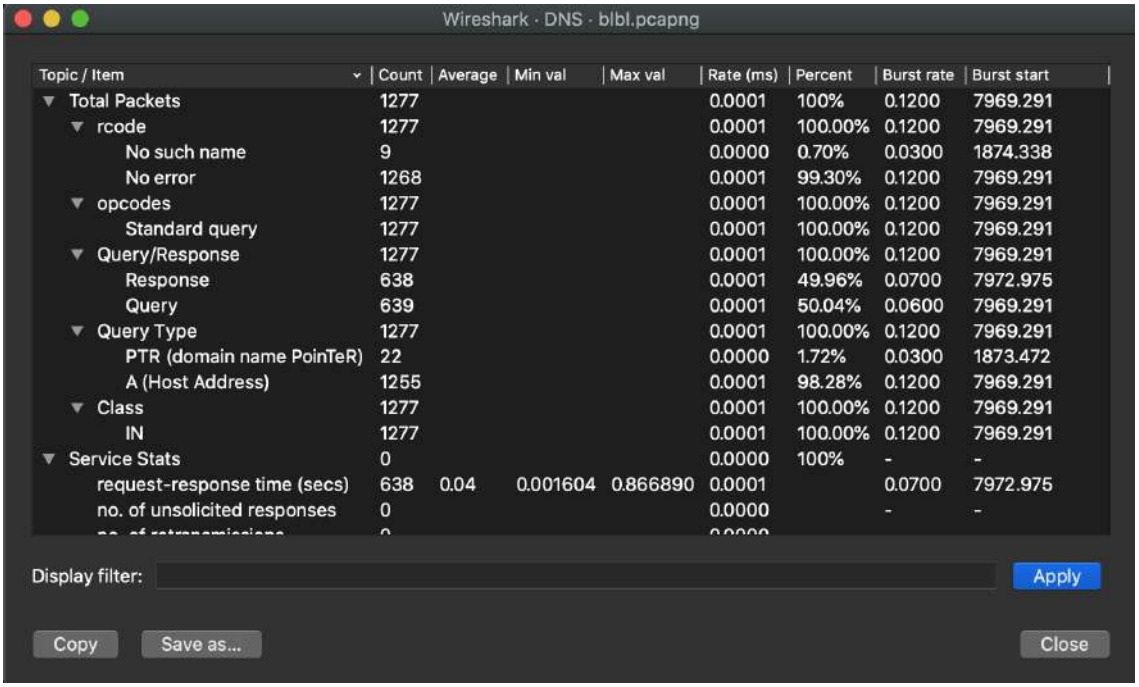


Figure 85. DNS statistics window

You might find these statistics useful for quickly examining the health of a DNS service or other investigations. See the few possible scenarios below:

- The DNS server might have issues if you see that DNS queries have a long request-response time or, if there are too many unanswered queries.
- DNS requests with abnormally large requests and responses might be indicative of DNS tunneling or command and control traffic.
- The order of magnitude more DNS responses than requests and the responses are very large might indicate that the target is being attacked with a DNS-based DDoS.

You can filter, copy or save the data into a file.

Flow Graph

The Flow Graph window shows connections between hosts. It displays the packet time, direction, ports and comments for each captured connection. You can filter all connections by ICMP Flows, ICMPv6 Flows, UIM Flows and TCP Flows. Flow Graph window is used for showing multiple different topics. Based on it, it offers different controls.

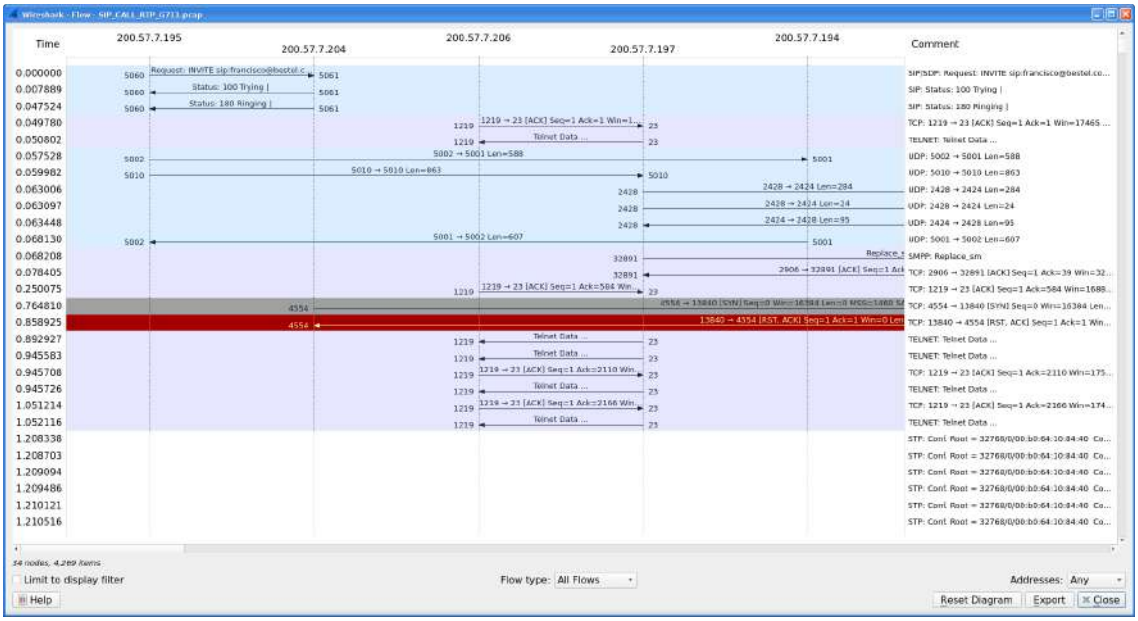


Figure 86. Flow Graph window

Each vertical line represents the specific host, which you can see in the top of the window.

The numbers in each row at the very left of the window represent the time packet. You can change the time format in the **View › Time Display Format**. If you change the time format, you must relaunch the Flow Graph window to observe the time in a new format.

The numbers at the both ends of each arrow between hosts represent the port numbers.

Left-click a row to select a corresponding packet in the packet list.

Right-click on the graph for additional options, such as selecting the previous, current, or next packet in the packet list. This menu also contains shortcuts for moving the diagram.

Available controls:

- **[Limit to display filter]** filters calls just to ones matching display filter. When display filter is active before window is opened, checkbox is checked.
- **[Flow type]** allows limit type of protocol flows should be based on.
- **[Addresses]** allows switch shown addresses in diagram.

- **[Reset Diagram]** resets view position and zoom to default state.
- **[Export]** allows export diagram as image in multiple different formats (PDF, PNG, BMP, JPEG and ASCII (diagram is stored with ASCII characters only)).

Figure 87. Flow Graph window showing VoIP call sequences

Additional shortcuts available for VoIP calls:

- On selected RTP stream
  - `S` - Selects the stream in RTP Streams window (if not opened, it opens it and put it on background).
  - `D` - Deselects the stream in RTP Streams window (if not opened, it opens it and put it on background).

Additional controls available for VoIP calls:

- **[Reset Diagram]** resets view position and zoom to default state.
- **[Play Streams]** sends selected RTP stream to playlist of RTP Player window.
- **[Export]** allows to export diagram as image in multiple different formats (PDF, PNG, BMP, JPEG and ASCII (diagram is stored with ASCII characters only)).

HART-IP

Highway Addressable Remote Transducer over IP (HART-IP) is an application layer protocol. It sends and receives digital information between smart devices and control or monitoring systems. The HART-IP statistics window shows the counter for response, request, publish and error packets. You can filter, copy or save the data to a file.

HPFEEDS

Hpfeeds protocol provides a lightweight authenticated publishing and subscription. It supports arbitrary binary payloads which can be separated into different channels. HPFEEDS statistics window shows a counter for payload size per channel and opcodes. You can filter, copy or save the data to a file.

HTTP Statistics

HTTP Packet Counter

Statistics for HTTP request types and response codes.

HTTP Requests

HTTP statistics based on the host and URI.

HTTP Load Distribution

HTTP request and response statistics based on the server address and host.

HTTP Request Sequences

HTTP Request Sequences uses HTTP's Referer and Location headers to sequence a capture's HTTP requests as a tree. This enables analysts to see how one HTTP request leads to the next.

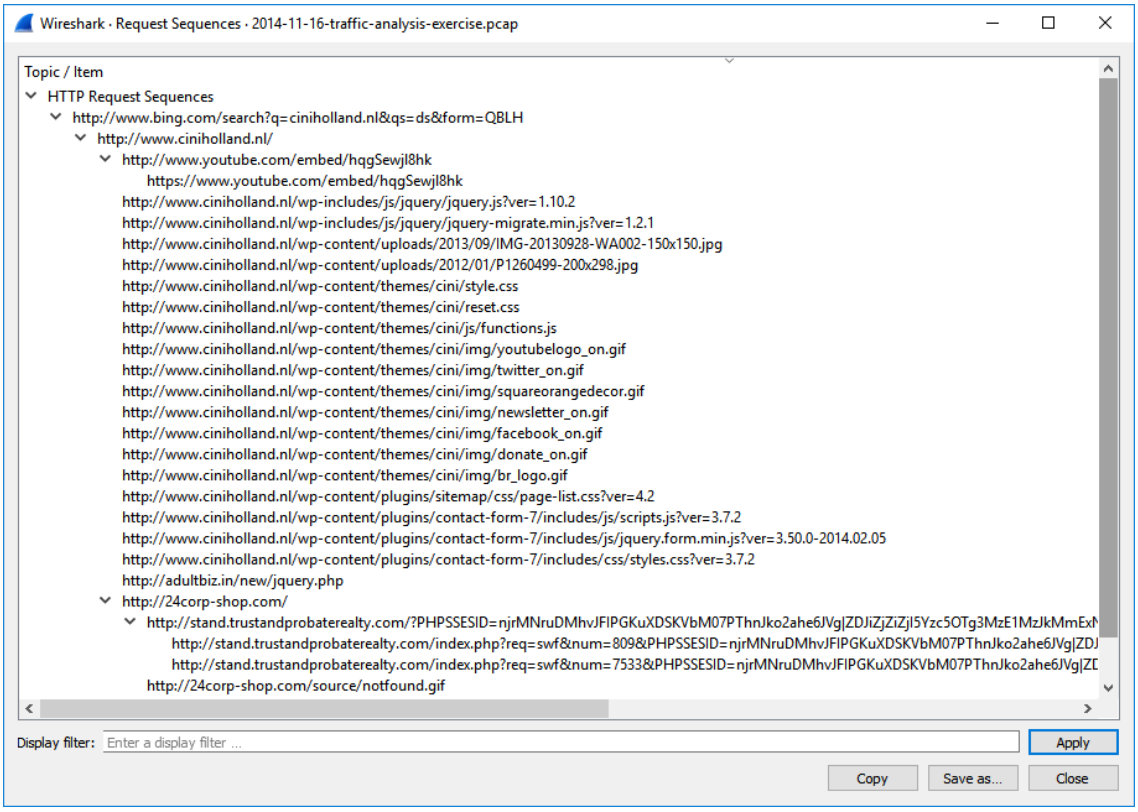


Figure 88. The "HTTP Request Sequences" window

HTTP2

Hypertext Transfer Protocol version 2 (HTTP/2) allows multiplexing various HTTP requests and responses over a single connection. It uses a binary encoding which is consisting of frames. The HTTP/2 statistics window shows the total number of HTTP/2 frames and also provides a breakdown per frame types, such as `HEADERS`, `DATA`, and others.

As HTTP/2 traffic is typically encrypted with TLS, you must configure decryption to observe HTTP/2 traffic. For more details, see the TLS wiki page.

Sametime

Sametime is a protocol for the IBM Sametime software. The Sametime statistics window shows the counter for message type, send type, and user status.

TCP Stream Graphs

Show different visual representations of the TCP streams in a capture.

Time Sequence (Stevens)

This is a simple graph of the TCP sequence number over time, similar to the ones used in Richard Stevens' "TCP/IP Illustrated" series of books.

Time Sequence (tcptrace)

Shows TCP metrics similar to the tcptrace utility, including forward segments, acknowledgements, selective acknowledgements, reverse window sizes, and zero windows.

Throughput

Average throughput and goodput.

Round Trip Time

Round trip time vs time or sequence number. RTT is based on the acknowledgement timestamp corresponding to a particular segment.

Window Scaling

Window size and outstanding bytes.

UDP Multicast Streams

The UDP Multicast Streams window shows statistics for all UDP multicast streams. It includes source addresses and ports, destination addresses and ports, packets counter and other data. You can specify the burst interval, the alarm limits and output speeds. To apply new settings, press **[Enter]**.

With these statistics you can:

- Measure the burst size for a video stream. This uses the sliding window algorithm.
- Measure of the output buffer size limit, that no packet drop will occur. This uses the Leaky bucket algorithm.
- Detect the packet loss inside the MPEG2 video stream.

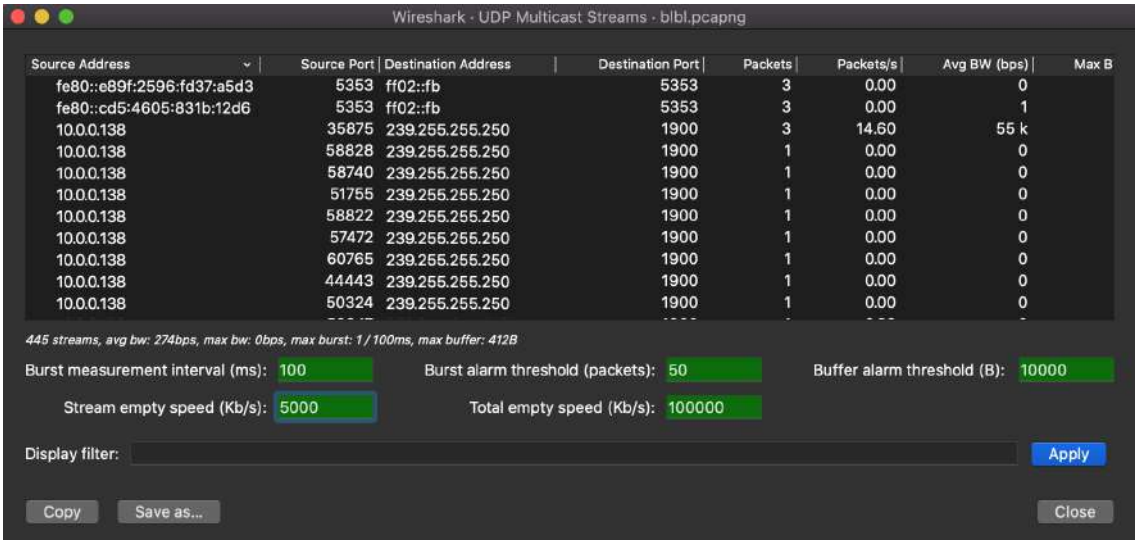


Figure 89. UDP Multicast Streams window

Reliable Server Pooling (RSerPool)

The Reliable Server Pooling (RSerPool) windows show statistics for the different protocols of Reliable Server Pooling (RSerPool):

- Aggregate Server Access Protocol (ASAP)
- Endpoint Handlespace Redundancy Protocol (ENRP)

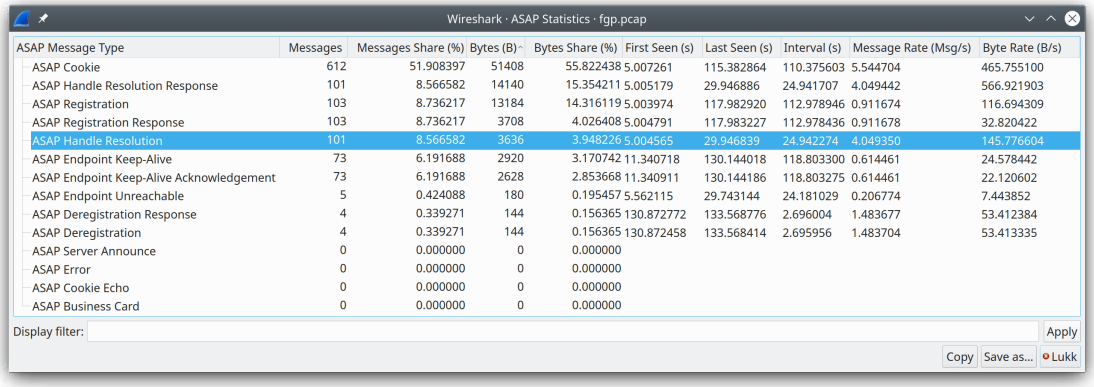
Furthermore, statistics for application protocols provided by RSPLIB are provided as well:

- Component Status Protocol (CSP)
- CalcApp Protocol
- Fractal Generator Protocol
- Ping Pong Protocol
- Scripting Service Protocol (SSP)

With these statistics you can:

- Observed number of messages and bytes per message type.
- The share of messages and bytes for each message type.
- See the first and last occurrence of each message type.
- See the interval between first and last occurrence of each message type (if there are at least 2 messages of the corresponding type).
- See the message and byte rate within the interval for each message type (if there are at least 2 messages of the corresponding type).

See Thomas Dreibholz's Reliable Server Pooling (RSerPool) Page and Chapter 3 of Reliable Server Pooling – Evaluation, Optimization and Extension of a Novel IETF Architecture for more details about RSerPool and its protocols.



ASAP Message Type	Messages	Messages Share (%)	Bytes (B)	Bytes Share (%)	First Seen (s)	Last Seen (s)	Interval (s)	Message Rate (Msg/s)	Byte Rate (B/s)
ASAP Cookie	612	51.908397	51408	55.822438	5.007261	115.382864	110.375603	5.544704	465.755100
ASAP Handle Resolution Response	101	8.566582	14140	15.354211	5.005179	29.946886	24.941707	4.049442	566.921903
ASAP Registration	103	8.736217	13184	14.316119	5.003974	117.982920	112.978946	0.911674	116.694309
ASAP Registration Response	103	8.736217	3708	4.026408	5.004791	117.983227	112.978436	0.911678	32.820422
ASAP Handle Resolution	101	8.566582	3636	3.948226	5.004565	29.946839	24.942274	4.049350	145.776604
ASAP Endpoint Keep-Alive	73	6.191688	2920	3.170742	11.340718	130.144018	118.803300	0.614461	24.578442
ASAP Endpoint Keep-Alive Acknowledgement	73	6.191688	2628	2.853668	11.340911	130.144186	118.803275	0.614461	22.120602
ASAP Endpoint Unreachable	5	0.424088	180	0.195457	5.562115	29.743144	24.181029	0.206774	7.443852
ASAP Deregistration Response	4	0.339271	144	0.156365	130.872772	133.568776	2.696004	1.483677	53.412384
ASAP Deregistration	4	0.339271	144	0.156365	130.872458	133.568414	2.695956	1.483704	53.413335
ASAP Server Announce	0	0.000000	0	0.000000					
ASAP Error	0	0.000000	0	0.000000					
ASAP Cookie Echo	0	0.000000	0	0.000000					
ASAP Business Card	0	0.000000	0	0.000000					

Figure 90. ASAP Statistics window

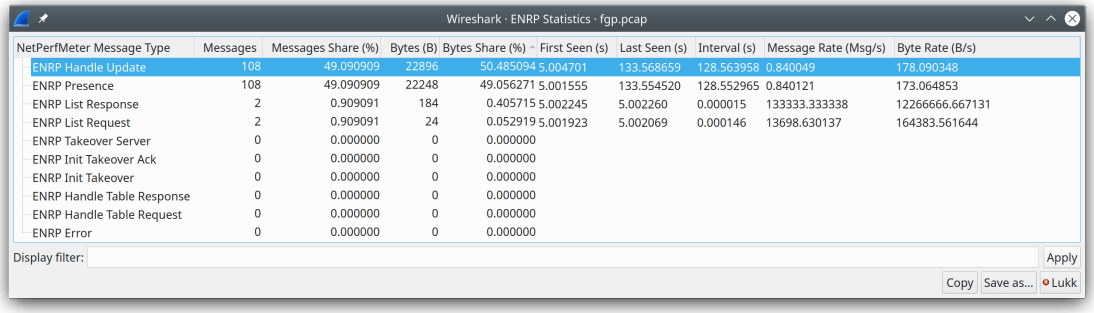


Figure 91. ENRP Statistics window

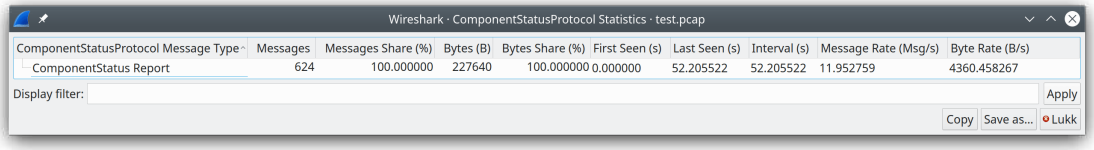


Figure 92. Component Status Protocol Statistics window

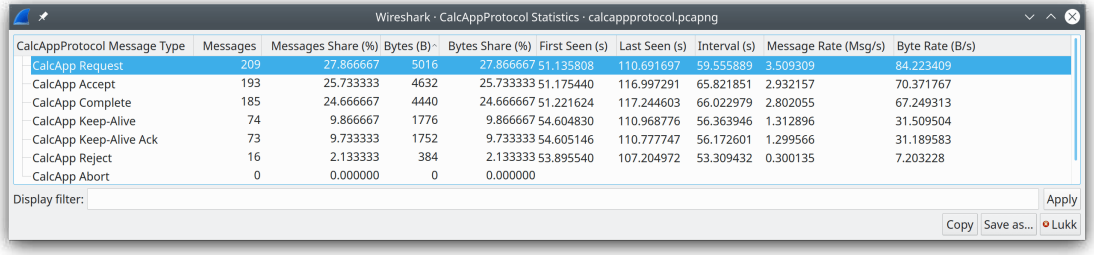


Figure 93. CalcApp Protocol Statistics window

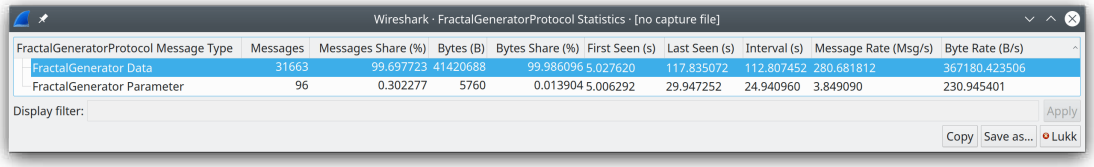


Figure 94. Fractal Generator Protocol Statistics window

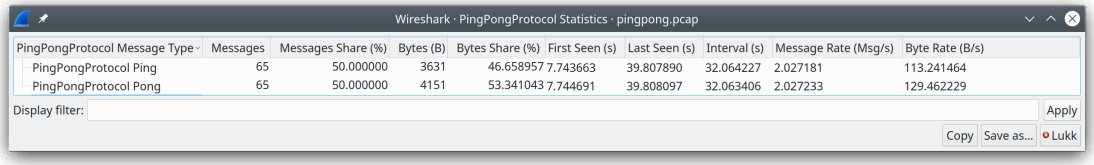


Figure 95. Ping Pong Protocol Statistics window

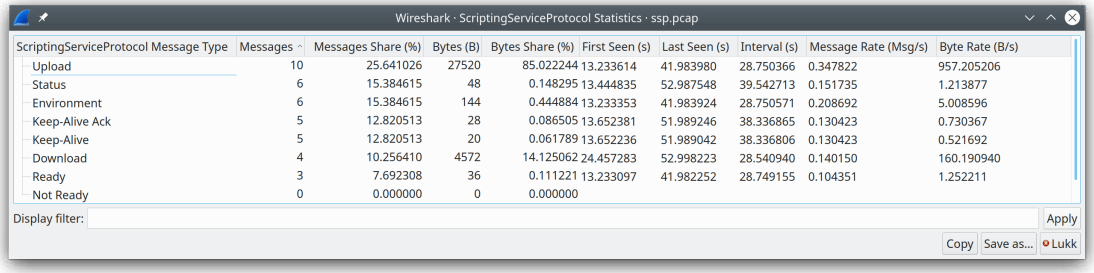


Figure 96. Scripting Service Protocol Statistics window

F5

In F5 Networks, **TMM** stands for Traffic Management Microkernel. It processes all load-balanced traffic on the BIG-IP system.

The F5 statistics menu shows packet and byte counts for both `Virtual Server Distribution` and `tmm Distribution` submenus.

Each `Virtual Server Distribution` window contains the statistics for the following data:

- A line for each named virtual server name.
- A line for traffic with a flow ID and no virtual server name.
- A line for traffic without a flow ID.

Each `tmm Distribution` window contains the statistics for the following data:

- A line for each tmm, which contains:
  - A line for each ingress and egress (should add to tmm total), which contains:
    - Traffic with a virtual server name.
    - Traffic with a flow ID and no virtual server name.
    - Traffic without a flow ID.

IPv4 Statistics

Internet Protocol version 4 (IPv4) is a core protocol for the internet layer. It uses 32-bit addresses and allows packets routing from one source host to the next one.

The **Statistics › IPv4** menu provides the packet counter by submenus:

- `All Addresses`. Divides data by IP address.
- `Destination and Ports`. Divides data by IP address, and further by IP protocol type, such as TCP, UDP, and others. It also shows port number.
- `IP Protocol Types`. Divides data by IP protocol type.

- `Source and Destination addresses`. Divides data by source and destination IP address.

You can see similar statistics in the **Statistics › Conversations** and **Statistics › Endpoints** menus.

IPv6 Statistics

Internet Protocol version 6 (IPv6) is a core protocol for the internet layer. It uses 128-bit addresses and routes internet traffic. Similar to IPv4 Statistics, the **Statistics › IPv6** menu shows the packet counter in each submenu.

Telephony

Introduction

Wireshark provides a wide range of telephony related network statistics which can be accessed via the **Telephony** menu.

These statistics range from specific signaling protocols, to analysis of signaling and media flows. If encoded in a compatible encoding the media flow can even be played.

The protocol specific statistics windows display detailed information of specific protocols and might be described in a later version of this document.

Some of these statistics are described at the https://gitlab.com/wireshark/wireshark/wikis/Statistics pages.

Playing VoIP Calls

The tool for playing VoIP calls is called RTP Player. It shows RTP streams and its waveforms, allows play stream and export it as audio or payload to file. Its capabilities depend on supported codecs.

Supported codecs

RTP Player is able to play any codec supported by an installed plugin. The codecs supported by RTP Player depend on the version of Wireshark you're using. The official builds contain all of the plugins maintained by the Wireshark developers, but custom/distribution builds might not include some of those codecs. To check your Wireshark installation's installed codec plugins, do the following:

- Open **Help** › **About Wireshark** window
- Select the **Plugins** tab
- In the **Filter by type** menu on the top-right, select codec

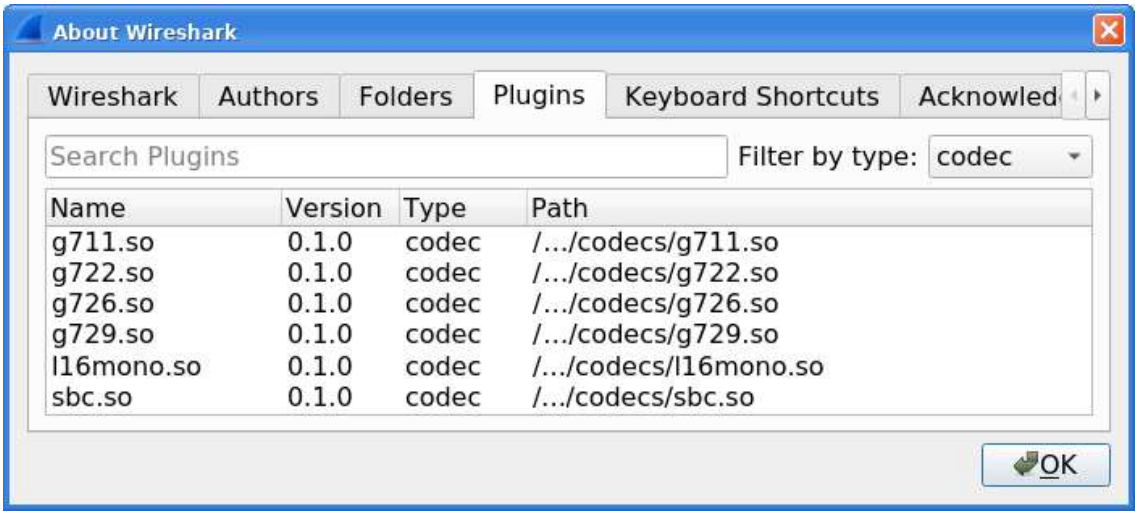


Figure 97. List of supported codecs

Work with RTP streams - Playlist

Wireshark can be used for RTP stream analysis. User can select one or more streams which can be played later. RTP Player window maintains playlist (list of RTP streams) for this purpose.

Playlist is created empty when RTP Player window is opened and destroyed when window is closed. RTP Player window can be opened on background when not needed and put to front later. During its live, playlist is maintained.

When RTP Player window is opened, playlist can be modified from other tools (Wireshark windows) in three ways:

- button **Play Streams › Set playlist** clears existing playlist and adds streams selected in the tool.
- button **Play Streams › Add to playlist** adds streams selected in the tool to playlist. Duplicated streams are not inserted again.
- button **Play Streams › Remove from playlist** removes streams selected in the tool from playlist, if they are in the playlist.

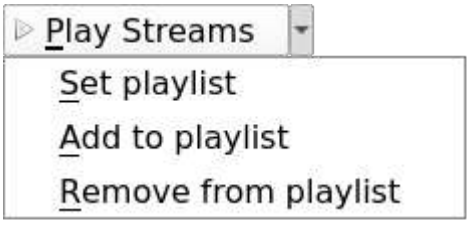


Figure 98. **[Play Streams]** *button with opened action menu*

[Play Streams] button can be clicked directly and opens RTP Player window directly with **[Set playlist]** action. All actions can be selected with the small down arrow next to the button.

When the playlist is empty, there is no difference between **[Set playlist]** and **[Add to playlist]**. When the RTP Player window is not opened, all three actions above open it.

[Remove from playlist] is useful e.g. in case user selected all RTP streams and wants to remove RTP streams from specific calls found with **VoIPCalls**.

Tools below can be used to maintain content of playlist, they contain **[Play Streams]** button. You can use one of procedures (Note: **[Add to playlist]** action is demonstrated):

- Open **Telephony › RTP › RTP Streams** window, it will show all streams in the capture. Select one or more streams and then press **[Play Streams]**. Selected streams are added to playlist.
- Select any RTP packet in packet list, open **Telephony › RTP › Stream Analysis** window. It will show analysis of selected forward stream and its reverse stream (if **[Ctrl]** is pressed during window opening). Then press **[Play Streams]**. Forward and reverse stream is added to playlist.
  - **RTP Stream Analysis** window can be opened from other tools too.
- Open **Telephony › VoIP Calls** or **Telephony › SIP Flows** window, it will show all calls. Select one or more calls and then press **[Play Streams]**. It will add all RTP streams related to selected calls to playlist.
- Open **[Flow Sequence]** window in **Telephony › VoIP Calls** or **Telephony › SIP Flows** window, it will show flow sequence of calls. Select any RTP stream and then press **[Play Streams]**. It will add selected RTP stream to playlist.

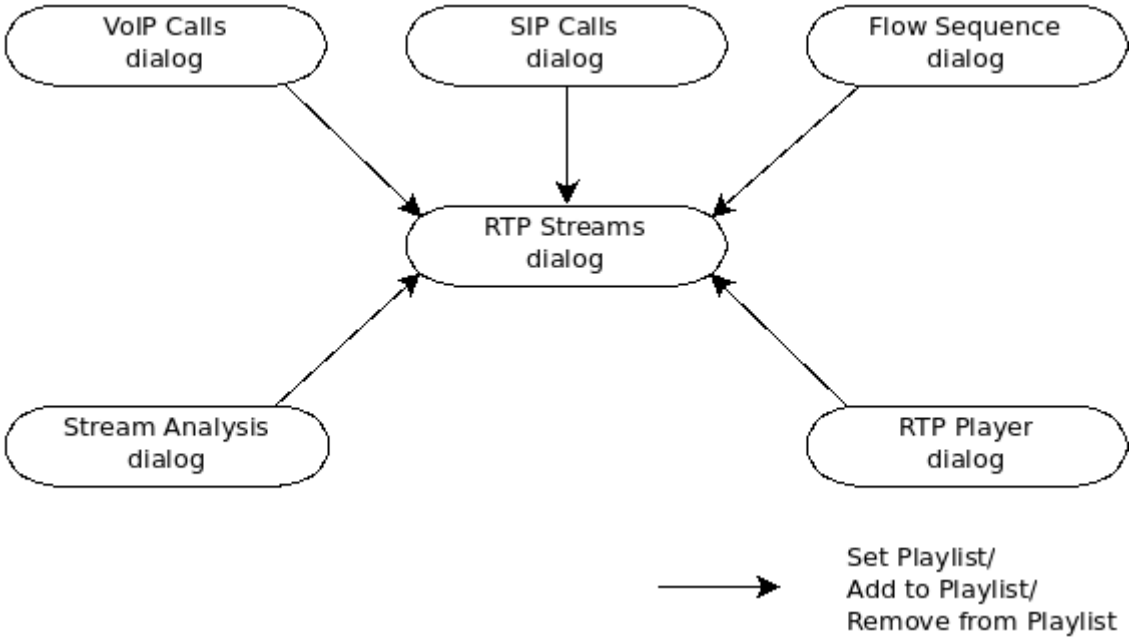


Figure 99. Tools for modifying playlist in RTP Player window

NOTE	Same approach with set/add/remove actions is used for RTP Stream Analysis window. The playlist is there handled as different tabs in the window, see RTP Stream Analysis window.

Playing audio during live capture

Decoding RTP payload and showing waveforms is time consuming task. To speedup it RTP Player

window uses copy of packet payload for all streams in the playlist. During live capture the dialog is not refreshed automatically as other Wireshark dialogs, but user must initiate it.

The copy is created or refreshed and dialog updated:

- Every time window is opened.
- Every time a new stream is added or set.
- During live capture, when **[Refresh streams]** is pressed.
- Every time live capture is finished/stopped by a user.

When capture file is opened (no live capturing), streams are read complete, no user action is required. Button **[Refresh streams]** is disabled as it is useless.

When live capture is running, streams are read only till "now" and are shown. When stream is continuous and user would like to see additional part, they must press **[Refresh stream]**. When the user ends live capture, view is refreshed and button is disabled.

NOTE	RTP Player dialog stays open even live capture is stopped and then started again. Play list stays unchanged. Therefore, **[Refresh stream]** tries to read same streams as before and shows them if they are still running. Past part of them (from previous live capture) is lost.

RTP Decoding Settings

RTP is carried usually in UDP packets with random source and destination ports. Therefore, Wireshark can only recognize RTP streams based on VoIP signaling, e.g., based on SDP messages in SIP signaling. If signaling is not captured, Wireshark shows just UDP packets. However, there are multiple settings which help Wireshark recognize RTP even when there is no related signaling.

You can use Decode As... function from **Analyze › Decode As...** menu or in mouse context menu. Here you can set that traffic on specific source or destination should be decoded as RTP. You can save settings for later use.

Use of **Decode As...** menu works fine, but is arduous for many streams.

You can enable heuristic dissector **rtp_udp** in **Analyze › Enabled Protocols...**. See Control Protocol dissection for details. Once **rtp_udp** is enabled, Wireshark tries to decode every UDP packet as RTP. If decoding is possible, packet (and entire UDP stream) is decoded as RTP.

When an RTP stream uses a well-known port, the heuristic dissector ignores it. So you might miss some RTP streams. You can enable setting for udp protocol **Preferences › Protocols › udp › Try heuristic sub-dissectors first**, see Preferences. In this case heuristics dissector tries to decode UDP packet even it uses a well-known port.

NOTE	Take into account that heuristics is just simple "test" whether packet can be read as RTP. It can be false positive and you can see decoded as RTP more UDP packets than expected. When you enable **udp › Try heuristic sub-dissectors first**, it increases possibility of false positives. If you capture all traffic in network, false positives rate can be quite high.

RTP Player must store decoded data somewhere to be able to play it. When data are decoded, there are audio samples and dictionary for fast navigation. Both types of data are stored in memory for default, but you can configure Wireshark to store it on disk. There are two settings:

- ui.rtp_player_use_disk1 - When set to FALSE (default), audio samples are kept in memory. When set to TRUE, audio samples are stored on temporary file.
- ui.rtp_player_use_disk2 - When set to FALSE (default), dictionary is kept in memory. When set to TRUE, dictionary is stored on temporary file.

When any data are configured to be stored on disk, one file is created for each stream. Therefore, there might be up to two files for one RTP stream (audio samples and dictionary). If your OS or user has OS enforced limit for count of opened files (most of Unix/Linux systems), you can see fewer streams that was added to playlist. Warnings are printed on console in this case and you will see fewer streams in the playlist than you send to it from other tools.

For common use you can use default settings - store everything in memory. When you will be out of memory, switch ui.rtp_player_use_disk1 to TRUE first - it saves much more memory than ui.rtp_player_use_disk2.

VoIP Processing Performance and Related Limits

Processing of RTP and decoding RTP voice takes resources. There are raw estimates you can use as guidelines…

RTP Streams window can show as many streams as found in the capture. Its performance is limited just by memory and CPU.

RTP Player can handle 1000+ streams, but take into account that waveforms are very small and difficult to recognize in this case.

RTP Player plays audio by OS sound system and OS is responsible for mixing audio when multiple streams are played. In many cases OS sound system has limited count of mixed streams it can play/mix. RTP Player tries to handle playback failures and show warning. If it happens, just mute some streams and start playback again.

RTP Analysis window can handle 1000+ streams, but it is difficult to use it with so many streams - it is difficult to navigate between them. It is expected that RTP Analysis window will be used for analysis of lower tens of streams.

VoIP Calls Window

The VoIP Calls window shows a list of all detected VoIP calls in the captured traffic. It finds calls by their signaling and shows related RTP streams. The current VoIP supported protocols are:

- H.323
- IAX2
- ISUP
- MGCP/MEGACO
- SIP
- SKINNY
- UNISTIM

See VOIPProtocolFamily for an overview of the used VoIP protocols.

VoIP Calls window can be opened as window showing all protocol types (**Telephony › VoIP Calls** window) or limited to SIP messages only (**Telephony › SIP Flows** window).

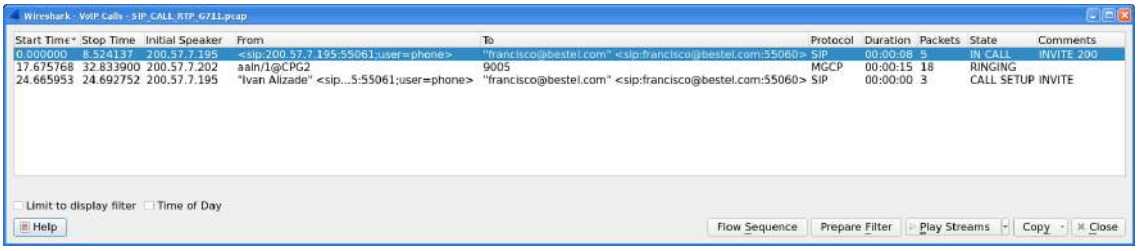


Figure 100. VoIP Calls window

User can use shortcuts:

- Selection
  - `Ctrl` + `A` - Select all streams
  - `Ctrl` + `I` - Invert selection
  - `Ctrl` + `Shift` + `A` - Select none
  - Note: Common `Mouse click`, `Shift` + `Mouse click` and `Ctrl` + `Mouse click` works too
- On selected call/calls
  - `S` - Selects stream/streams related to call in RTP Streams window (if not opened, it opens it and put it on background).
  - `D` - Deselects stream/streams related to call in RTP Streams window (if not opened, it opens it and put it on background).

Available controls are:

- **[Limit to display filter]** filters calls just to ones matching display filter. When display filter is active before window is opened, checkbox is checked.
- **[Time of Day]** switches format of shown time between relative to start of capture or absolute time of received packets.
- **[Flow Sequence]** opens Flow Sequence window and shows selected calls in it.
- **[Prepare Filter]** generates display filter matching to selected calls (signaling and RTP streams) and apply it.
- **[Play Streams]** opens RTP Player window. Actions **[Set]**, **[Add]** and **[Remove]** are available.
- **[Copy]** copies information from table to clipboard in CSV or YAML.

ANSI

This menu shows groups of statistic data for mobile communication protocols according to ETSI GSM standards.

A-I/F BSMAP Statistics Window

The A-Interface Base Station Management Application Part (BSMAP) Statistics window shows the messages list and the number of the captured messages. There is a possibility to filter the messages, copy or save the date into a file.

A-I/F DTAP Statistics Window

The A-Interface Direct Transfer Application Part (DTAP) Statistics widow shows the messages list and the number of the captured messages. There is a possibility to filter the messages, copy or save the date into a file.

GSM Windows

The Global System for Mobile Communications (GSM) is a standard for mobile networks. This menu shows a group of statistic data for mobile communication protocols according to ETSI GSM standard.

IAX2 Stream Analysis Window

The “IAX2 Stream Analysis” window shows statistics for the forward and reverse streams of a selected IAX2 call along with a graph.

ISUP Messages Window

Integrated Service User Part (ISUP) protocol provides voice and non-voice signaling for telephone communications. ISUP Messages menu opens the window which shows the related statistics. The

user can filter, copy or save the data into a file.

LTE

LTE MAC Traffic Statistics Window

Statistics of the captured LTE MAC traffic. This window will summarize the LTE MAC traffic found in the capture.

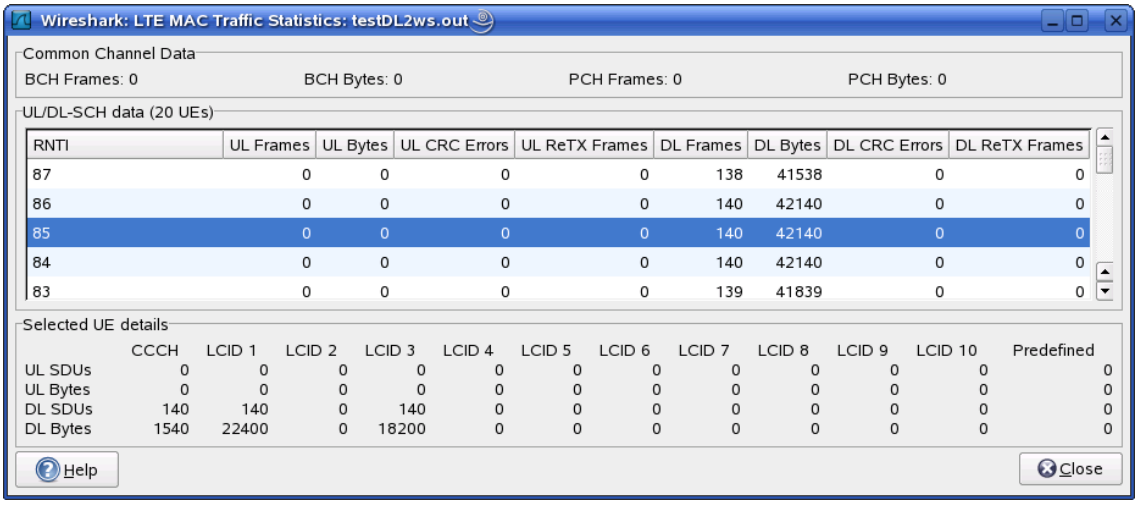


Figure 101. The "LTE MAC Traffic Statistics" window

The top pane shows statistics for common channels. Each row in the middle pane shows statistical highlights for exactly one UE/C-RNTI. In the lower pane, you can see the for the currently selected UE/C-RNTI the traffic broken down by individual channel.

LTE RLC Graph Window

The LTE RLC Graph menu launches a graph which shows LTE Radio Link Control protocol sequence numbers changing over time along with acknowledgements which are received in the opposite direction.

NOTE That graph shows data of a single bearer and direction. The user can also launch it from the `RLC Statistics` window.

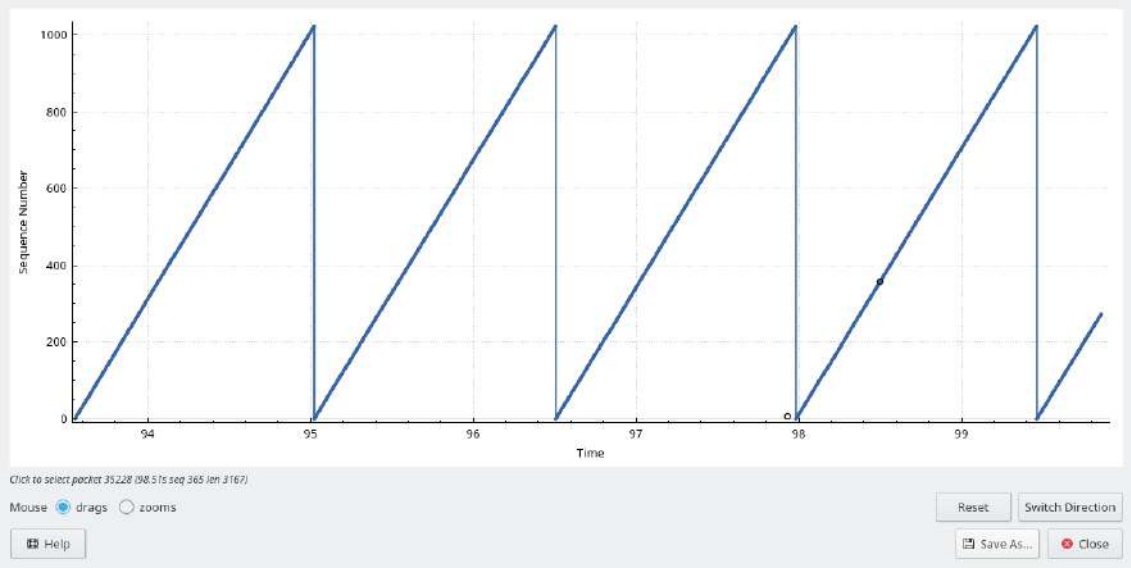


Figure 102. The RLC Graph window

The image of the RLC Graph is borrowed from Wireshark wiki.

LTE RLC Traffic Statistics Window

Statistics of the captured LTE RLC traffic. This window will summarize the LTE RLC traffic found in the capture.

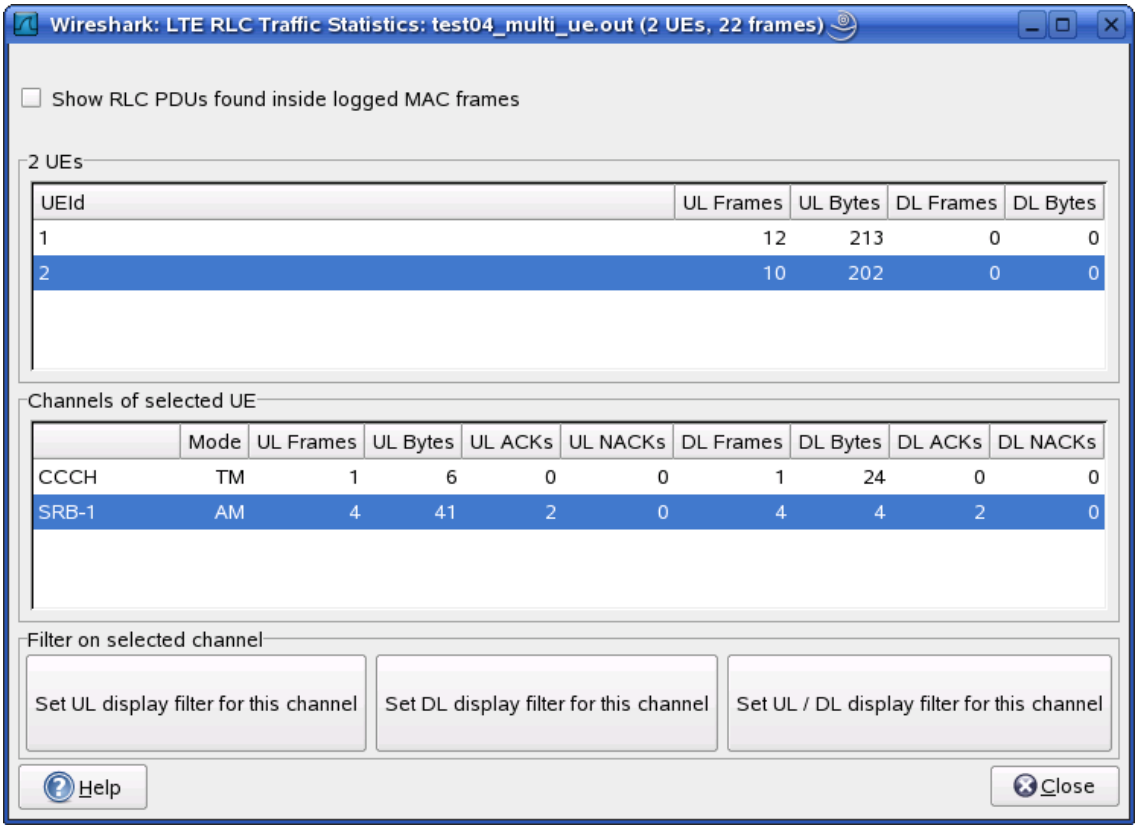


Figure 103. The "LTE RLC Traffic Statistics" window

At the top, the check-box allows this window to include RLC PDUs found within MAC PDUs or not.

This will affect both the PDUs counted as well as the display filters generated (see below).

The upper list shows summaries of each active UE. Each row in the lower list shows statistical highlights for individual channels within the selected UE.

The lower part of the windows allows display filters to be generated and set for the selected channel. Note that in the case of Acknowledged Mode channels, if a single direction is chosen, the generated filter will show data in that direction and control PDUs in the opposite direction.

MTP3 Windows

The Message Transfer Part level 3 (MTP3) protocol is a part of the Signaling System 7 (SS7). The Public Switched Telephone Networks use it for reliable, unduplicated and in-sequence transport of SS7 messaging between communication partners.

This menu shows MTP3 Statistics and MTP3 Summary windows.

Osmux Windows

OSmux is a multiplex protocol designed to reduce bandwidth usage of satellite-based GSM systems's voice (RTP-AMR) and signaling traffic. The OSmux menu opens the packet counter window with the related statistic data. The user can filter, copy or save the data into a file.

RTP

RTP Streams Window

The RTP streams window shows all RTP streams in capture file. Streams can be selected there and on selected streams other tools can be initiated.

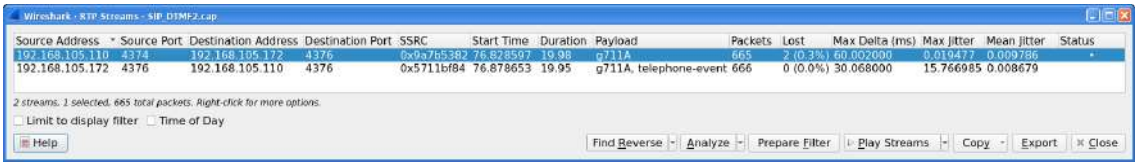


Figure 104. The "RTP Streams" window

User can use shortcuts:

- Selection
  - `Ctrl` + `A` - Select all streams
  - `Ctrl` + `I` - Invert selection
  - `Ctrl` + `Shift` + `A` - Select none
  - Note: Common `Mouse click`, `Shift` + `Mouse click` and `Ctrl` + `Mouse click` works too

- Find Reverse
  - `R` - Try search for reverse streams related to already selected streams. If found, selects them in the list too.
  - **[Shift+R]** - Select all pair streams (forward/reverse relation).
  - **[Ctrl+R]** - Select all single streams (no reverse stream does exist).
- `G` - Go to packet of stream under the mouse cursor.
- `M` - Mark all packets of selected streams.
- `P` - Prepare filter matching selected streams and apply it.
- `E` - Export selected streams in RTPDump format.
- `A` - Open RTP Stream Analysis window and add selected streams to it.

Available controls are:

- Find Reverse
  - **[Find Reverse]** search for reverse stream of every selected stream. If found, selects it in the list too.
  - **[Find All Pairs]** select all streams which have forward/reverse relation.
  - **[Find Only Single]** select all streams which are single - have no reverse stream.
- **[Analyze]** opens RTP Stream Analysis window. Actions **[Set]**, **[Add]** and **[Remove]** are available.
- **[Prepare Filter]** prepares filter matching selected streams and apply it.
- **[Play Streams]** opens RTP Player window. Actions **[Set]**, **[Add]** and **[Remove]** are available.
- **[Copy]** copies information from table to clipboard in CSV or YAML.
- **[Export]** exports selected streams in RTPDump format.

RTP Stream Analysis Window

The RTP analysis function takes the selected RTP streams and generates a list of statistics on it including graph.

Menu **Telephony › RTP › RTP Stream Analysis** is enabled only when selected packed is RTP packet. When window is opened, selected RTP stream is added to analysis. If **[Ctrl]** is pressed during menu opening, reverse RTP stream (if exists) is added to the window too.

Every stream is shown on own tab. Tabs are numbered as streams are added and its tooltip shows identification of the stream. When tab is closed, number is not reused. Color of tab matches color of graphs on graph tab.

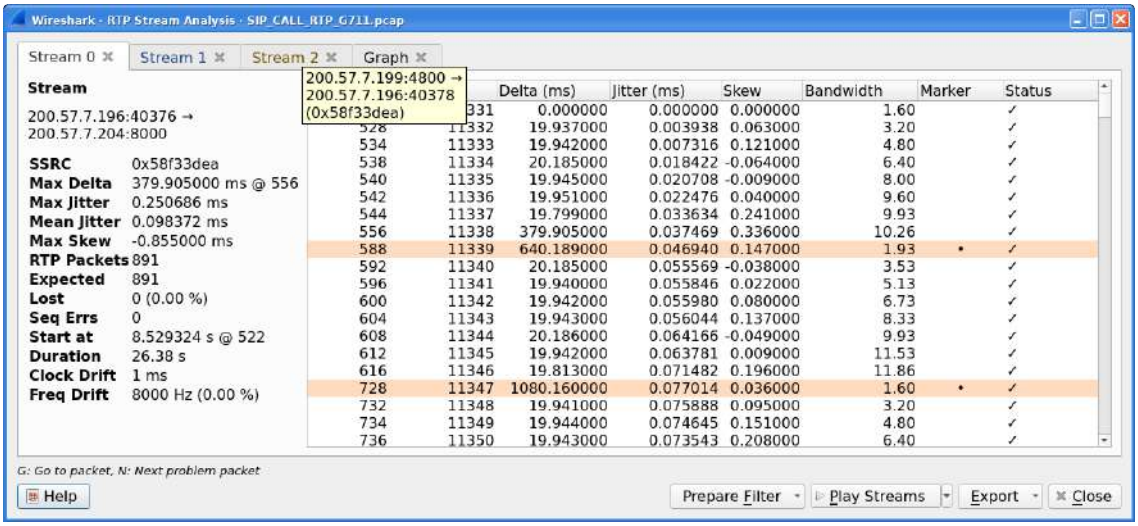


Figure 105. The “RTP Stream Analysis” window

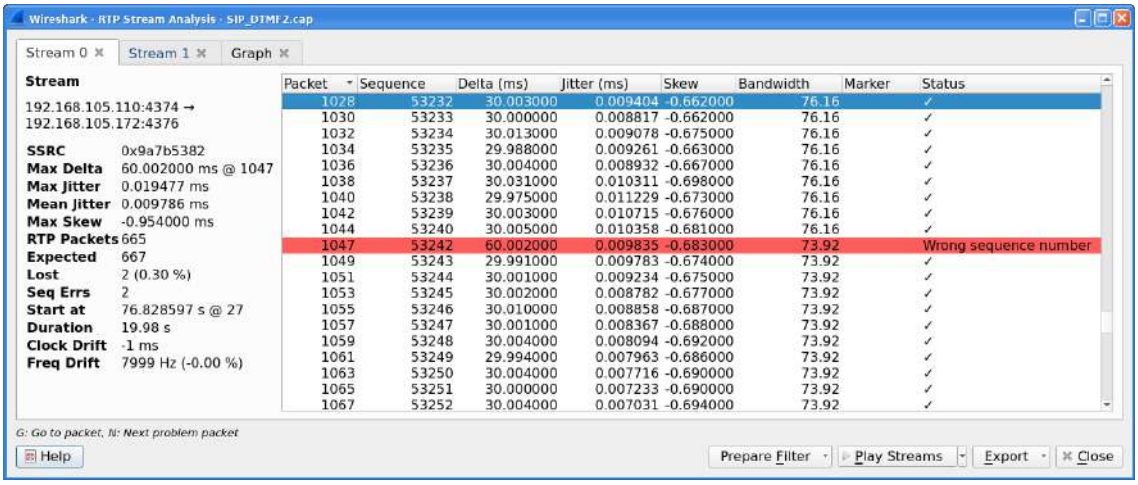


Figure 106. Error indicated in “RTP Stream Analysis” window

Per packet statistic shows:

- Packet number
- Sequence number
- Delta (ms) to last packet
- Jitter (ms)
- Skew
- Bandwidth
- Marker - packet is marked in RTP header
- Status - information related to the packet. E. g. change of codec, DTMF number, warning about incorrect sequence number.

Side panel left to packet list shows stream statistics:

- Maximal delta and at which packet it occurred

- Maximal jitter
- Mean jitter
- Maximal skew
- Count of packets
- Count of lost packets - calculated from sequence numbers
- When the stream starts and first packet number
- Duration of the stream
- Clock drift
- Frequency drift

NOTE	Some statistic columns are calculated only when Wireshark is able to decode codec of RTP stream.

Available shortcuts are:

- `G` - Go to selected packet of stream in packet list
- `N` - Move to next problem packet

Available controls are:

- Prepare Filter
  - **[Current Tab]** prepares filter matching current tab and applies it.
  - **[All Tabs]** prepares filter matching all tabs and applies it.
- **[Play Streams]** opens RTP Player window. Actions **[Set]**, **[Add]** and **[Remove]** are available.
- **[Export]** allows export current stream or all streams as CSV or export graph as image in multiple different formats (PDF, PNG, BMP and JPEG).

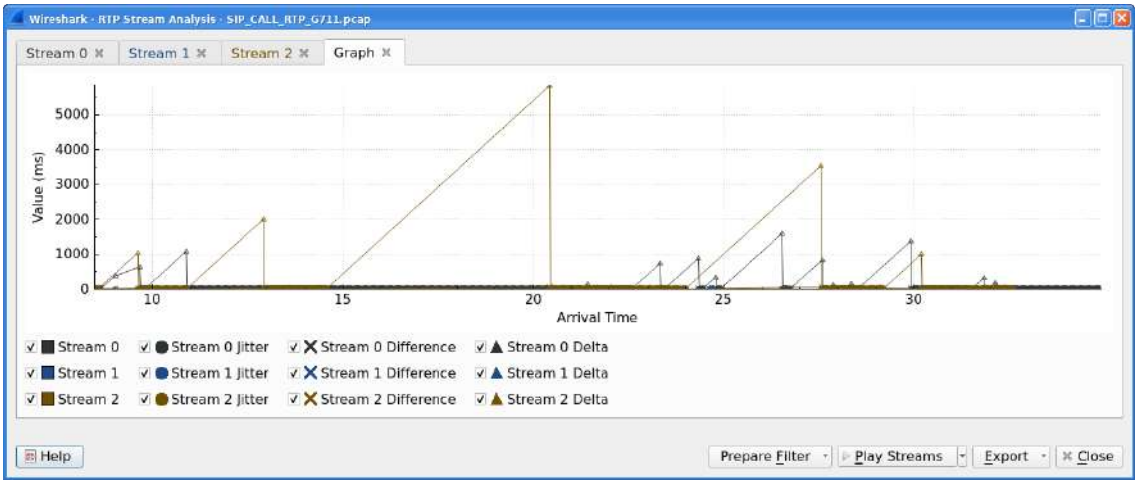


Figure 107. Graph in “RTP Stream Analysis” window

Graph view shows graph of:

- jitter
- difference - absolute value of difference between expected and real time of packet arrival
- delta - time difference from reception of previous packet

for every stream. Checkboxes below graph are enabling or disabling showing of a graph for every stream. **[Stream X]** checkbox enables or disables all graphs for the stream.

NOTE	Stream Analysis window contained tool for save audio and payload for analyzed streams. This tool was moved in Wireshark 3.5.0 to RTP Player window. New tool has more features.

RTP Player Window

The RTP Player function is tool for playing VoIP calls. It shows RTP streams and its waveforms, allows play stream and export it as audio or payload to file. See related concepts in Playing VoIP Calls.

Menu **Telephony › RTP › RTP Player** is enabled only when selected packed is RTP packet. When window is opened, selected RTP stream is added to playlist. If **[Ctrl]** is pressed during menu opening, reverse RTP stream (if exists) is added to the playlist too.

Figure 108. RTP Player window

RTP Player Window consists of three parts:

1. Waveform view
2. Playlist
3. Controls

Waveform view shows visual presentation of RTP stream. Color of waveform and playlist row are matching. Height of wave shows volume.

Waveform shows error marks for Out of Sequence, Jitter Drops, Wrong Timestamps and Inserted Silence marks if it happens in a stream.

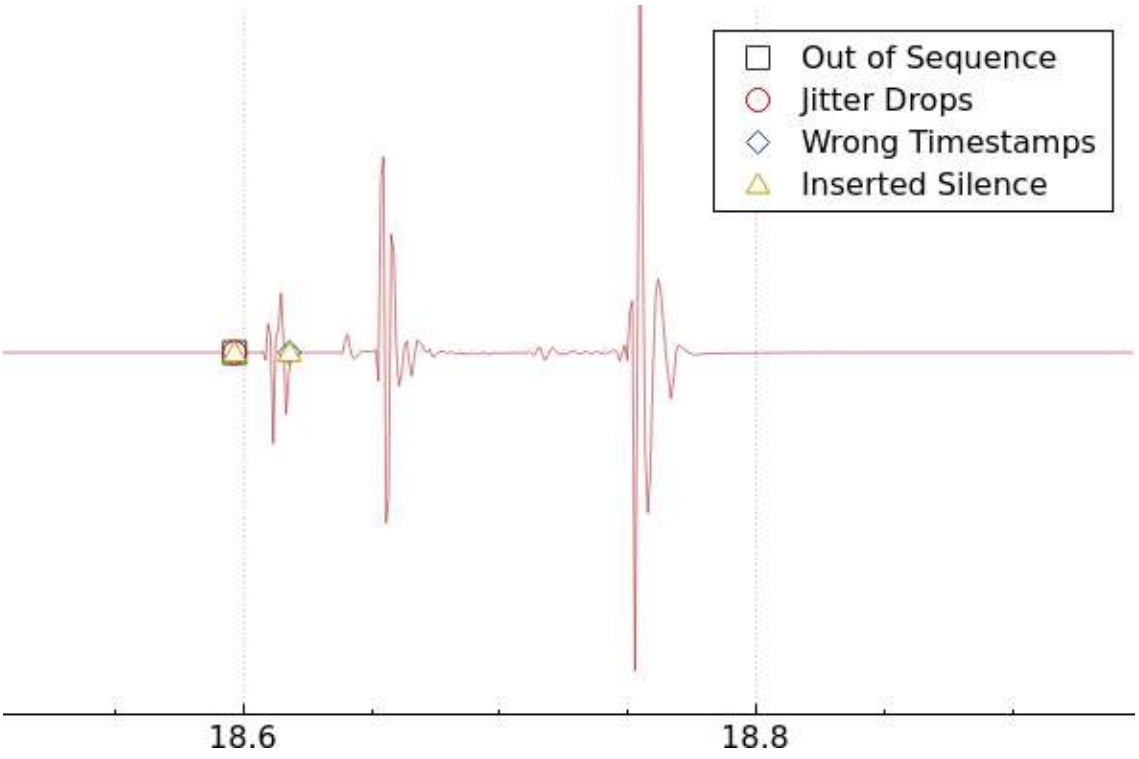


Figure 109. Waveform with error marks

Playlist shows information about every stream:

- Play - Audio routing
- Source Address, Source Port, Destination Address, Destination Port, SSRC
- Setup Frame
  - SETUP <number> is shown, when there is known signaling packet. Number is packet number of signaling packet. Note: Word SETUP is shown even RTP stream was initiated e. g. by SKINNY where no SETUP message exists.
  - RTP <number> is shown, when no related signaling was found. Number is packet number of first packet of the stream.
- Packets - Count of packets in the stream.
- Time Span - Start - Stop (Duration) of the stream
- SR - Sample rate of used codec
- PR - Decoded play rate used for stream playing
- Payloads - One or more payload types used by the stream

NOTE	When rtp_udp is active, most of streams shows just RTP <number> even there is setup frame in capture. When RTP stream contains multiple codecs, SR and PR is based on first observed coded. Later codecs in stream are resampled to first one.

Controls allow a user to:

- **[Start]/[Pause]/[Stop]** playing of unmuted streams
- **[>>]** enabling/disabling silence skipping
  - Min silence - Minimal duration of silence to skip in seconds. Shorter silence is played as it is.
- Select **[Output audio device]** and **[Output audio rate]**
- Select **[Playback Timing]**
  - Jitter Buffer - Packets outside **[Jitter Buffer]** size are discarded during decoding
  - RTP Timestamp - Packets are ordered and played by its Timestamp, no Jitter Buffer is used
  - Uninterrupted Mode - All gaps (e. g. Comfort Noise, lost packets) are discarded therefore audio is shorted than timespan
- **[Time of Day]** selects whether waveform timescale is shown in seconds from start of capture or in absolute time of received packets
- **[Refresh streams]** refreshes streams during live capture (see Playing audio during live capture). Button is disabled when no live capture is running.
- Inaudible streams
  - **[Select]** select all inaudible streams (streams with zero play rate)
  - **[Deselect]** deselect all inaudible streams (streams with zero play rate)
- **[Analyze]** open RTP Stream Analysis window. Actions **[Set]**, **[Add]** and **[Remove]** are available.
- **[Prepare Filter]** prepare filter matching selected streams and apply it.
- **[Export]** - See Export.

NOTE	RTP Player detects silence just by missing voice samples (Comfort Noise, interrupted RTP, missing RTP, ...) or when some streams are muted.

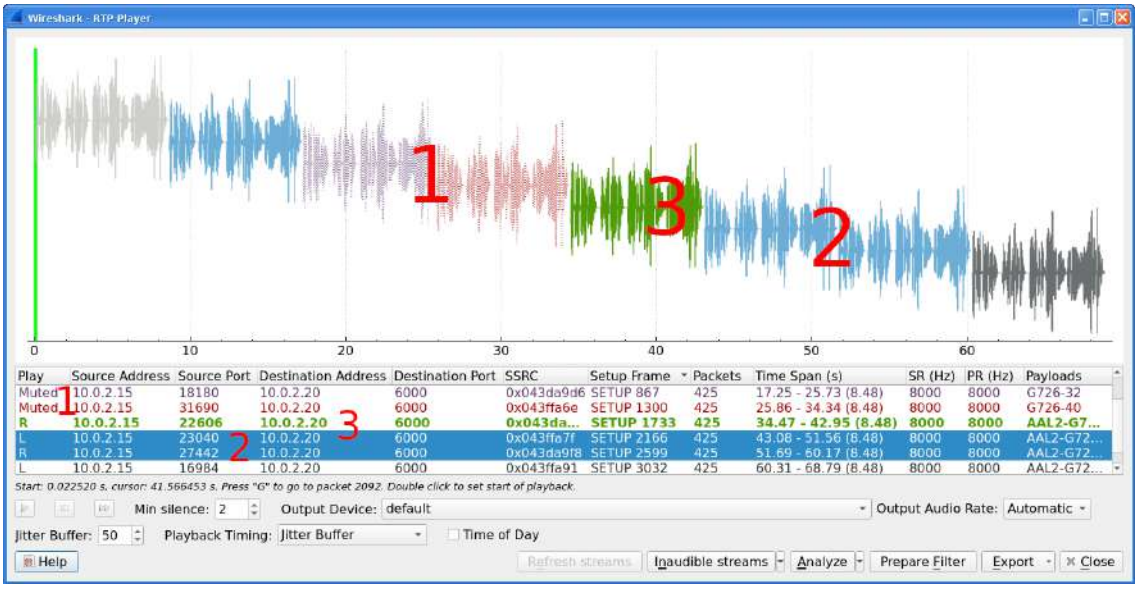


Figure 110. RTP stream state indication

Waveform view and playlist shows state of a RTP stream:

1. stream is muted (dashed waveform, **Muted** is shown in Play column) or unmuted (non-dashed waveform, audio routing is shown in Play column)
2. stream is selected (blue waveform, blue row)
3. stream is below mouse cursor (bold waveform, bold font)

User can control to where audio of a stream is routed to:

- L - Left channel
- L+R - Left and Right (Middle) channel
- R - Left channel
- P - Play (when mono soundcard is available only)
- M - Muted

Audio routing can be changed by double clicking on first column of a row, by shortcut or by menu.

User can use shortcuts:

- Selection
  - `Ctrl` + `A` - Select all streams
  - `Ctrl` + `I` - Invert selection
  - `Ctrl` + `Shift` + `A` - Select none
  - Note: Common `Mouse click`, `Shift` + `Mouse click` and `Ctrl` + `Mouse click` works too
- Go to packet
  - `G` - Go to packet of stream under the mouse cursor

- `Shift` + `G` - Go to setup packet of stream under the mouse cursor

- Audio routing
  - `M` - Mute all selected streams
  - `Shift` + `M` - Unmute all selected streams
  - `Ctrl` + `M` - Invert muting of all selected streams
- `P` - Play audio
- `S` - Stop playing
- `Del` or `Ctrl` + `X` - Remove all selected streams from playlist
- Inaudible steams
  - `N` - Select all inaudible streams
  - `Shift` + `N` - Deselect all inaudible streams

Export

	Export was moved from **RTP Stream Analysis** window to **RTP Player** window in 3.5.0.
NOTE	Wireshark is able to export decoded audio in .au or .wav file format. Prior to version 3.2.0, Wireshark only supported exporting audio using the G.711 codec. From 3.2.0 it supports audio export using any codec with 8000 Hz sampling. From 3.5.0 is supported export of any codec, rate is defined by Output Audio Rate.

Export options available:

- for one or more selected non-muted streams
  - From cursor - Streams are saved from play start cursor. If some streams are shorter, they are removed from the list before save and count of saved streams is lower than count of selected streams.
  - Stream Synchronized Audio - File starts at the begin of earliest stream in export, therefore there is no silence at beginning of exported file.
  - File Synchronized Audio - Streams starts at beginning of file, therefore silence can be at start of file.
- for just one selected stream
  - Payload - just payload with no information about coded is stored in the file

Audio is exported as multi-channel file - one channel per RTP stream. One or two channels are equal to mono or stereo, but Wireshark can export e.g., 100 channels. For playing a tool with multi-channel support must be used (e.g., https://www.audacityteam.org/).

Export of payload function is useful for codecs not supported by Wireshark.

NOTE

Default value of **[Output Audio Rate]** is **[Automatic]**. When multiple codecs with different codec rates are captured, Wireshark decodes each stream with its own play audio rate. Therefore, each stream can have a different audio rate. If you attempt to export audio when there are multiple audio rates, it will fail because .au or .wav require a fixed audio rate.

In this case user must manually select one of rates in **[Output Audio Rate]**, streams will be resampled and audio export succeeds.

RTSP Window

In the Real Time Streaming Protocol (RTSP) menu the user can check the Packet Counter window. It shows Total RTCP Packets and divided into RTSP Response Packets, RTSP Request Packets and Other RTSP packets. The user can filter, copy or save the data into a file.

SCTP Windows

Stream Control Transmission Protocol (SCTP) is a computer network protocol which provides a message transfer in telecommunication in the transport layer. It overcomes some lacks of User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). The SCTP packets consist of the *common header* and the *data chunks*.

The SCTP Analyze Association window shows the statistics of the captured packets between two Endpoints. You can check the different chunk types by pressing **[Chunk Statistics]** button in the `Statistics` tab. In the `Endpoint` tabs you can see various statistics, such as IP addresses, ports and others. You can also check different graphs here.

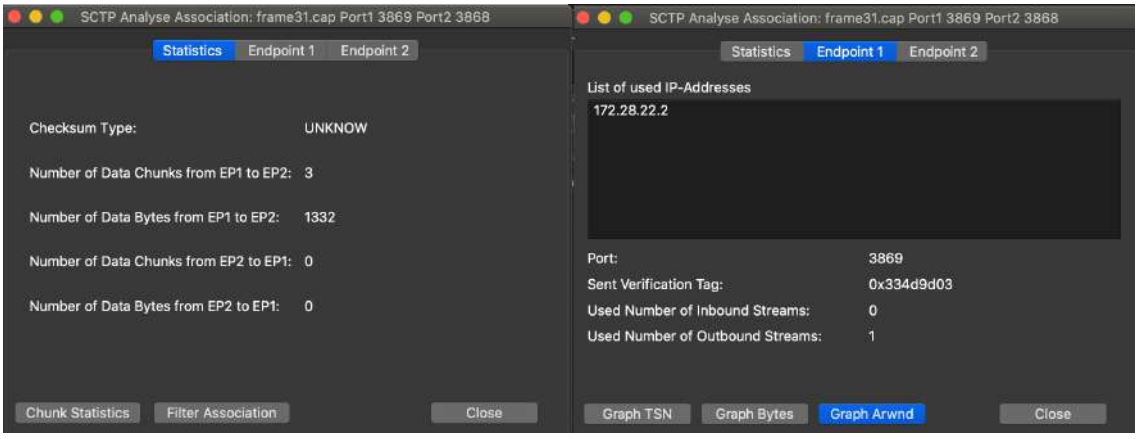


Figure 111. SCTP Analyze Association window

The SCTP Associations window shows the table with the data for captured packets, such as port and counter. You can also call for the SCTP Analyze Association window by pressing the **[Analyze]** button.

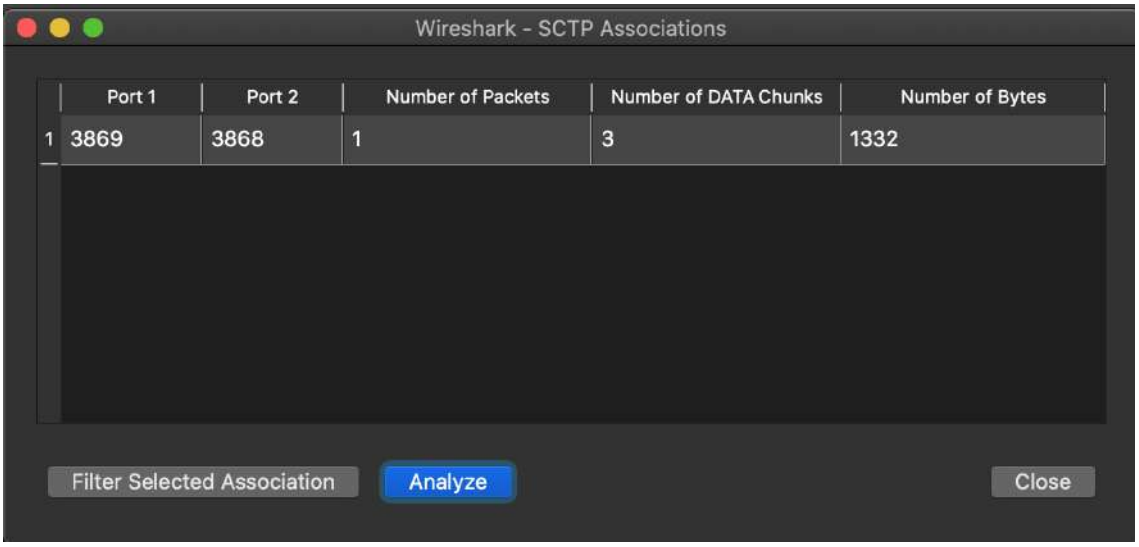


Figure 112. SCTP Associations window

SMPP Operations Window

Short Message Peer-to-Peer (SMPP) protocol uses TCP protocol as its transfer for exchanging Short Message Service (SMS) Messages, mainly between Short Message Service Centers (SMSC). The dissector determines whether the captured packet is SMPP or not by using the heuristics in the fixed header. The SMPP Operations window displays the related statistical data. The user can filter, copy or save the data into a file.

UCP Messages Window

The Universal Computer Protocol (UCP) plays role in transferring Short Messages between a Short Message Service Centre (SMSC) and an application, which is using transport protocol, such as TCP or X.25. The UCP Messages window displays the related statistical data. The user can filter, copy or save the data into a file.

H.225 Window

H.225 telecommunication protocol which is responsible for messages in call signaling and media stream packetization for packet-based multimedia communication systems. The H.225 window shows the counted messages by types and reasons. The user can filter, copy or save the data into a file.

SIP Flows Window

Session Initiation Protocol (SIP) Flows window shows the list of all captured SIP transactions, such as client registrations, messages, calls and so on.

This window will list both complete and in-progress SIP transactions.

Window has same features as VoIP Calls window.

SIP Statistics Window

SIP Statistics window shows captured SIP transactions. It is divided into SIP Responses and SIP Requests. In this window the user can filter, copy or save the statistics into a file.

WAP-WSP Packet Counter Window

The WAP-WSP Packet Counter menu displays the number of packets for each Status Code and PDU Type in Wireless Session Protocol traffic. The user can filter, copy or save the data into a file.

Wireless

Introduction

The Wireless menu provides access to statistics related to wireless traffic.

Bluetooth ATT Server Attributes

Bluetooth ATT Server Attributes window displays a list of captured Attribute Protocol (ATT) packets. The user can filter the list by the interfaces or devices, and also exclude repetitions by checking the `Remove duplicates` check box.

`Handle` is a unique attribute which is specific to the device.

`UUID` is a value which defines a type of an attribute.

`UUID Name` is a specified name for the captured packet.

Bluetooth Devices

The Bluetooth Devices window displays the list of the captured information about devices, such as MAC address, Organizationally Unique Identifier (OUI), Name and other. Users can filter it by interface.

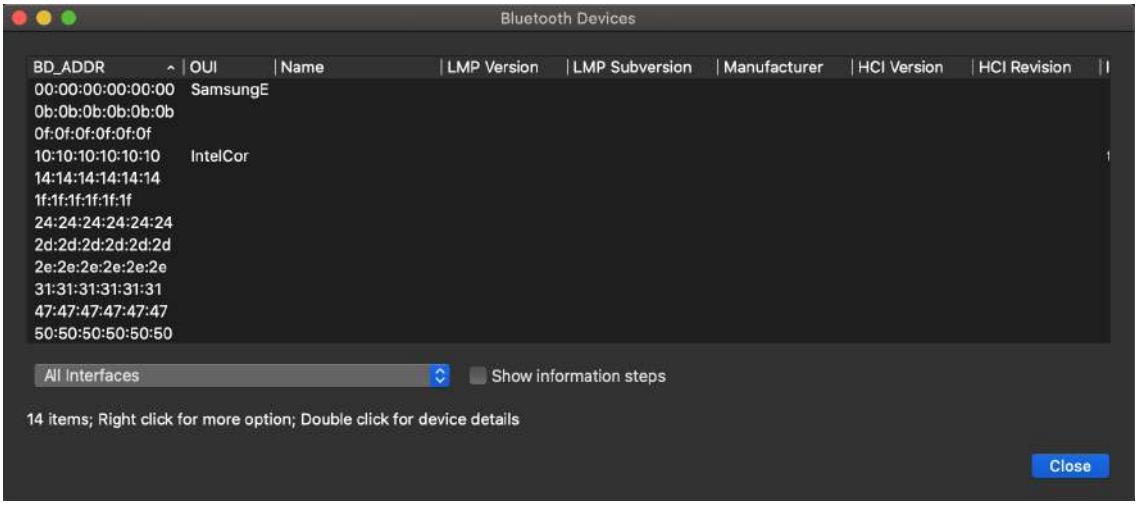


Figure 113. Bluetooth Devices window

Bluetooth HCI Summary

The Bluetooth HCI Summary window displays the summary for the captured Host Controller Interface (HCI) layer packets. This window allows users to apply filters and choose to display information about specific interfaces or devices.

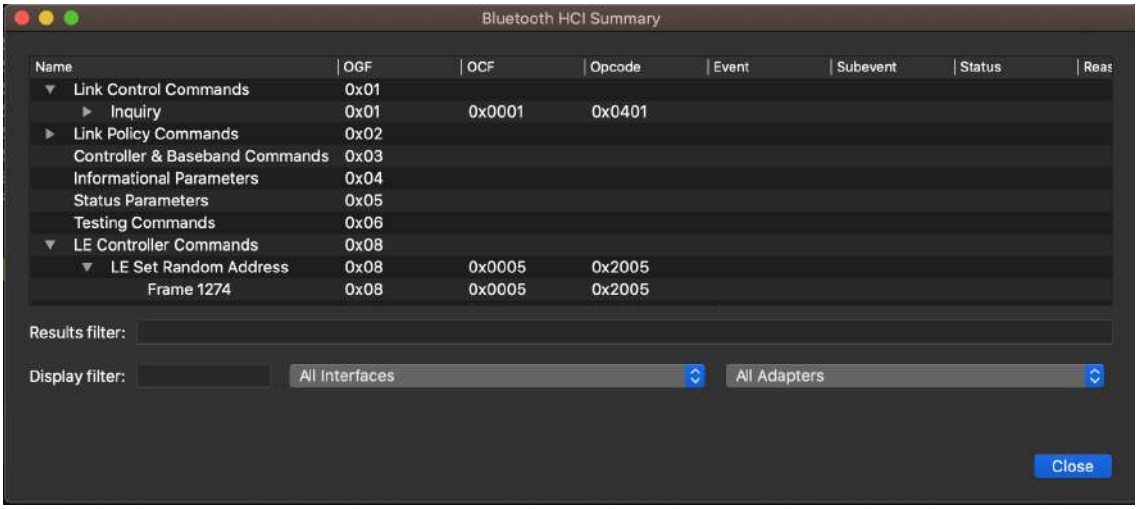


Figure 114. Bluetooth HCI Summary window

WLAN Traffic

Statistics about captured WLAN traffic. This can be found under the **Wireless** menu and summarizes the wireless network traffic found in the capture. Probe requests will be merged into an existing network if the SSID matches.

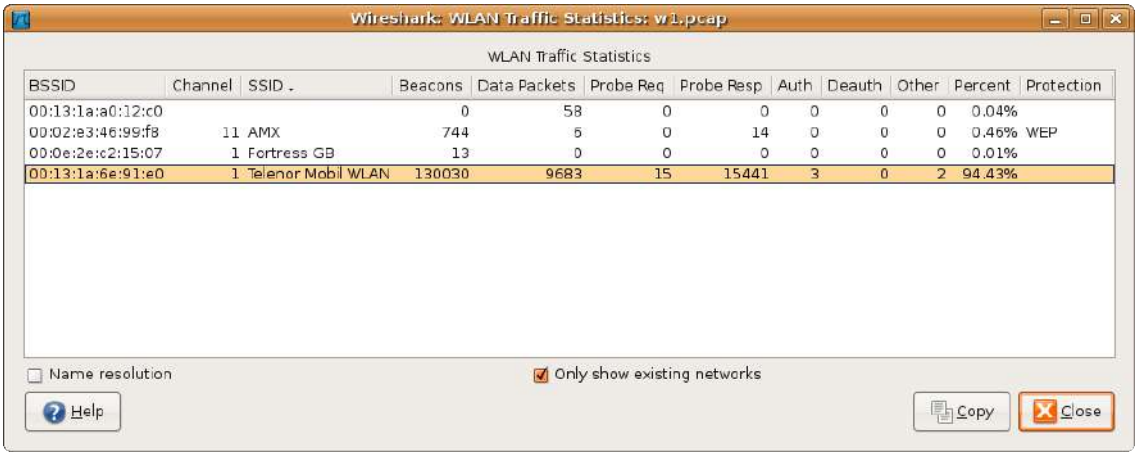


Figure 115. The "WLAN Traffic Statistics" window

Each row in the list shows the statistical values for exactly one wireless network.

Name resolution will be done if selected in the window and if it is active for the MAC layer.

Only show existing networks will exclude probe requests with a SSID not matching any network from the list.

The **[Copy]** button will copy the list values to the clipboard in CSV (Comma Separated Values) format.

TIP	This window will be updated frequently, so it will be useful, even if you open it before (or while) you are doing a live capture.

Customizing Wireshark

Introduction

Wireshark's default behavior will usually suit your needs pretty well. However, as you become more familiar with Wireshark, it can be customized in various ways to suit your needs even better. In this chapter we explore:

- How to start Wireshark with command line parameters
- How to colorize the packet list
- How to control protocol dissection
- How to use the various preference settings

Start Wireshark from the command line

You can start Wireshark from the command line, but it can also be started from most Window managers as well. In this section we will look at starting it from the command line.

Wireshark supports a large number of command line parameters. To see what they are, simply enter the command *wireshark -h* and the help information shown in Help information available from Wireshark (or something similar) should be printed.

Help information available from Wireshark

```
Wireshark 4.1.0 (v4.1.0rc0-55-gccf720d95daf)
Interactively dump and analyze network traffic.
See https://www.wireshark.org for more information.

Usage: wireshark [options] ... [ <infile> ]

Capture interface:
  -i <interface>, --interface <interface>
                           name or idx of interface (def: first non-loopback)
  -f <capture filter>      packet filter in libpcap filter syntax
  -s <snaplen>, --snapshot-length <snaplen>
                           packet snapshot length (def: appropriate maximum)
  -p, --no-promiscuous-mode
                           don't capture in promiscuous mode
  -k                       start capturing immediately (def: do nothing)
  -S                       update packet display when new packets are captured
  -l                       turn on automatic scrolling while -S is in use
  -I, --monitor-mode       capture in monitor mode, if available
  -B <buffer size>, --buffer-size <buffer size>
                           size of kernel buffer (def: 2MB)
```

```
  -y <link type>, --linktype <link type>
                           link layer type (def: first appropriate)
  --time-stamp-type <type> timestamp method for interface
  -D, --list-interfaces    print list of interfaces and exit
  -L, --list-data-link-types
                           print list of link-layer types of iface and exit
  --list-time-stamp-types  print list of timestamp types for iface and exit

Capture stop conditions:
  -c <packet count>        stop after n packets (def: infinite)
  -a <autostop cond.> ..., --autostop <autostop cond.> ...
                           duration:NUM - stop after NUM seconds
                           filesize:NUM - stop this file after NUM KB
                              files:NUM - stop after NUM files
                            packets:NUM - stop after NUM packets
Capture output:
  -b <ringbuffer opt.> ..., --ring-buffer <ringbuffer opt.>
                           duration:NUM - switch to next file after NUM secs
                           filesize:NUM - switch to next file after NUM KB
                              files:NUM - ringbuffer: replace after NUM files
                            packets:NUM - switch to next file after NUM packets
                           interval:NUM - switch to next file when the time is
                                          an exact multiple of NUM secs
Input file:
  -r <infile>, --read-file <infile>
                           set the filename to read from (no pipes or stdin!)

Processing:
  -R <read filter>, --read-filter <read filter>
                           packet filter in Wireshark display filter syntax
  -n                       disable all name resolutions (def: all enabled)
  -N <name resolve flags>  enable specific name resolution(s): "mnNtdv"
  -d <layer_type>==<selector>,<decode_as_protocol> ...
                           "Decode As", see the man page for details
                           Example: tcp.port==8888,http
  --enable-protocol <proto_name>
                           enable dissection of proto_name
  --disable-protocol <proto_name>
                           disable dissection of proto_name
  --enable-heuristic <short_name>
                           enable dissection of heuristic protocol
  --disable-heuristic <short_name>
                           disable dissection of heuristic protocol

User interface:
  -C <config profile>      start with specified configuration profile
  -H                       hide the capture info dialog during packet capture
  -Y <display filter>, --display-filter <display filter>
```

```
                           start with the given display filter
  -g <packet number>       go to specified packet number after "-r"
  -J <jump filter>         jump to the first packet matching the (display)
                           filter
  -j                       search backwards for a matching packet after "-J"
  -t a|ad|adoy|d|dd|e|r|u|ud|udoy
                           format of time stamps (def: r: rel. to first)
  -u s|hms                 output format of seconds (def: s: seconds)
  -X <key>:<value>         eXtension options, see man page for details
  -z <statistics>          show various statistics, see man page for details

Output:
  -w <outfile|->           set the output filename (or '-' for stdout)
  --capture-comment <comment>
                           add a capture file comment, if supported
  --temp-dir <directory>   write temporary files to this directory
                           (default: /tmp)

Diagnostic output:
  --log-level <level>      sets the active log level ("critical", "warning", etc.)
  --log-fatal <level>      sets level to abort the program ("critical" or "warning")
  --log-domains <[!]list>  comma separated list of the active log domains
  --log-debug <[!]list>    comma separated list of domains with "debug" level
  --log-noisy <[!]list>    comma separated list of domains with "noisy" level
  --log-file <path>        file to output messages to (in addition to stderr)

Miscellaneous:
  -h, --help               display this help and exit
  -v, --version            display version info and exit
  -P <key>:<path>          persconf:path - personal configuration files
                           persdata:path - personal data files
  -o <name>:<value> ...    override preference or recent setting
  -K <keytab>              keytab file to use for kerberos decryption
  --display <X display>    X display to use
  --fullscreen             start Wireshark in full screen
```

We will examine each of the command line options in turn.

The first thing to notice is that issuing the command `wireshark` by itself will launch Wireshark. However, you can include as many of the command line parameters as you like. Their meanings are as follows (in alphabetical order):

-a <capture autostop condition>

--autostop <capture autostop condition>

Specify a criterion that specifies when Wireshark is to stop writing to a capture file. The criterion is of the form test:value, where test is one of:

duration:value

Stop writing to a capture file after value of seconds have elapsed.

filesize:value

Stop writing to a capture file after it reaches a size of value kilobytes (where a kilobyte is 1000 bytes, not 1024 bytes). If this option is used together with the -b option, Wireshark will stop writing to the current capture file and switch to the next one if filesize is reached.

files:value

Stop writing to capture files after value number of files were written.

packets:value

Stop writing to a capture file after value number of packets were written.

-b <capture ring buffer option>

If a maximum capture file size was specified, this option causes Wireshark to run in "ring buffer" mode, with the specified number of files. In "ring buffer" mode, Wireshark will write to several capture files. Their name is based on the number of the file and on the creation date and time.

When the first capture file fills up Wireshark will switch to writing to the next file, and so on. With the files option it's also possible to form a "ring buffer." This will fill up new files until the number of files specified, at which point the data in the first file will be discarded so a new file can be written.

If the optional duration is specified, Wireshark will also switch to the next file when the specified number of seconds has elapsed even if the current file is not completely filled up.

duration:value

Switch to the next file after value seconds have elapsed, even if the current file is not completely filled up.

filesize:value

Switch to the next file after it reaches a size of value kilobytes (where a kilobyte is 1000 bytes, not 1024 bytes).

files:value

Begin again with the first file after value number of files were written (form a ring buffer).

packets:value

Switch to the next file after value number of packets were written, even if the current file is not completely filled up.

interval:value

Switch to the next file when the time is an exact multiple of value seconds.

-B <capture buffer size>

--buffer-size <capture buffer size>

Set capture buffer size (in MB, default is 2MB). This is used by the capture driver to buffer packet data until that data can be written to disk. If you encounter packet drops while capturing, try to increase this size. Not supported on some platforms.

-C <config profile>

Start with the specified configuration profile.

-c <capture packet count>

This option specifies the maximum number of packets to capture when capturing live data. It would be used in conjunction with the `-k` option.

--capture-comment <comment>

Add the comment string to the capture file, if supported by the file format.

-d <layer_type>==<selector>,<decode_as_protocol>

"Decode As", see User Specified Decodes for details. Example: tcp.port==8888,http

-D

--list-interfaces

Print a list of the interfaces on which Wireshark can capture, then exit. For each network interface, a number and an interface name, possibly followed by a text description of the interface, is printed. The interface name or the number can be supplied to the `-i` flag to specify an interface on which to capture.

This can be useful on systems that don't have a command to list them (e.g., Windows systems, or UNIX systems lacking `ifconfig -a`). The number can be especially useful on Windows, where the interface name is a GUID.

Note that "can capture" means that Wireshark was able to open that device to do a live capture. If, on your system, a program doing a network capture must be run from an account with special privileges, then, if Wireshark is run with the `-D` flag and is not run from such an account, it will not list any interfaces.

--display <DISPLAY>

Set the X display to use, instead of the one defined in the environment, or the default display.

--enable-protocol <proto_name>

--disable-protocol <proto_name>

Enable and disable the dissection of the protocol.

--enable-heuristic <short_name>

--disable-heuristic <short_name>

Enable and disable the dissection of the heuristic protocol.

-f <capture filter>

This option sets the initial capture filter expression to be used when capturing packets.

--fullscreen

Start Wireshark in full screen.

-g <packet number>

After reading in a capture file using the -r flag, go to the given packet number.

-h

--help

This option requests Wireshark to print its version and usage instructions (as shown here) and exit.

-H

Hide the capture info dialog during live packet capture.

-i <capture interface>

--interface <capture interface>

Set the name of the network interface or pipe to use for live packet capture.

Network interface names should match one of the names listed in `wireshark -D` (described above). A number, as reported by `wireshark -D`, can also be used. If you're using UNIX, `netstat -i`, `ifconfig -a` or `ip link` might also work to list interface names, although not all versions of UNIX support the `-a` flag to `ifconfig`.

If no interface is specified, Wireshark searches the list of interfaces, choosing the first non-loopback interface if there are any non-loopback interfaces, and choosing the first loopback interface if there are no non-loopback interfaces; if there are no interfaces, Wireshark reports an error and doesn't start the capture.

Pipe names should be either the name of a FIFO (named pipe) or "-" to read data from the standard input. Data read from pipes must be in standard libpcap format.

-J <jump filter>

After reading in a capture file using the `-r` flag, jump to the first packet which matches the filter expression. The filter expression is in display filter format. If an exact match cannot be found the first packet afterwards is selected.

-I

--monitor-mode

Capture wireless packets in monitor mode if available.

-j

Use this option after the `-J` option to search backwards for a first packet to go to.

-k

The `-k` option specifies that Wireshark should start capturing packets immediately. This option requires the use of the `-i` parameter to specify the interface that packet capture will occur from.

-K <keytab file>

Use the specified file for Kerberos decryption.

-l

This option turns on automatic scrolling if the packet list pane is being updated automatically as packets arrive during a capture (as specified by the `-S` flag).

-L

--list-data-link-types

List the data link types supported by the interface and exit.

--list-time-stamp-types

List timestamp types configurable for the interface and exit.

-m <font>

This option sets the name of the font used for most text displayed by Wireshark.

-n

Disable network object name resolution (such as hostname, TCP and UDP port names).

-N <name resolving flags>

Turns on name resolving for particular types of addresses and port numbers. The argument is a string that may contain the following letters:

N

Use external name resolver.

d

Enable name resolution from captured DNS packets.

m

Enable MAC address resolution.

n

Enable network address resolution.

t

Enable transport layer port number resolution.

v

Enable VLAN ID resolution.

-o <preference or recent settings>

Sets a preference or recent value, overriding the default value and any value read from a preference or recent file. The argument to the flag is a string of the form *prefname:value*, where *prefname* is the name of the preference (which is the same name that would appear in the `preferences` or `recent` file), and *value* is the value to which it should be set. Multiple instances of `-o <preference settings> ` can be given on a single command line.

An example of setting a single preference would be:

```
wireshark -o mgcp.display_dissect_tree:TRUE
```

An example of setting multiple preferences would be:

```
wireshark -o mgcp.display_dissect_tree:TRUE -o mgcp.udp.callagent_port:2627
```

You can get a list of all available preference strings from the preferences file. See Files and Folders for details.

User access tables can be overridden using “uat,” followed by the UAT file name and a valid record for the file:

```
wireshark -o "uat:user_dlts:\"User 0 (DLT=147)\",\"http\",\"0\",\"\",\"0\",\"\""
```

The example above would dissect packets with a libpcap data link type 147 as HTTP, just as if you had configured it in the DLT_USER protocol preferences.

-p

--no-promiscuous-mode

Don’t put the interface into promiscuous mode. Note that the interface might be in promiscuous mode for some other reason. Hence, `-p` cannot be used to ensure that the only traffic that is captured is traffic sent to or from the machine on which Wireshark is running, broadcast traffic, and multicast traffic to addresses received by that machine.

-P <path setting>

Special path settings usually detected automatically. This is used for special cases, e.g., starting Wireshark from a known location on an USB stick.

The criterion is of the form key:path, where key is one of:

persconf:path

Path of personal configuration files, like the preferences files.

persdata:path

Path of personal data files, it's the folder initially opened. After the initialization, the recent file will keep the folder last used.

-r <infile>

--read-file <infile>

This option provides the name of a capture file for Wireshark to read and display. This capture file can be in one of the formats Wireshark understands.

-R <read (display) filter>

--read-filter <read (display) filter>

This option specifies a display filter to be applied when reading packets from a capture file. The syntax of this filter is that of the display filters discussed in Filtering Packets While Viewing. Packets not matching the filter are discarded.

-s <capture snapshot length>

--snapshot-length <capture snapshot length>

This option specifies the snapshot length to use when capturing packets. Wireshark will only capture *snaplen* bytes of data for each packet.

-S

This option specifies that Wireshark will display packets as it captures them. This is done by capturing in one process and displaying them in a separate process. This is the same as "Update list of packets in real time" in the "Capture Options" dialog box.

-t <time stamp format>

This option sets the format of packet timestamps that are displayed in the packet list window. The format can be one of:

r

Relative, which specifies timestamps are displayed relative to the first packet captured.

a

Absolute, which specifies that actual times be displayed for all packets.

ad

Absolute with date, which specifies that actual dates and times be displayed for all packets.

adoy

Absolute with YYYY/DOY date, which specifies that actual dates and times be displayed for all packets.

d

Delta, which specifies that timestamps are relative to the previous packet.

dd: Delta, which specifies that timestamps are relative to the previous displayed packet.

e

Epoch, which specifies that timestamps are seconds since epoch (Jan 1, 1970 00:00:00)

u

Absolute, which specifies that actual times be displayed for all packets in UTC.

ud

Absolute with date, which specifies that actual dates and times be displayed for all packets in UTC.

udoy

Absolute with YYYY/DOY date, which specifies that actual dates and times be displayed for all packets in UTC.

-u <s | hms>

Show timesamps as seconds (“s”, the default) or hours, minutes, and seconds (“hms”)

-v

--version

This option requests Wireshark to print out its version information and exit.

-w <savefile>

This option sets the name of the file to be used to save captured packets. This can be '-' for stdout.

-y <capture link type>

--link-type <capture like types>

If a capture is started from the command line with `-k`, set the data link type to use while capturing packets. The values reported by `-L` are the values that can be used.

--time-stamp-type <type>

If a capture is started from the command line with `-k`, set the time stamp type to use while capturing packets. The values reported by `--list-time-stamp-types` are the values that can be used.

-X <eXtension option>

Specify an option to be passed to a Wireshark/TShark module. The eXtension option is in the form extension_key:value, where extension_key can be:

lua_script:<lua_script_filename>

Tells Wireshark to load the given script in addition to the default Lua scripts.

lua_script[num]:argument

Tells Wireshark to pass the given argument to the lua script identified by *num*, which is the number indexed order of the *lua_script* command. For example, if only one script was loaded with `-X lua_script:my.lua`, then `-X lua_script1:foo` will pass the string *foo* to the *my.lua* script. If two scripts were loaded, such as `-X lua_script:my.lua -X lua_script:other.lua` in that order, then a `-X lua_script2:bar` would pass the string *bar* to the second lua script, ie., *other.lua*.

read_format:<file_type>

Tells Wireshark to use a specific input file type, instead of determining it automatically.

stdin_descr:<description>

Define a description for the standard input interface, instead of the default: "Standard input".

-Y <display filter>

--display-filter <display filter>

Start with the given display filter.

-z <statistics-string>

Get Wireshark to collect various types of statistics and display the result in a window that updates in semi-real time. For the currently implemented statistics consult the Wireshark manual page.

Packet colorization

A very useful mechanism available in Wireshark is packet colorization. You can set up Wireshark so that it will colorize packets according to a display filter. This allows you to emphasize the packets you might be interested in.

You can find a lot of coloring rule examples at the *Wireshark Wiki Coloring Rules page* at https://gitlab.com/wireshark/wireshark/wikis/ColoringRules.

There are two types of coloring rules in Wireshark: temporary rules that are only in effect until you quit the program, and permanent rules that are saved in a preference file so that they are available the next time you run Wireshark.

Temporary rules can be added by selecting a packet and pressing the `Ctrl` key together with one of the number keys. This will create a coloring rule based on the currently selected conversation. It

will try to create a conversation filter based on TCP first, then UDP, then IP and at last Ethernet. Temporary filters can also be created by selecting the **Colorize with Filter › Color X** menu items when right-clicking in the packet detail pane.

To permanently colorize packets, select **View › Coloring Rules...**. Wireshark will display the "Coloring Rules" dialog box as shown in The "Coloring Rules" dialog box.

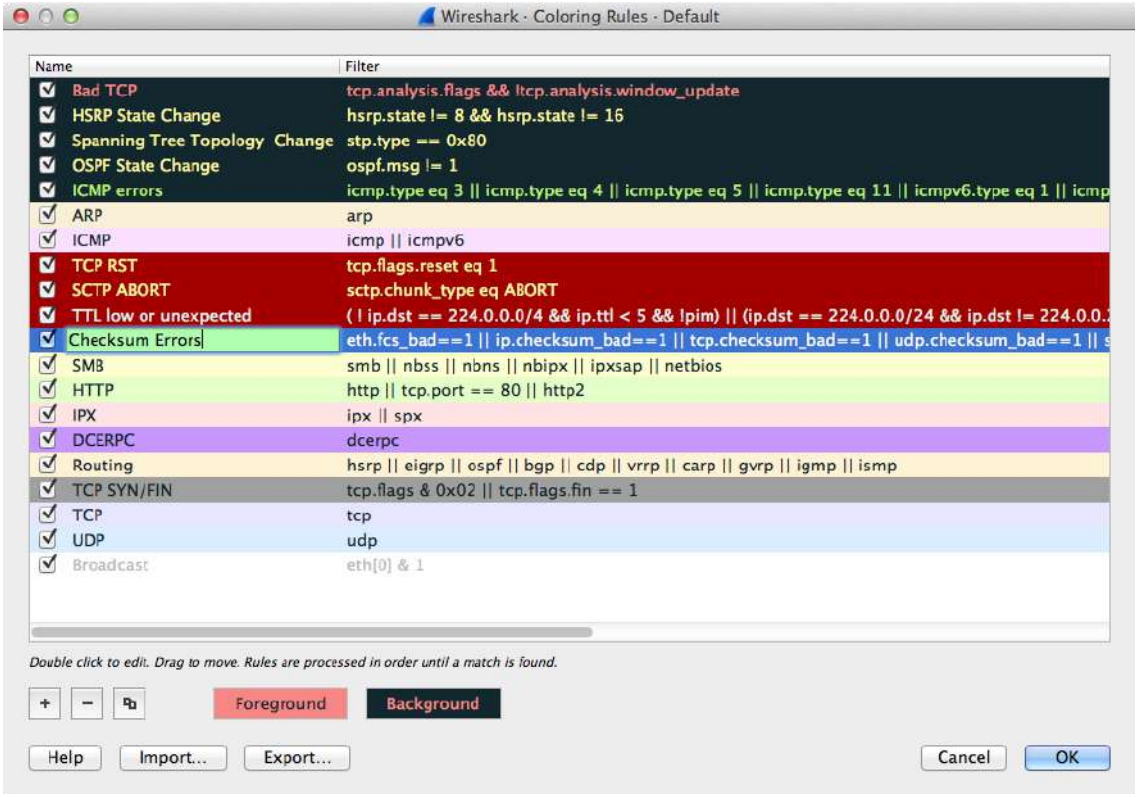


Figure 116. The "Coloring Rules" dialog box

If this is the first time using the Coloring Rules dialog and you're using the default configuration profile you should see the default rules, shown above.

NOTE

The first match wins

More specific rules should usually be listed before more general rules. For example, if you have a coloring rule for UDP before the one for DNS, the rule for DNS may not be applied (DNS is typically carried over UDP and the UDP rule will match first).

You can create a new rule by clicking on the **[+]** button. You can delete one or more rules by clicking the **[-]** button. The "copy" button will duplicate a rule.

You can edit a rule by double-clicking on its name or filter. In The "Coloring Rules" dialog box the name of the rule "Checksum Errors" is being edited. Clicking on the **[Foreground]** and **[Background]** buttons will open a color chooser (A color chooser) for the foreground (text) and background colors respectively.

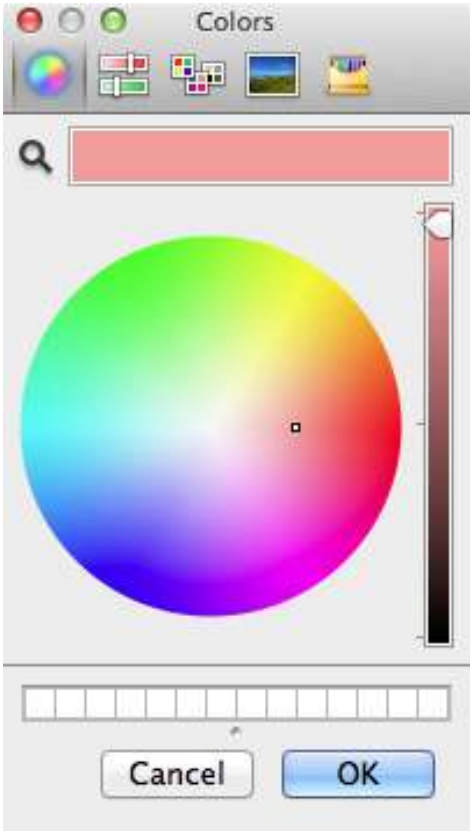


Figure 117. A color chooser

The color chooser appearance depends on your operating system. The macOS color picker is shown. Select the color you desire for the selected packets and click **[OK]**.

Using color filters with Wireshark shows an example of several color filters being used in Wireshark. Note that the frame detail shows that the “Bad TCP” rule was applied, along with the matching filter.

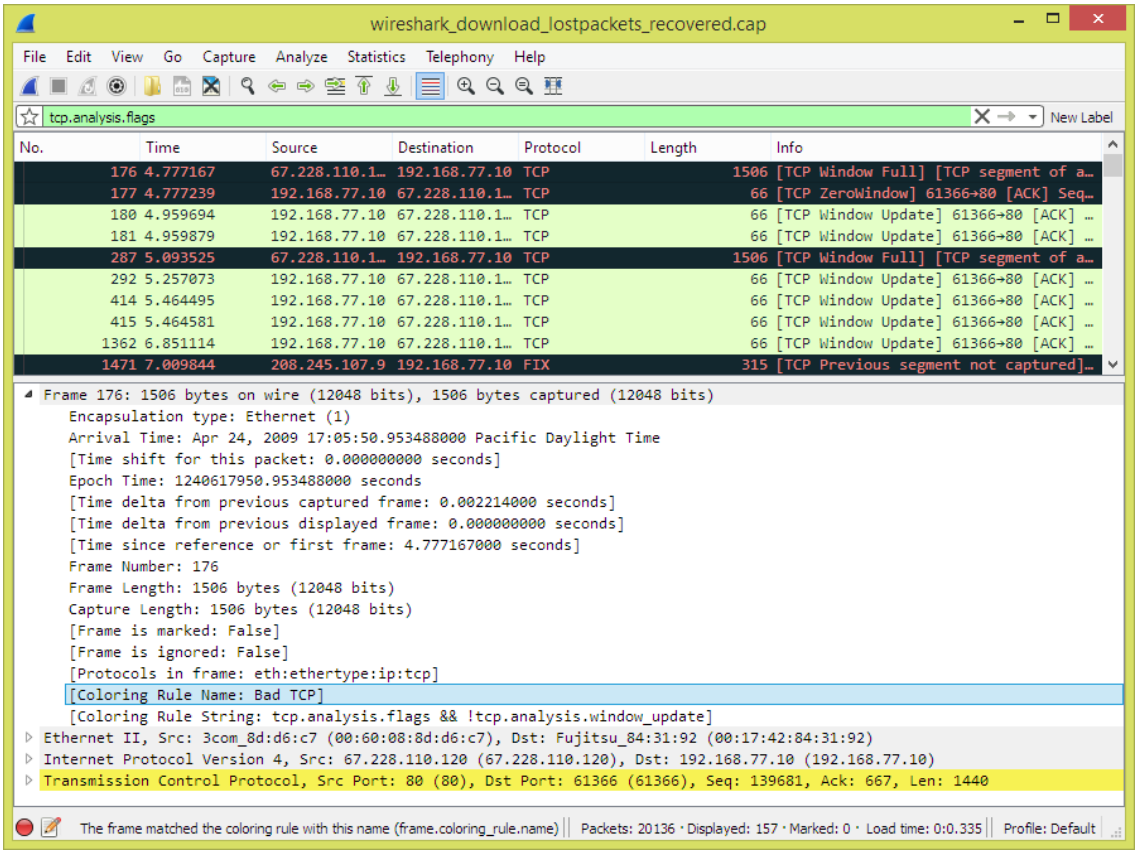


Figure 118. Using color filters with Wireshark

Control Protocol dissection

The user can control how protocols are dissected.

Each protocol has its own dissector, so dissecting a complete packet will typically involve several dissectors. As Wireshark tries to find the right dissector for each packet (using static “routes” and heuristics “guessing”), it might choose the wrong dissector in your specific case. For example, Wireshark won’t know if you use a common protocol on an uncommon TCP port, e.g., using HTTP on TCP port 800 instead of the standard port 80.

There are two ways to control the relations between protocol dissectors: disable a protocol dissector completely or temporarily divert the way Wireshark calls the dissectors.

The “Enabled Protocols” dialog box

The Enabled Protocols dialog box lets you enable or disable specific protocols. Most protocols are enabled by default. When a protocol is disabled, Wireshark stops processing a packet whenever that protocol is encountered.

NOTE	Disabling a protocol will prevent information about higher-layer protocols from being displayed. For example, suppose you disabled the IP protocol and selected a packet containing Ethernet, IP, TCP, and HTTP information. The Ethernet information would be displayed, but the IP, TCP and HTTP information would not - disabling IP would prevent it and the higher-layer protocols from being displayed.

To enable or disable protocols select **Analyze › Enabled Protocols...**. Wireshark will pop up the "Enabled Protocols" dialog box as shown in The "Enabled Protocols" dialog box.

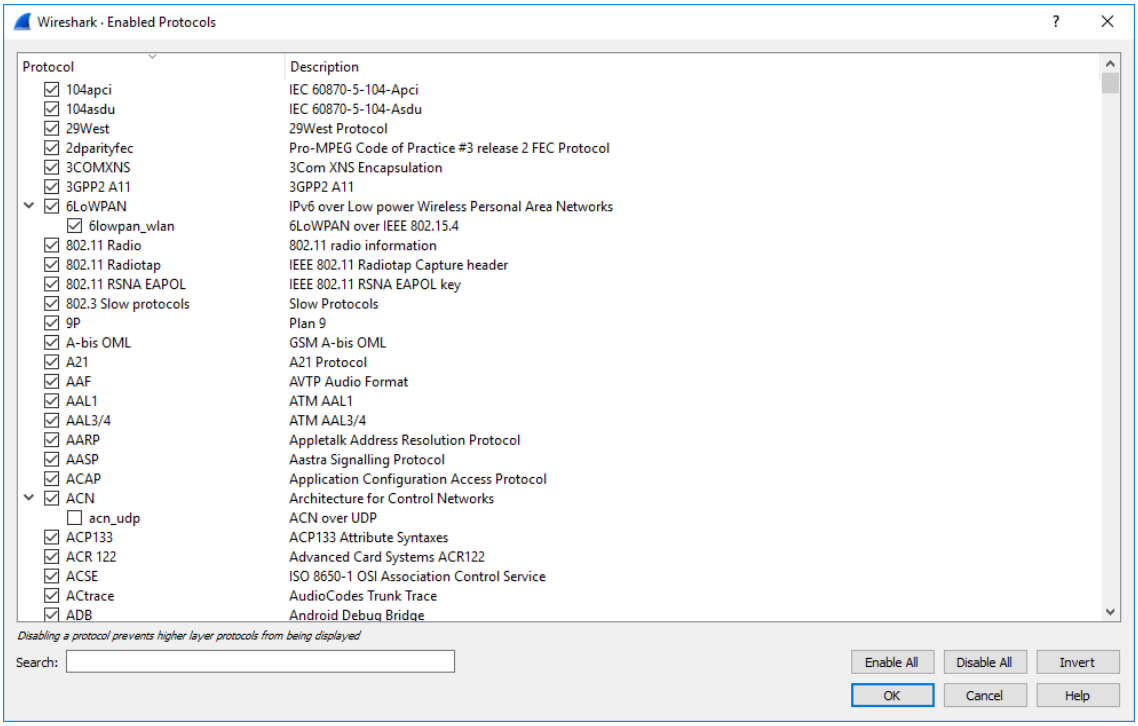


Figure 119. The "Enabled Protocols" dialog box

To disable or enable a protocol, simply click the checkbox using the mouse. Note that typing a few letters of the protocol name in the search box will limit the list to those protocols that contain these letters.

You can choose from the following actions:

[Enable All]

Enable all protocols in the list.

[Disable All]

Disable all protocols in the list.

[Invert]

Toggle the state of all protocols in the list.

[OK]

Save and apply the changes and close the dialog box, see Files and Folders for details.

[Cancel]

Cancel the changes and close the dialog box.

User Specified Decodes

The “Decode As” functionality lets you temporarily divert specific protocol dissections. This might be useful for example, if you do some uncommon experiments on your network.

Decode As is accessed by selecting the **Analyze** › **Decode As…**. Wireshark will pop up the “Decode As” dialog box as shown in The “Decode As” dialog box.

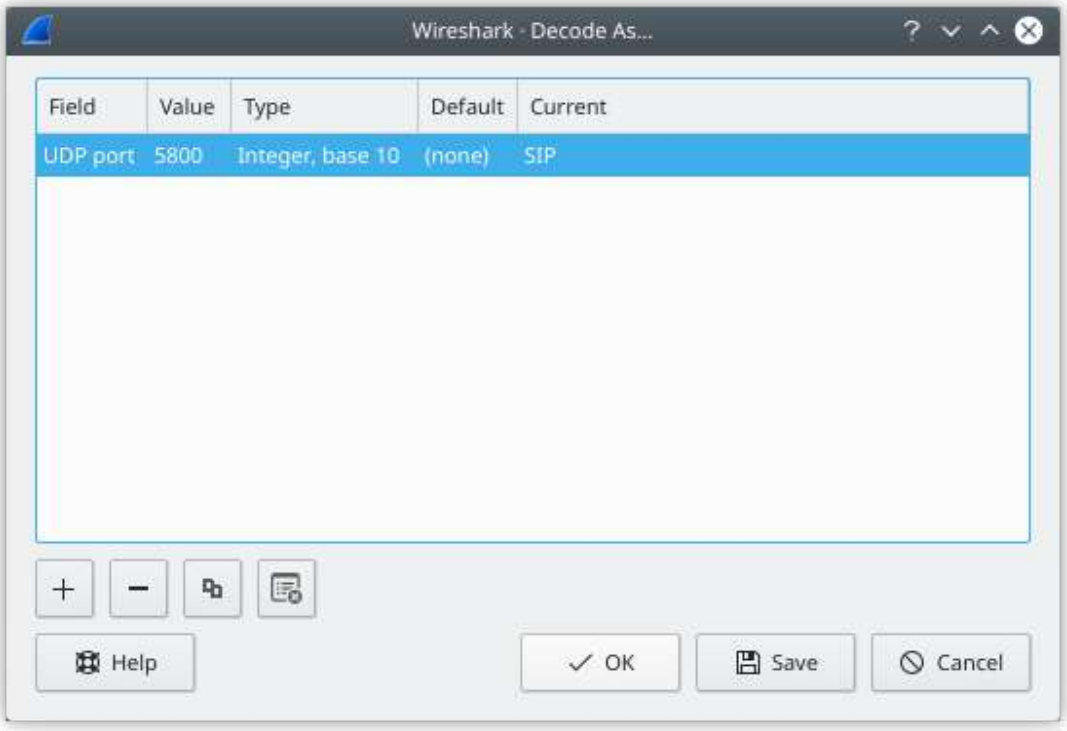


Figure 120. The “Decode As” dialog box

In this dialog you are able to edit entries by means of the edit buttons on the left.

You can also pop up this dialog box from the context menu in the packet list or packet details. It will then contain a new line based on the currently selected packet.

These settings will be lost if you quit Wireshark or change profile unless you save the entries.

[+]

Add new entry for selected packet

[-]

Remove the selected entry.

[Copy]

Copy the selected entry.

[Clear]

Clear the list of user specified decodes.

[OK]

Apply the user specified decodes and close the dialog box.

[Save]

Save and apply the user specified decodes and close the dialog box.

[Cancel]

Cancel the changes and close the dialog box.

Preferences

There are a number of preferences you can set. Simply select the **Edit › Preferences...** (**Wireshark › Preferences...** on macOS) and Wireshark will pop up the Preferences dialog box as shown in The preferences dialog box, with the "User Interface" page as default. On the left side is a tree where you can select the page to be shown.

- The **[OK]** button will apply the preferences settings and close the dialog.
- The **[Cancel]** button will restore all preferences settings to the last saved state.

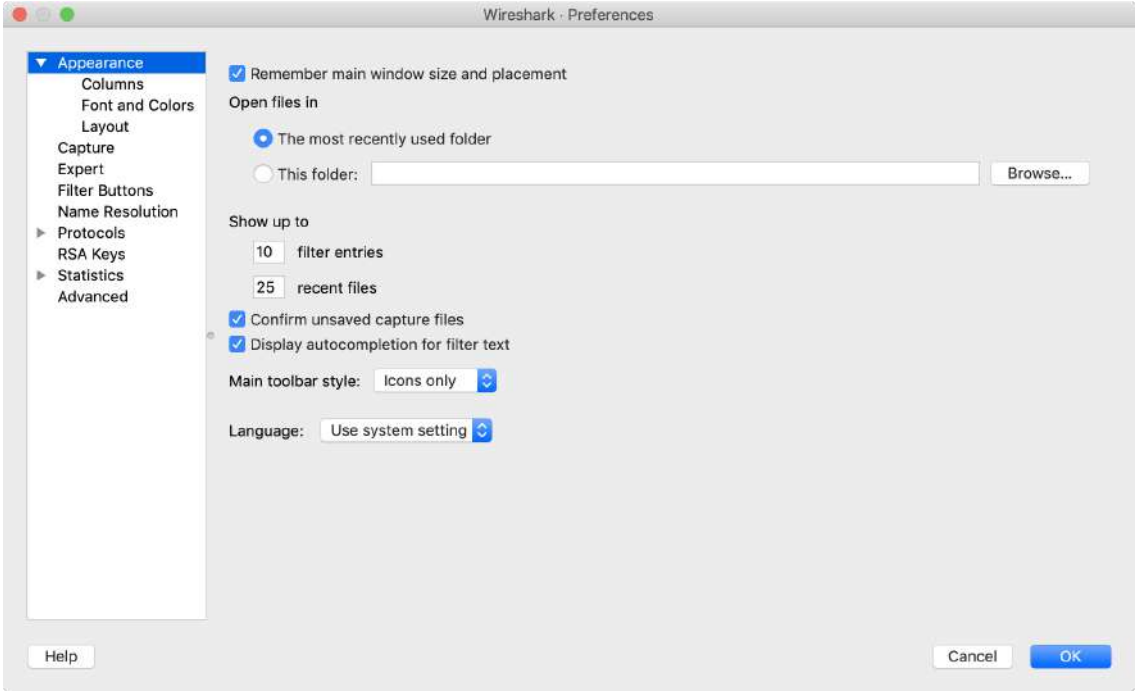


Figure 121. The preferences dialog box

Wireshark supports quite a few protocols, which is reflected in the long list of entries in the “Protocols” pane. You can jump to the preferences for a specific protocol by expanding “Protocols” and quickly typing the first few letters of the protocol name.

The “Advanced” pane will let you view and edit all of Wireshark’s preferences, similar to about:config and chrome:flags in the Firefox and Chrome web browsers.

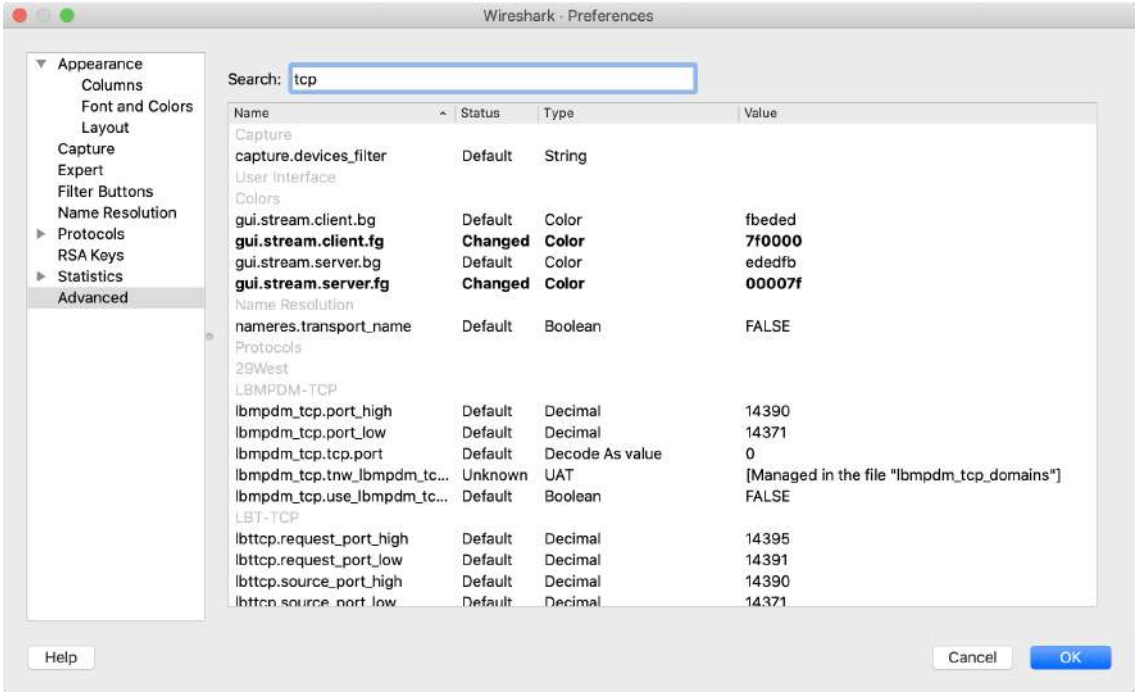


Figure 122. Advanced preferences

You can search for a preference by typing text into the “Search” entry. You can also pass preference names to Wireshark and TShark on the command line. For example, the *gui.prepend_window_title* can be used to differentiate between different instances of Wireshark:

```
$ wireshark -o "gui.prepend_window_title:Internal Network" &
$ wireshark -o "gui.prepend_window_title:External Network" &
```

Configuration Profiles

Configuration Profiles can be used to configure and use more than one set of preferences and configurations. Select the **Edit › Configuration Profiles…** menu item or press `Shift` + `Ctrl` + `A` or `Shift` + `⌘` + `A` (macOS) and Wireshark will pop up the Configuration Profiles dialog box as shown in The configuration profiles dialog box. It is also possible to click in the “Profile” part of the statusbar to popup a menu with available Configuration Profiles (The Statusbar with a configuration profile menu).

Configuration files stored in each profile include:

- Preferences (preferences) (Preferences)

- Capture Filters (cfilters) (Defining And Saving Filters)
- Display Filters (dfilters) (Defining And Saving Filters)
- Coloring Rules (colorfilters) (Packet colorization)
- Disabled Protocols (disabled_protos) (The “Enabled Protocols” dialog box)
- User Accessible Tables:
    - Custom HTTP headers (custom_http_header_fields)
    - Custom IMF headers (imf_header_fields)
    - Custom LDAP AttributeValue types (custom_ldap_attribute_types)
    - Display Filter Macros (dfilter_macros) (Display Filter Macros)
    - ESS Category Attributes (ess_category_attributes) (ESS Category Attributes)
    - MaxMind Database Paths (maxmind_db_paths) (MaxMind Database Paths)
    - K12 Protocols (k12_protos) (Tektronix K12xx/15 RF5 protocols Table)
    - Object Identifier Names and Associated Syntaxes (Object Identifiers)
    - PRES Users Context List (pres_context_list) (PRES Users Context List)
    - SCCP Users Table (sccp_users) (SCCP users Table)
    - SNMP Enterprise Specific Trap Types (snmp_specific_traps) (SNMP Enterprise Specific Trap Types)
    - SNMP Users (snmp_users) (SNMP users Table)
    - User DLTs Table (user_dlts) (User DLTs protocol table)
    - IKEv2 decryption table (ikev2_decryption_table) (IKEv2 decryption table)
    - Protobuf Search Paths (protobuf_search_paths) (Protobuf Search Paths)
    - Protobuf UDP Message Types (protobuf_udp_message_types) (Protobuf UDP Message Types)
- Changed dissector assignments (*decode_as_entries*), which can be set in the “Decode As…” dialog box (User Specified Decodes).
- Some recent settings (recent), such as pane sizes in the Main window (The Main window), column widths in the packet list (The “Packet List” Pane), all selections in the **View** menu (The “View” Menu) and the last directory navigated to in the “File Open” dialog.

All other configurations are stored in the personal configuration folder and are common to all profiles.

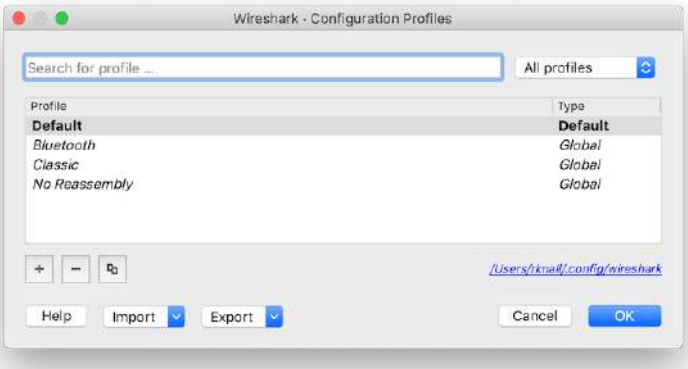


Figure 123. The configuration profiles dialog box

Search for profile ...

The list of profiles can be filtered by entering part of the profile's name into the search box.

Type selection

Profiles can be filtered between displaying "All profiles", "Personal profiles" and "Global profiles"

- Personal profiles - these are profiles stored in the user's configuration directory
- Global profiles - these are profiles provided with Wireshark

New (+)

Create a new profile. The name of the created profile is "New profile" and is highlighted so that you can more easily change it.

Delete (-)

Deletes the selected profile. This includes all configuration files used in this profile. Multiple profiles can be selected and deleted at the same time. It is not possible to delete the "Default" profile or global profiles. Deletion of the "Default" profile will reset this profile.

Copy

Copies the selected profile. This copies the configuration of the profile currently selected in the list. The name of the created profile is the same as the copied profile, with the text "(copy)" and is highlighted so that you can more easily change it.

[Import]

Profiles can be imported from zip-archives as well as directly from directory structures. Profiles, which already exist by name will be skipped, as well as profiles named "Default".

[Export]

Profiles can be exported to a zip-archive. Global profiles, as well as the default profile will be skipped during export. Profiles can be selected in the list individually and only the selected profiles will be exported

[OK]

This button saves all changes, applies the selected profile and closes the dialog.

[Cancel]

Close this dialog. This will discard unsaved settings, new profiles will not be added and deleted profiles will not be deleted.

[Help]

Show this help page.

User Table

The User Table editor is used for managing various tables in Wireshark. Its main dialog works very similarly to that of Packet colorization.

Display Filter Macros

Display Filter Macros are a mechanism to create shortcuts for complex filters. For example, defining a display filter macro named *tcp_conv* whose text is

```
(ip.src == $1 and ip.dst == $2 and tcp.srcport == $3 and tcp.dstport == $4)
or (ip.src == $2 and ip.dst == $1 and tcp.srcport == $4 and tcp.dstport == $3)
```

would allow to use a display filter like

```
${tcp_conv:10.1.1.2;10.1.1.3;1200;1400}
```

instead of typing the whole filter.

Display Filter Macros can be managed with a user table, as described in User Table, by selecting **Analyze › Display Filter Macros** from the menu. The User Table has the following fields:

Name

The name of the macro.

Text

The replacement text for the macro it uses $1, $2, $3, ... as the input arguments.

ESS Category Attributes

Wireshark uses this table to map ESS Security Category attributes to textual representations. The values to put in this table are usually found in an XML SPIF, which is used for defining security

labels.

This table is a user table, as described in User Table, with the following fields:

Tag Set

An Object Identifier representing the Category Tag Set.

Value

The value (Label And Cert Value) representing the Category.

Name

The textual representation for the value.

MaxMind Database Paths

If your copy of Wireshark supports MaxMind's MaxMindDB library, you can use their databases to match IP addresses to countries, cites, autonomous system numbers, and other bits of information. Some databases are available at no cost for registered users, while others require a licensing fee. See the MaxMind web site for more information.

The configuration for the MaxMind database is a user table, as described in User Table, with the following fields:

Database pathname

This specifies a directory containing MaxMind data files. Any files ending with *.mmdb* will be automatically loaded.

The locations for your data files are up to you, but `/usr/share/GeoIP` and `/var/lib/GeoIP` are common on Linux and `C:\ProgramData\GeoIP`, `C:\Program Files\Wireshark\GeoIP` might be good choices on Windows.

Previous versions of Wireshark supported MaxMind's original GeoIP Legacy database format. They were configured similar to MaxMindDB files above, except GeoIP files must begin with *Geo* and end with *.dat*. They are no longer supported and MaxMind stopped distributing GeoLite Legacy databases in April 2018.

IKEv2 decryption table

Wireshark can decrypt Encrypted Payloads of IKEv2 (Internet Key Exchange version 2) packets if necessary information is provided. Note that you can decrypt only IKEv2 packets with this feature. If you want to decrypt IKEv1 packets or ESP packets, use Log Filename setting under ISAKMP protocol preference or settings under ESP protocol preference respectively.

This is handled by a user table, as described in User Table, with the following fields:

Initiator's SPI

Initiator's SPI of the IKE_SA. This field takes hexadecimal string without "0x" prefix and the length must be 16 hex chars (represents 8 octets).

Responder's SPI

Responder's SPI of the IKE_SA. This field takes hexadecimal string without "0x" prefix and the length must be 16 hex chars (represents 8 octets).

SK_ei

Key used to encrypt/decrypt IKEv2 packets from initiator to responder. This field takes hexadecimal string without "0x" prefix and its length must meet the requirement of the encryption algorithm selected.

SK_er

Key used to encrypt/decrypt IKEv2 packets from responder to initiator. This field takes hexadecimal string without "0x" prefix and its length must meet the requirement of the encryption algorithm selected.

Encryption Algorithm

Encryption algorithm of the IKE_SA.

SK_ai

Key used to calculate Integrity Checksum Data for IKEv2 packets from responder to initiator. This field takes hexadecimal string without "0x" prefix and its length must meet the requirement of the integrity algorithm selected.

SK_ar

Key used to calculate Integrity Checksum Data for IKEv2 packets from initiator to responder. This field takes hexadecimal string without "0x" prefix and its length must meet the requirement of the integrity algorithm selected.

Integrity Algorithm

Integrity algorithm of the IKE_SA.

Object Identifiers

Many protocols that use ASN.1 use Object Identifiers (OIDs) to uniquely identify certain pieces of information. In many cases, they are used in an extension mechanism so that new object identifiers (and associated values) may be defined without needing to change the base standard.

While Wireshark has knowledge about many of the OIDs and the syntax of their associated values, the extensibility means that other values may be encountered.

Wireshark uses this table to allow the user to define the name and syntax of Object Identifiers that Wireshark does not know about (for example, a privately defined X.400 extension). It also allows

the user to override the name and syntax of Object Identifiers that Wireshark does know about (e.g., changing the name “id-at-countryName” to just “c”).

This table is a user table, as described in User Table, with the following fields:

OID

The string representation of the Object Identifier e.g., “2.5.4.6”.

Name

The name that should be displayed by Wireshark when the Object Identifier is dissected e.g., (“c”);

Syntax

The syntax of the value associated with the Object Identifier. This must be one of the syntaxes that Wireshark already knows about (e.g., “PrintableString”).

PRES Users Context List

Wireshark uses this table to map a presentation context identifier to a given object identifier when the capture does not contain a PRES package with a presentation context definition list for the conversation.

This table is a user table, as described in User Table, with the following fields:

Context Id

An Integer representing the presentation context identifier for which this association is valid.

Syntax Name OID

The object identifier representing the abstract syntax name, which defines the protocol that is carried over this association.

SCCP users Table

Wireshark uses this table to map specific protocols to a certain DPC/SSN combination for SCCP.

This table is a user table, as described in User Table, with the following fields:

Network Indicator

An Integer representing the network indicator for which this association is valid.

Called DPCs

A range of integers representing the dpcs for which this association is valid.

Called SSNs

A range of integers representing the ssns for which this association is valid.

User protocol

The protocol that is carried over this association

SMI (MIB and PIB) Modules

If your copy of Wireshark supports libSMI, you can specify a list of MIB and PIB modules here. The COPS and SNMP dissectors can use them to resolve OIDs.

Module name

The name of the module, e.g., IF-MIB.

SMI (MIB and PIB) Paths

If your copy of Wireshark supports libSMI, you can specify one or more paths to MIB and PIB modules here.

Directory name

A module directory, e.g., `/usr/local/snmp/mibs`. Wireshark automatically uses the standard SMI path for your system, so you usually don't have to add anything here.

SNMP Enterprise Specific Trap Types

Wireshark uses this table to map specific-trap values to user defined descriptions in a Trap PDU. The description is shown in the packet details specific-trap element.

This table is a user table, as described in User Table, with the following fields:

Enterprise OID

The object identifier representing the object generating the trap.

Trap Id

An Integer representing the specific-trap code.

Description

The description to show in the packet details.

SNMP users Table

Wireshark uses this table to verify authentication and to decrypt encrypted SNMPv3 packets.

This table is a user table, as described in User Table, with the following fields:

Engine ID

If given this entry will be used only for packets whose engine id is this. This field takes a

hexadecimal string in the form 0102030405.

Username

This is the userName. When a single user has more than one password for different SNMP-engines the first entry to match both is taken, if you need a catch all engine-id (empty) that entry should be the last one.

Authentication model

Which auth model to use (either “MD5” or “SHA1”).

Password

The authentication password. Use \xDD for unprintable characters. A hexadecimal password must be entered as a sequence of \xDD characters. For example, the hex password 010203040506 must be entered as \x01\x02\x03\x04\x05\x06. The | character must be treated as an unprintable character, i.e., it must be entered as \x5C or \x5c.

Privacy protocol

Which encryption algorithm to use (either “DES” or “AES”).

Privacy password

The privacy password. Use \xDD for unprintable characters. A hexadecimal password must be entered as a sequence of \xDD characters. For example, the hex password 010203040506 must be entered as \x01\x02\x03\x04\x05\x06. The | character must be treated as an unprintable character, i.e., it must be entered as \x5C or \x5c.

Tektronix K12xx/15 RF5 protocols Table

The Tektronix K12xx/15 rf5 file format uses helper files (*.stk) to identify the various protocols that are used by a certain interface. Wireshark doesn’t read these stk files, it uses a table that helps it identify which lowest layer protocol to use.

Stk file to protocol matching is handled by a user table, as described in User Table, with the following fields:

Match string

A partial match for an stk filename, the first match wins, so if you have a specific case and a general one the specific one must appear first in the list.

Protocol

This is the name of the encapsulating protocol (the lowest layer in the packet data) it can be either just the name of the protocol (e.g., mtp2, eth_withoutfcs, sscf-nni) or the name of the encapsulation protocol and the “application” protocol over it separated by a colon (e.g., sscop:sscf-nni, sscop:alcap, sscop:nbap, ...)

User DLTs protocol table

When a pcap file uses one of the user DLTs (147 to 162) Wireshark uses this table to know which protocol(s) to use for each user DLT.

This table is a user table, as described in User Table, with the following fields:

DLT

One of the user dlts.

Payload protocol

This is the name of the payload protocol (the lowest layer in the packet data). (e.g., “eth” for ethernet, “ip” for IPv4)

Header size

If there is a header protocol (before the payload protocol) this tells which size this header is. A value of 0 disables the header protocol.

Header protocol

The name of the header protocol to be used (uses “data” as default).

Trailer size

If there is a trailer protocol (after the payload protocol) this tells which size this trailer is. A value of 0 disables the trailer protocol.

Trailer protocol

The name of the trailer protocol to be used (uses “data” as default).

Protobuf Search Paths

The binary wire format of Protocol Buffers (Protobuf) messages are not self-described protocol. For example, the `varint` wire type in protobuf packet may be converted to int32, int64, uint32, uint64, sint32, sint64, bool or enum field types of protocol buffers language. Wireshark should be configured with Protocol Buffers language files (*.proto) to enable proper dissection of protobuf data (which may be payload of gRPC) based on the message, enum and field definitions.

You can specify protobuf search paths at the Protobuf protocol preferences. For example, if you defined a proto file with path `d:/my_proto_files/helloworld.proto` and the `helloworld.proto` contains a line of `import "google/protobuf/any.proto";` because the `any` type of official protobuf library is used. And the real path of `any.proto` is `d:/protobuf-3.4.1/include/google/protobuf/any.proto`. You should add the `d:/protobuf-3.4.1/include/` and `d:/my_proto_files` paths into protobuf search paths.

The configuration for the protobuf search paths is a user table, as described in User Table, with the following fields:

Protobuf source directory

This specifies a directory containing protobuf source files. For example, `d:/protobuf-3.4.1/include/` and `d:/my_proto_files` in Windows, or `/usr/include/` and `/home/alice/my_proto_files` in Linux/UNIX.

Load all files

If this option is enabled, Wireshark will load all *.proto files in this directory and its subdirectories when Wireshark startup or protobuf search paths preferences changed. Note that the source directories that configured to protobuf official or third libraries path (like `d:/protobuf-3.4.1/include/`) should not be set to load all files, that may cause unnecessary memory use.

Protobuf UDP Message Types

If the payload of UDP on certain ports is Protobuf encoding, Wireshark use this table to know which Protobuf message type should be used to parsing the data on the specified UDP port(s).

The configuration for UDP Port(s) to Protobuf message type maps is a user table, as described in User Table, with the following fields:

UDP Ports

The range of UDP ports. The format may be "8000" or "8000,8008-8088,9080".

Message Type

The Protobuf message type as which the data on the specified udp port(s) should be parsed. The message type is allowed to be empty, that means let Protobuf to dissect the data on specified UDP ports as normal wire type without precise definitions.

Tips: You can create your own dissector to call Protobuf dissector. If your dissector is written in C language, you can pass the message type to Protobuf dissector by `data` parameter of call_dissector_with_data() function. If your dissector is written in Lua, you can pass the message type to Protobuf dissector by `pinfo.private["pb_msg_type"]`. The format of `data` and `pinfo.private["pb_msg_type"]` is

```
"message," message_type_name
```

For example:

```
message,helloworld.HelloRequest
```

the `helloworld` is package name, `HelloRequest` is message type.

MATE

Introduction

MATE: Meta Analysis and Tracing Engine

What is MATE? Well, to keep it very short, with MATE you can create user configurable extension(s) of the display filter engine.

MATE's goal is to enable users to filter frames based on information extracted from related frames or information on how frames relate to each other. MATE was written to help troubleshooting gateways and other systems where a "use" involves more protocols. However, MATE can be used as well to analyze other issues regarding an interaction between packets like response times, incompleteness of transactions, presence/absence of certain attributes in a group of PDUs and more.

MATE is a Wireshark plugin that allows the user to specify how different frames are related to each other. To do so, MATE extracts data from the frames' tree and then, using that information, tries to group the frames based on how MATE is configured. Once the PDUs are related, MATE will create a "protocol" tree with fields the user can filter with. The fields will be almost the same for all the related frames, so one can filter a complete session spanning several frames containing more protocols based on an attribute appearing in some related frame. Other than that MATE allows to filter frames based on response times, number of PDUs in a group and a lot more.

So far MATE has been used to:

- Filter all packets of a call using various protocols knowing just the calling number. (MATE's original goal)
- Filter all packets of all calls using various protocols based on the release cause of one of its "segments".
- Extrapolate slow transactions from very "dense" captures. (finding requests that timeout)
- Find incomplete transactions (no responses)
- Follow requests through more gateways/proxies.
- more...

Getting Started

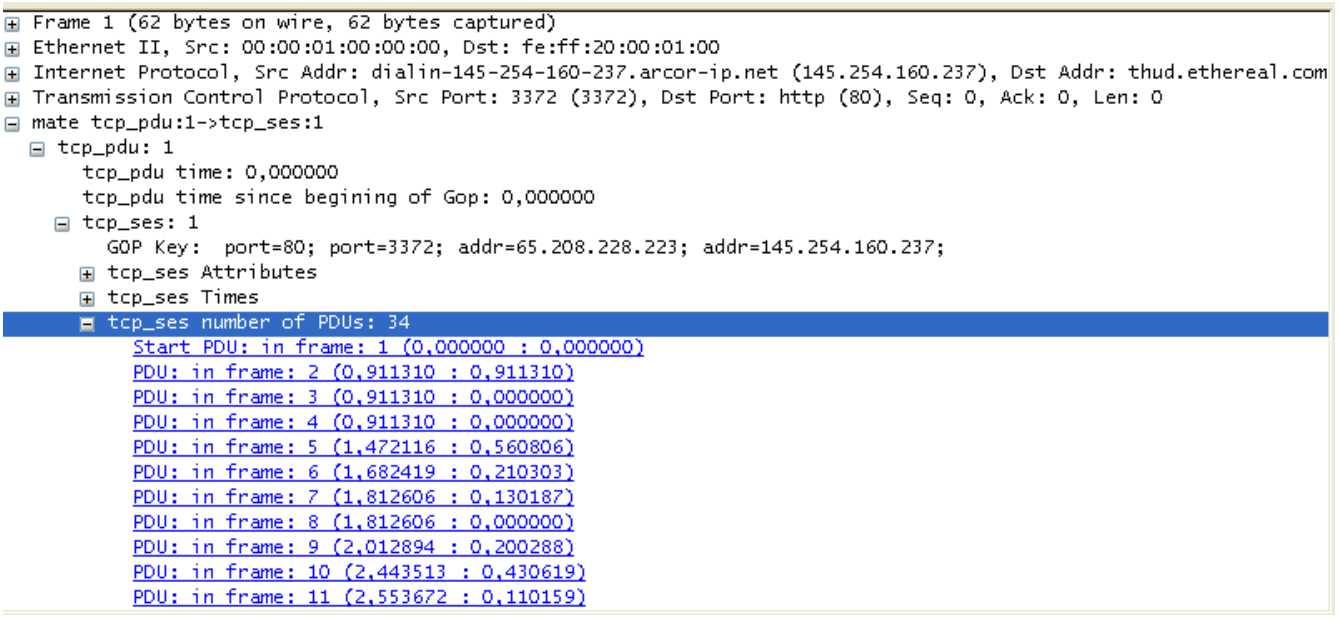
These are the steps to try out MATE:

- Run Wireshark and check if the plugin is installed correct (MATE should appear in Help→About→Plugins)
- Get a configuration file e.g., tcp.mate (see Mate/Examples for more) and place it somewhere on

your harddisk.

- Go to Preferences→Protocols→MATE and set the config filename to the file you want to use (you don't have to restart Wireshark)
- Load a corresponding capture file (e.g., http.cap) and see if MATE has added some new display filter fields, something like: `mate tcp_pdu:1→tcp_ses:1` or, at prompt: `path_to/wireshark -o "mate.config: tcp.mate" -r http.cap`.

If anything went well, your packet details might look something like this:



MATE Manual

Introduction

MATE creates a filterable tree based on information contained in frames that share some relationship with information obtained from other frames. The way these relationships are made is described in a configuration file. The configuration file tells MATE what makes a PDU and how to relate it to other PDUs.

MATE analyzes each frame to extract relevant information from the "protocol" tree of that frame. The extracted information is contained in MATE PDUs; these contain a list of relevant attributes taken from the tree. From now on, I will use the term "PDU" to refer to the objects created by MATE containing the relevant information extracted from the frame; I'll use "frame" to refer to the "raw" information extracted by the various dissectors that pre-analyzed the frame.

For every PDU, MATE checks if it belongs to an existing "Group of PDUs" (Gop). If it does, it assigns the PDU to that Gop and moves any new relevant attributes to the Gop's attribute list. How and when do PDUs belong to Gops is described in the configuration file as well.

Every time a Gop is assigned a new PDU, MATE will check if it matches the conditions to make it

belong to a "Group of Groups" (Gog). Naturally the conditions that make a Gop belong to a Gog are taken from the configuration file as well.

Once MATE is done analyzing the frame it will be able to create a "protocol" tree for each frame based on the PDUs, the Gops they belong to and naturally any Gogs the former belongs to.

How to tell MATE what to extract, how to group it and then how to relate those groups is made using AVPs and AVPLs.

Information in MATE is contained in Attribute/Value Pairs (AVPs). AVPs are made of two strings: the name and the value. AVPs are used in the configuration and there they have an operator as well. There are various ways AVPs can be matched against each other using those operators.

AVPs are grouped into AVP Lists (AVPLs). PDUs, Gops and Gogs have an AVPL each. Their AVPLs will be matched in various ways against others coming from the configuration file.

MATE will be instructed how to extract AVPs from frames in order to create a PDU with an AVPL. It will be instructed as well, how to match that AVPL against the AVPLs of other similar PDUs in order to relate them. In MATE the relationship between PDUs is a Gop, it has an AVPL as well. MATE will be configured with other AVPLs to operate against the Gop's AVPL to relate Gops together into Gogs.

A good understanding on how AVPs and AVPLs work is fundamental to understand how MATE works.

Attribute Value Pairs

Information used by MATE to relate different frames is contained in Attribute/ Value Pairs (AVPs). AVPs are made of two strings - the name and the value. When AVPs are used in the configuration, an operator is defined as well. There are various ways AVPs can be matched against each other using those operators.

```
avp_name="avp's value"
another_name= "1234 is the value"
```

The name is a string used to refer to a "kind" of an AVP. Two AVPs won't match unless their names are identical.

You should not use uppercase characters in names, or names that start with "." or "_". Capitalized names are reserved for configuration parameters (we'll call them keywords); nothing forbids you from using capitalized strings for other things as well but it probably would be confusing. I'll avoid using capitalized words for anything but the keywords in this document, the reference manual, the examples and the base library. Names that start with a "." would be very confusing as well because in the old grammar, AVPL transformations use names starting with a "." to indicate they belong to the replacement AVPL.

The value is a string that is either set in the configuration (for configuration AVPs) or by Wireshark

while extracting interesting fields from a frame's tree. The values extracted from fields use the same representation as they do in filter strings except that no quotes are used.

The name can contain only alphanumeric characters, "_", and ".". The name ends with an operator.

The value will be dealt with as a string even if it is a number. If there are any spaces in the value, the value must be between quotes "".

```
ip_addr=10.10.10.11,
tcp_port=1234,
binary_data=01:23:45:67:89:ab:cd:ef,
parameter12=0x23aa,
parameter_with_spaces="this value has spaces"
```

The way two AVPs with the same name might match is described by the operator. Remember two AVPs won't match unless their names are identical. In MATE, match operations are always made between the AVPs extracted from frames (called data AVPs) and the configuration's AVPs.

Currently defined MATE's AVP match operators are:

- Equal = will match if the string given completely matches the data AVP's value string
- Not Equal *!* will match only if the given value string is not equal to the data AVP's value string
- One Of *{}* will match if one of the possible strings listed is equal to the data AVP's value string
- Starts With ^ will match if the string given matches the first characters of the data AVP's value string
- Ends With *$* will match if the string given matches the last characters of the data AVP's value string
- Contains ~ will match if the string given matches any substring of the data AVP's value string
- Lower Than < will match if the data AVP's value string is semantically lower than the string given
- Higher Than > will match if the data AVP's value string is semantically higher than the string given
- Exists *?* (the ? can be omitted) will match as far as a data AVP of the given name exists

AVP lists

An AVPL is a set of diverse AVPs that can be matched against other AVPLs. Every PDU, Gop and Gog has an AVPL that contains the information regarding it. The rules that MATE uses to group Pdus and Gops are AVPL operations.

There will never be two identical AVPs in a given AVPL. However, we can have more than one AVP with the same name in an AVPL as long as their values are different.

Some AVPL examples:

```
  ( addr=10.20.30.40, addr=192.168.0.1, tcp_port=21, tcp_port=32534, user_cmd=PORT,
data_port=12344, data_addr=192.168.0.1 )
  ( addr=10.20.30.40, addr=192.168.0.1, channel_id=22:23, message_type=Setup,
calling_number=1244556673 )
  ( addr=10.20.30.40, addr=192.168.0.1, ses_id=01:23:45:67:89:ab:cd:ef )
  ( user_id=pippo, calling_number=1244556673, assigned_ip=10.23.22.123 )
```

In MATE there are two types of AVPLs:

- data AVPLs that contain information extracted from frames.
- operation AVPLs that come from the configuration and are used to tell MATE how to relate items based on their data AVPLs.

Data AVPLs can be operated against operation AVPLs in various ways:

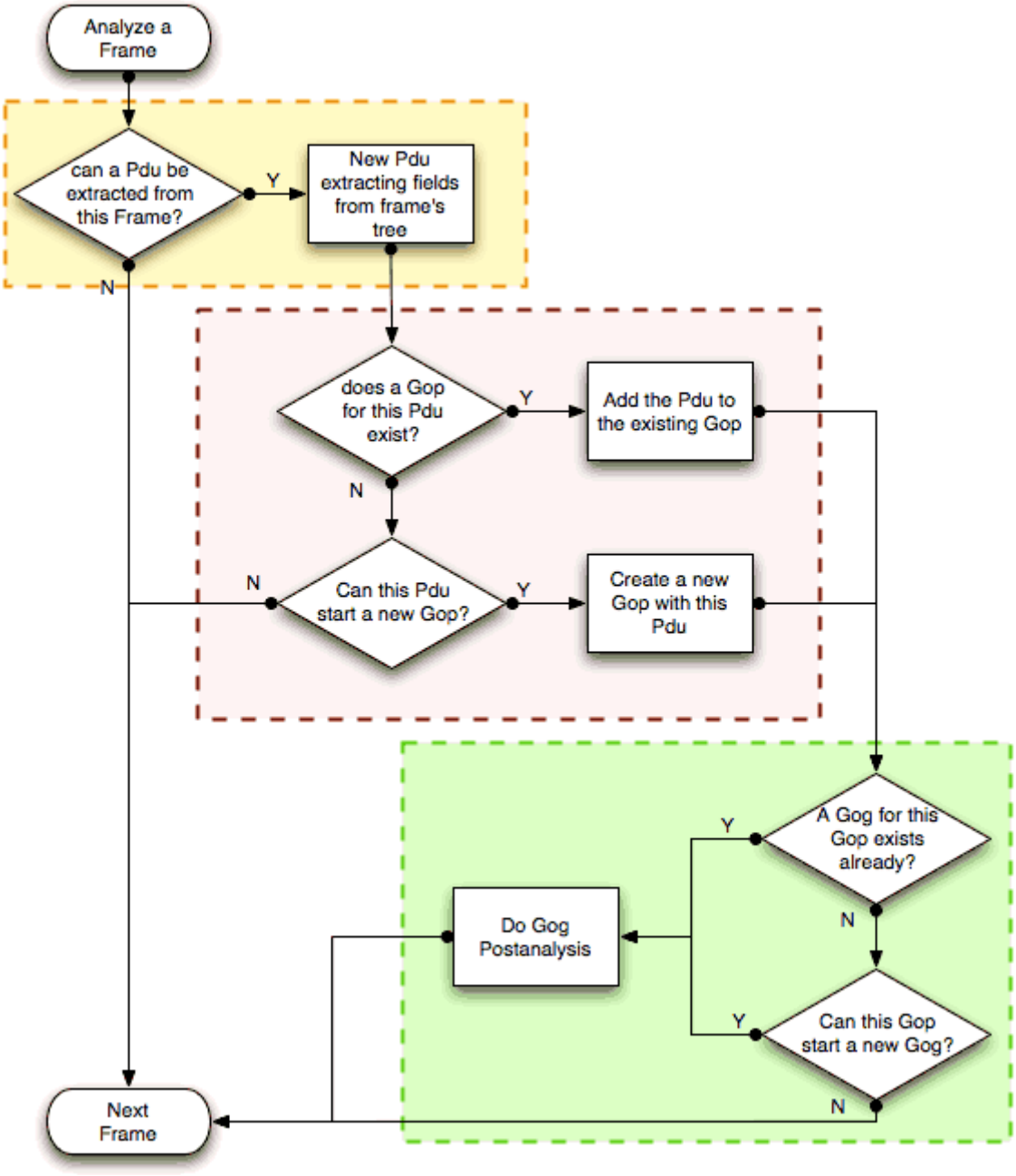
- Loose Match: Will match if at least one of the AVPs of each AVPL match. If it matches it will return an AVPL containing all AVPs from the operand AVPL that did match the operator's AVPs.
- "Every" Match: Will match if none of the AVPs of the operator AVPL fails to match a present AVP in the operand AVPL, even if not all of the operator's AVPs have a match. If it matches it will return an AVPL containing all AVPs from the operand AVPL that did match one AVP in the operator AVPL.
- Strict Match: Will match if and only if every one of the operator's AVPs have at least one match in the operand AVPL. If it matches it will return an AVPL containing the AVPs from the operand that matched.
- There's also a Merge operation that is to be performed between AVPLs where all the AVPs that don't exist in the operand AVPL but exist in the operand will be added to the operand AVPL.
- Other than that, there are Transformations - a combination of a match AVPL and an AVPL to merge.

MATE Analysis

MATE's analysis of a frame is performed in three phases:

- In the first phase, MATE attempts to extract a MATE Pdu from the frame's protocol tree. MATE will create a Pdu if MATE's config has a *Pdu* declaration whose *Proto* is contained in the frame.
- In the second phase, if a Pdu has been extracted from the frame, MATE will try to group it to other Pdus into a Gop (Group of Pdus) by matching the key criteria given by a *Gop* declaration. If there is no Gop yet with the key criteria for the Pdu, MATE will try to create a new Gop for it if it matches the *Start* criteria given in the Gop declaration.
- In the third phase, if there's a Gop for the Pdu, MATE will try to group this Gop with other Gops

into a Gog (Group of Groups) using the criteria given by the *Member* criteria of a Gog declaration.



The extraction and matching logic comes from MATE's configuration; MATE's configuration file is declared by the *mate.config* preference. By default it is an empty string which means: do not configure MATE.

The config file tells MATE what to look for in frames; How to make PDUs out of it; How will PDUs be related to other similar PDUs into Gops; And how Gops relate into Gogs.

The MATE configuration file is a list of declarations. There are 4 types of declarations: *Transform*, *Pdu*, *Gop* and *Gog*.

Mate's PDU's

MATE will look in the tree of every frame to see if there is useful data to extract, and if there is, it will create one or more PDU objects containing the useful information.

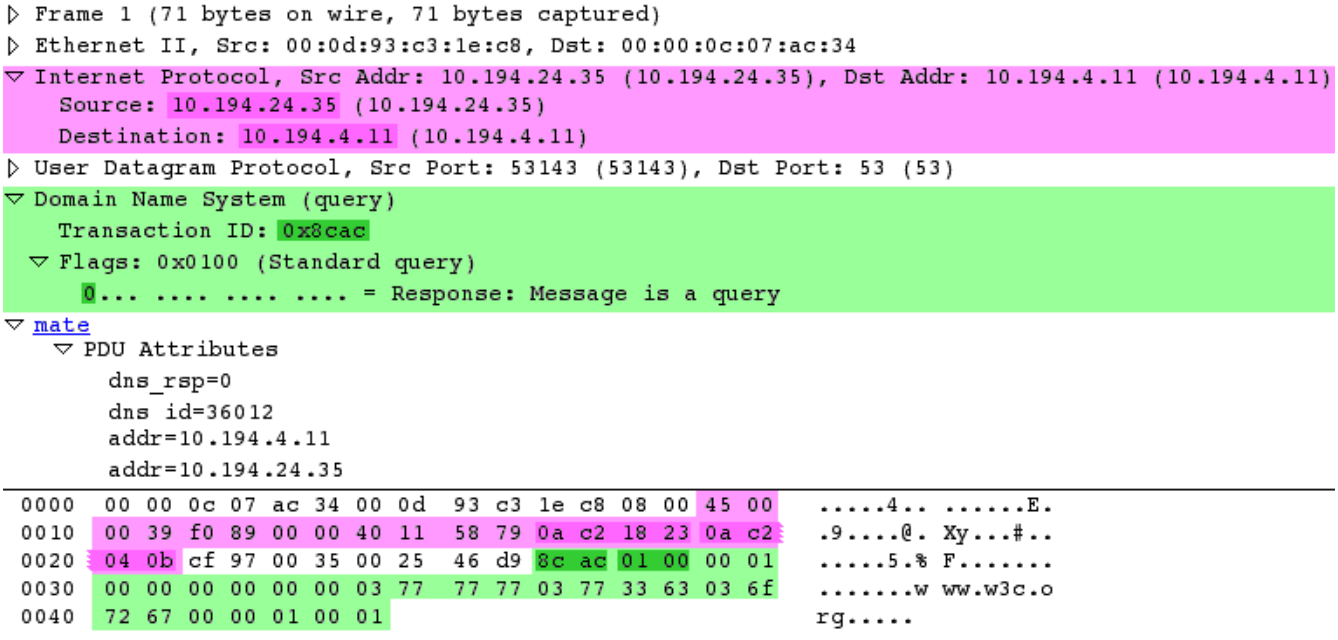
The first part of MATE's analysis is the "PDU extraction"; there are various "Actions" that are used to instruct MATE what has to be extracted from the current frame's tree into MATE's PDUs.

PDU data extraction

MATE will make a Pdu for each different proto field of Proto type present in the frame. MATE will fetch from the field's tree those fields that are defined in the Pdsu's configuration actions declaration whose initial offset in the frame is within the boundaries of the current Proto and those of the given Transport and Payload statements.

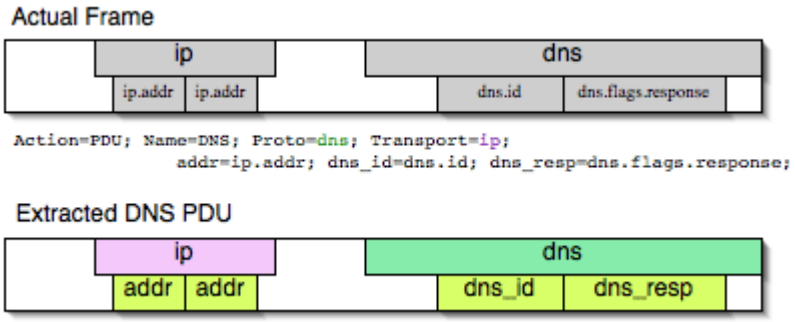
```
Pdu dns_pdu Proto dns Transport ip {
    Extract addr From ip.addr;
    Extract dns_id From dns.id;
    Extract dns_resp From dns.flags.response;
};
```

MATE will make a Pdu for each different proto field of Proto type present in the frame. MATE will fetch from the field's tree those fields that are defined in the Pdsu's configuration actions AVPL whose initial offset in the frame is within the boundaries of the current Proto and those of the various assigned Transports.



Once MATE has found a *Proto* field for which to create a Pdu from the frame it will move backwards in the frame looking for the respective *Transport* fields. After that it will create AVPs named as each of those given in the rest of the AVPL for every instance of the fields declared as its

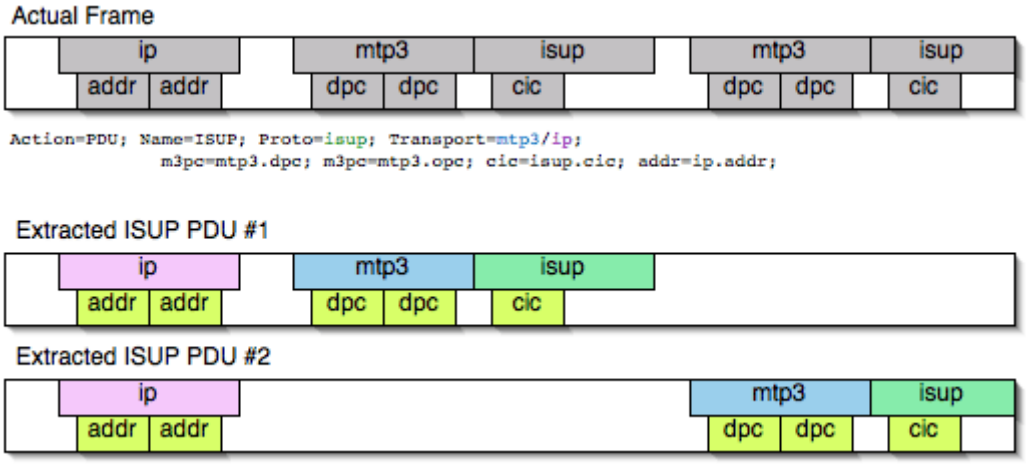
values.



Sometimes we need information from more than one *Transport* protocol. In that case MATE will check the frame looking backwards to look for the various *Transport* protocols in the given stack. MATE will choose only the closest transport boundary per "protocol" in the frame.

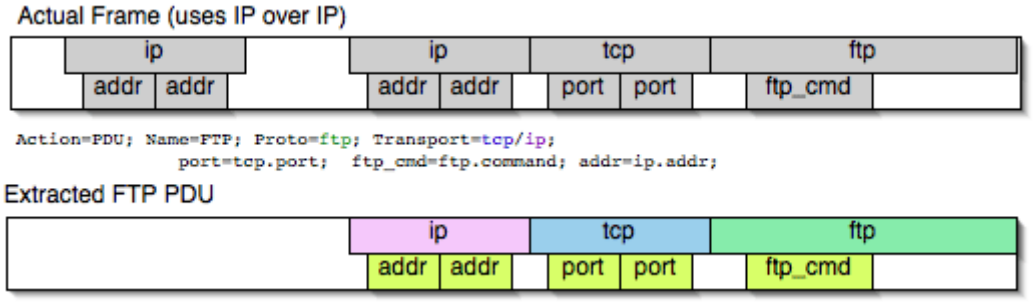
This way we'll have all Pdus for every *Proto* that appears in a frame match its relative transports.

```
Pdu isup_pdu Proto isup Transport mtp3/ip {
        Extract m3pc From mtp3.dpc;
        Extract m3pc From mtp3.opc;
        Extract cic From isup.cic;
        Extract addr From ip.addr;
        Extract isup_msg From isup.message_type;
};
```



This allows to assign the right *Transport* to the Pdu avoiding duplicate transport protocol entries (in case of tunneled ip over ip for example).

```
Pdu ftp_pdu Proto ftp Transport tcp/ip {
        Extract addr From ip.addr;
        Extract port From tcp.port;
        Extract ftp_cmd From ftp.command;
};
```

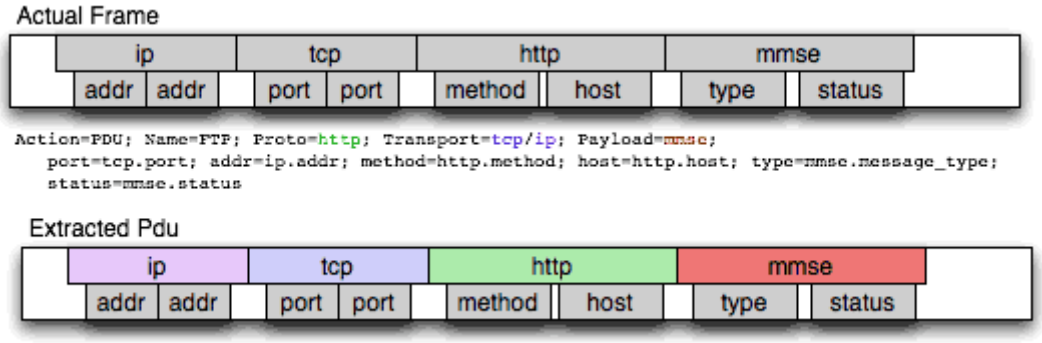


Other than the mandatory *Transport* there is also an optional *Payload* statement, which works pretty much as *Transport* but refers to elements after the *Proto*'s range. It is useful in those cases where the payload protocol might not appear in a Pdu but nevertheless the Pdu belongs to the same category.

```
Pdu mmse_over_http_pdu Proto http Transport tcp/ip {

        Payload mmse;

        Extract addr From ip.addr;
        Extract port From tcp.port;
        Extract method From http.request.method;
        Extract content From http.content_type;
        Extract http_rq From http.request;
        Extract resp From http.response.code;
        Extract host From http.host;
        Extract trx From mmse.transaction_id;
        Extract msg_type From mmse.message_type;
        Extract notify_status From mmse.status;
        Extract send_status From mmse.response_status;
};
```



Conditions on which to create PDUs

There might be cases in which we won't want MATE to create a PDU unless some of its extracted attributes meet or do not meet some criteria. For that we use the *Criteria* statements of the *Pdu* declarations.

```
Pdu isup_pdu Proto isup Transport mtp3/ip {
    ...

   // MATE will create isup_pdu PDUs only when there is not a point code '1234'
   Criteria Reject Strict (m3pc=1234);
};

Pdu ftp_pdu Proto ftp Transport tcp/ip {
    ...

    // MATE will create ftp_pdu PDUs only when they go to port 21 of our ftp_server
    Criteria Accept Strict (addr=10.10.10.10, port=21);
};
```

The *Criteria* statement is given an action (*Accept* or *Reject*), a match mode (*Strict*, *Loose* or *Every*) and an AVPL against which to match the currently extracted one.

Transforming the attributes of a PDU

Once the fields have been extracted into the Pdu's AVPL, MATE will apply any declared transformation to it. The way transforms are applied and how they work is described later on. However, it's useful to know that once the AVPL for the Pdu is created, it may be transformed before being analyzed. That way we can massage the data to simplify the analysis.

MATE's PDU tree

Every successfully created Pdu will add a MATE tree to the frame dissection. If the Pdu is not related to any Gop, the tree for the Pdu will contain just the Pdu's info, if it is assigned to a Gop, the tree will also contain the Gop items, and the same applies for the Gog level.

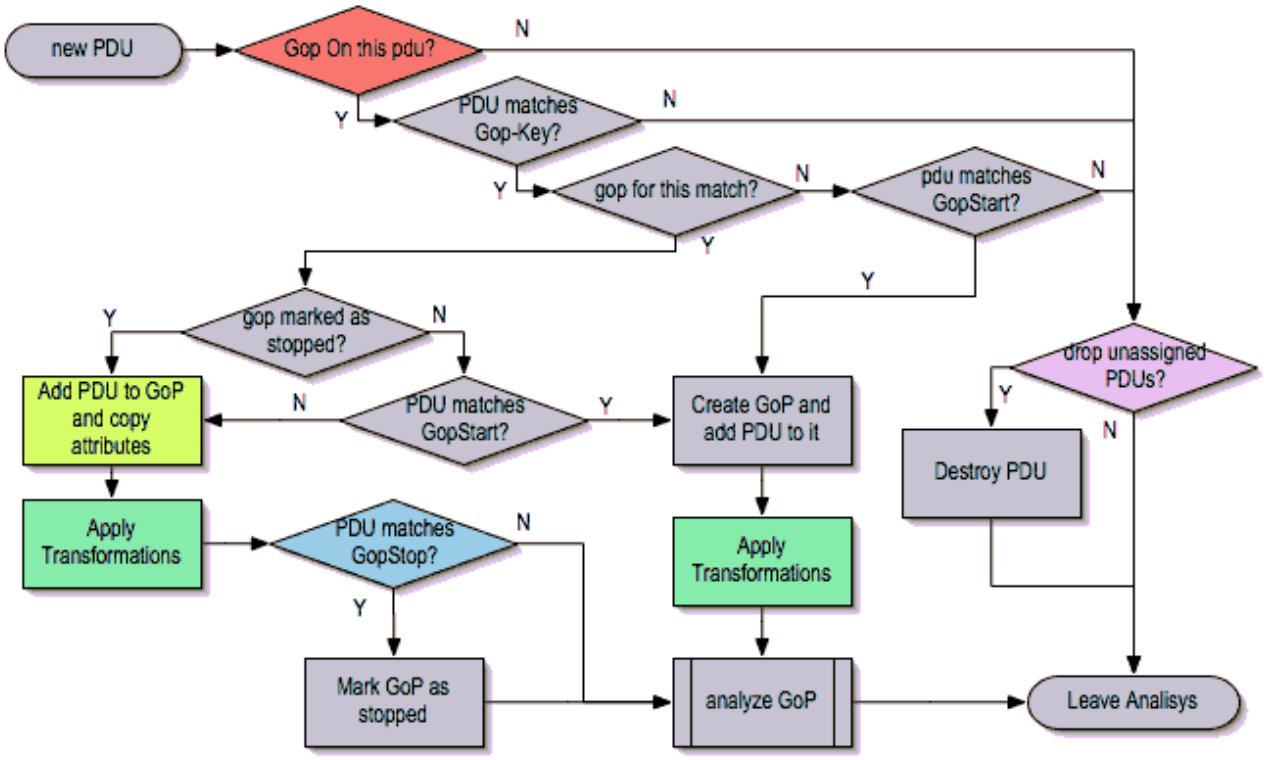
```
mate dns_pdu:1
    dns_pdu: 1
        dns_pdu time: 3.750000
        dns_pdu Attributes
            dns_resp: 0
            dns_id: 36012
            addr: 10.194.4.11
            addr: 10.194.24.35
```

The Pdu's tree contains some filterable fields

- *mate.dns_pdu* will contain the number of the "dns_pdu" Pdu
- *mate.dns_pdu.RelativeTime* will contain the time passed since the beginning of the capture in seconds
- the tree will contain the various attributes of the Pdu as well, these will all be strings (to be used in filters as "10.0.0.1", not as 10.0.0.1)
    - mate.dns_pdu.dns_resp
    - mate.dns_pdu.dns_id
    - mate.dns_pdu.addr

Grouping Pdus together (Gop)

Once MATE has created the Pdus it passes to the Pdu analysis phase. During the PDU analysis phase MATE will try to group Pdus of the same type into 'Groups of Pdus' (aka *Gop*s) and copy some AVPs from the Pdu's AVPL to the Gop's AVPL.



What can belong to a Gop

Given a Pdu, the first thing MATE will do is to check if there is any Gop declaration in the configuration for the given Pdu type. If so, it will use its *Match* AVPL to match it against the Pdu's AVPL; if they don't match, the analysis phase is done. If there is a match, the AVPL is the Gop's candidate key which will be used to search the Gop's index for the Gop to which to assign the current PDU. If there is no such Gop and this Pdu does not match the *Start* criteria of a Gop declaration for the Pdu type, the Pdu will remain unassigned and only the analysis phase will be done.

```
Gop ftp_ses On ftp_pdu Match (addr, addr, port, port);
Gop dns_req On dns_pdu Match (addr, addr, dns_id);
Gop isup_leg On isup_pdu Match (m3pc, m3pc, cic);
```

Start of a Gop

If there was a match, the candidate key will be used to search the Gop's index to see if there is already a Gop matching the Gop's key the same way. If there is such a match in the Gops collection, and the PDU doesn't match the *Start* AVPL for its kind, the PDU will be assigned to the matching Gop. If it is a *Start* match, MATE will check whether or not that Gop has been already stopped. If the Gop has been stopped, a new Gop will be created and will replace the old one in the Gop's index.

```
Gop ftp_ses On ftp_pdu Match (addr, addr, port, port) {
    Start (ftp_cmd=USER);
};

Gop dns_req On dns_pdu Match (addr, addr, dns_id) {
    Start (dns_resp=0);
};

Gop isup_leg On isup_pdu Match (m3pc, m3pc, cic) {
    Start (isup_msg=1);
};
```

If no *Start* is given for a Gop, a Pdu whose AVPL matches an existing Gog's key will act as the start of a Gop.

What goes into the Gop's AVPL

Once we know a Gop exists and the Pdu has been assigned to it, MATE will copy into the Gop's AVPL all the attributes matching the key plus any AVPs of the Pdu's AVPL matching the *Extra* AVPL.

```
Gop ftp_ses On ftp_pdu Match (addr, addr, port, port) {
    Start (ftp_cmd=USER);
    Extra (pasv_prt, pasv_addr);
};

Gop isup_leg On isup_pdu Match (m3pc, m3pc, cic) {
    Start (isup_msg=1);
    Extra (calling, called);
};
```

End of a Gop

Once the Pdu has been assigned to the Gop, MATE will check whether or not the Pdu matches the *Stop*, if it happens, MATE will mark the Gop as stopped. Even after stopped, a Gop may get assigned new Pdus matching its key, unless such Pdu matches *Start*. If it does, MATE will instead create a new Gop starting with that Pdu.

```
Gop ftp_ses On ftp_pdu Match (addr, addr, port, port) {
    Start (ftp_cmd=USER);
    Stop (ftp_cmd=QUIT); // The response to the QUIT command will be assigned to the
same Gop
    Extra (pasv_prt, pasv_addr);
};

Gop dns_req On dns_pdu Match (addr, addr, dns_id) {
    Start (dns_resp=0);
    Stop (dns_resp=1);
};

Gop isup_leg On isup_pdu Match (m3pc, m3pc, cic) {
    Start (isup_msg=1); // IAM
    Stop (isup_msg=16); // RLC
    Extra (calling, called);
};
```

If no *Stop* criterium is stated for a given Gop, the Gop will be stopped as soon as it is created. However, as with any other Gop, Pdus matching the Gop's key will still be assigned to the Gop unless they match a *Start* condition, in which case a new Gop using the same key will be created.

Gop's tree

For every frame containing a Pdu that belongs to a Gop, MATE will create a tree for that Gop.

The example below represents the tree created by the *dns_pdu* and *dns_req* examples.

```
...
mate dns_pdu:6->dns_req:1
    dns_pdu: 6
        dns_pdu time: 2.103063
        dns_pdu time since beginning of Gop: 2.103063
        dns_req: 1
            dns_req Attributes
                dns_id: 36012
                addr: 10.194.4.11
                addr: 10.194.24.35
            dns_req Times
                dns_req start time: 0.000000
                dns_req hold time: 2.103063
                dns_req duration: 2.103063
            dns_req number of PDUs: 2
                Start PDU: in frame 1
                Stop PDU: in frame 6 (2.103063 : 2.103063)
        dns_pdu Attributes
            dns_resp: 1
            dns_id: 36012
            addr: 10.194.4.11
            addr: 10.194.24.35
```

Other than the pdu's tree, this one contains information regarding the relationship between the Pdus that belong to the Gop. That way we have:

- mate.dns_req which contains the id of this dns_req Gop. This will be present in frames that belong to dns_req Gops.
- mate.dns_req.dns_id and mate.dns_req.addr which represent the values of the attributes copied into the Gop.
- the timers of the Gop
    - mate.dns_req.StartTime time (in seconds) passed since beginning of capture until Gop's start.
    - mate.dns_req.Time time passed between the start Pdu and the stop Pdu assigned to this Gop (only created if a Stop criterion has been declared for the Gop and a matching Pdu has arrived).
    - mate.dns_req.Duration time passed between the start Pdu and the last Pdu assigned to this Gop.
- mate.dns_req.NumOfPdus the number of Pdus that belong to this Gop
    - a filterable list of frame numbers of the pdus of this Gop

Gop's timers

Note that there are two "timers" for a Gop:

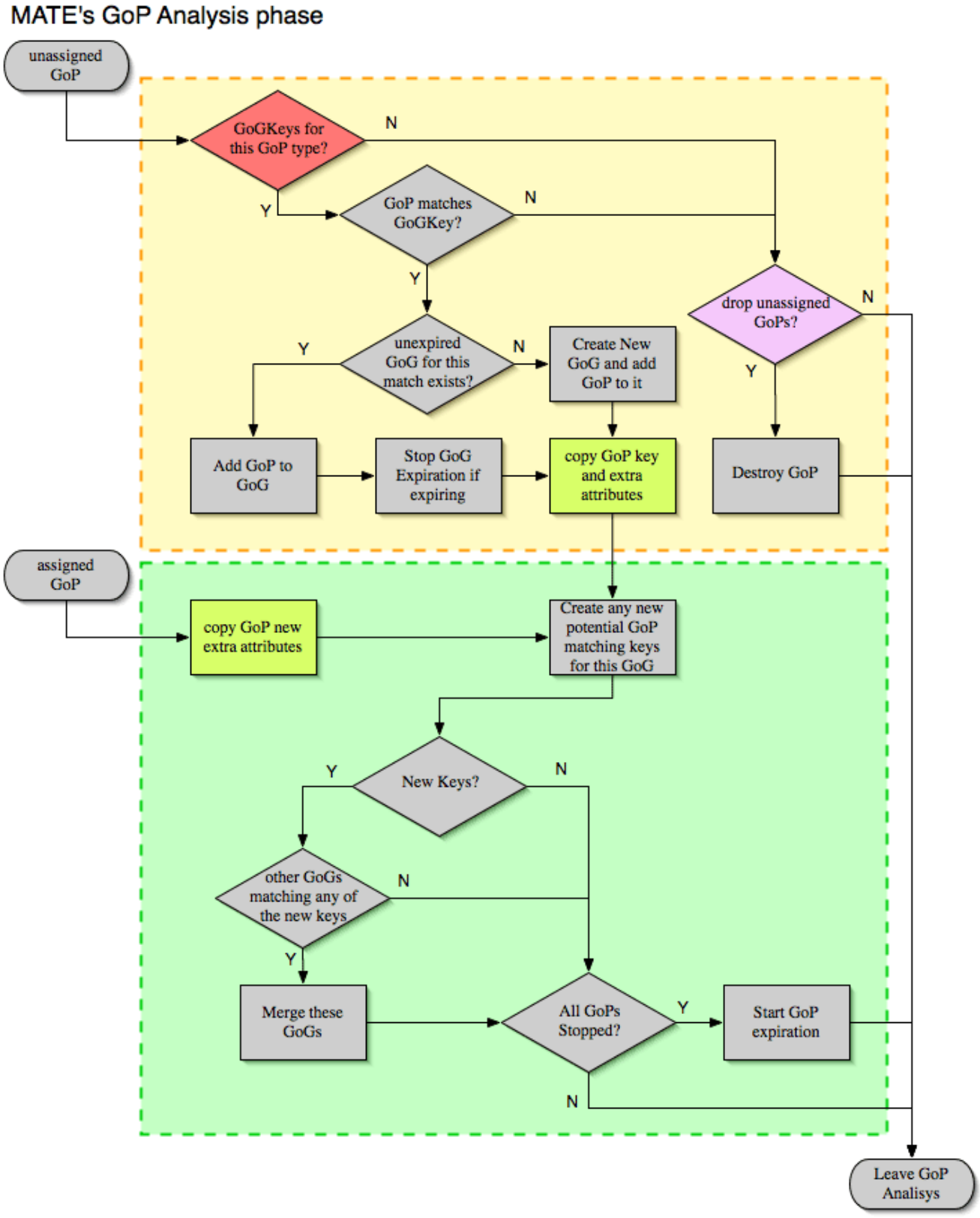
- **Time**, which is defined only for Gops that have been Stopped, and gives the time passed between the *Start* and the *Stop* Pdus.
- **Duration**, which is defined for every Gop regardless of its state, and give the time passed between its *Start* Pdu and the last Pdu that was assigned to that Gop.

So:

- we can filter for Pdus that belong to Gops that have been Stopped with **mate.xxx.Time**
- we can filter for Pdus that belong to unstopped Gops with **mate.xxx && mate.xxx.Time**
- we can filter for Pdus that belong to stopped Gops using **mate.xxx.Duration**
- we can filter for Pdus that belong to Gops that have taken more (or less) time that 0.5s to complete with **mate.xxx.Time > 0.5** (you can try these also as color filters to find out when response times start to grow)

Grouping Gops together (Gog)

When Gops are created, or whenever their AVPL changes, Gops are (re)analyzed to check if they match an existent group of groups (Gog) or can create a new one. The Gop analysis is divided into two phases. In the first phase, the still unassigned Gop is checked to verify whether it belongs to an already existing Gog or may create a new one. The second phase eventually checks the Gog and registers its keys in the Gogs index.



There are several reasons for the author to believe that this feature needs to be reimplemented, so probably there will be deep changes in the way this is done in the near future. This section of the documentation reflects the version of MATE as of Wireshark 0.10.9; in future releases this will change.

Declaring a Group Of Groups

The first thing we have to do configuring a Gog is to tell MATE that it exists.

```
Gog web_use {
   ...
};
```

Telling MATE what could be a Gog member

Then we have to tell MATE what to look for a match in the candidate Gops.

```
Gog web_use {
    Member http_ses (host);
    Member dns_req (host);
};
```

Getting interesting data into the Gop

Most often, also other attributes than those used for matching would be interesting. In order to copy from Gop to Gog other interesting attributes, we might use *Extra* like we do for Gops.

```
Gog web_use {
    ...
    Extra (cookie);
};
```

Gog's tree

```
mate http_pdu:4->http_req:2->http_use:1
    http_pdu: 4
        http_pdu time: 1.309847
        http_pdu time since beginning of Gop: 0.218930
        http_req: 2
            ... (the gop's tree for http_req: 2) ..
        http_use: 1
            http_use Attributes
                host: www.example.com
            http_use Times
                http_use start time: 0.000000
                http_use duration: 1.309847
            number of GOPs: 3
                dns_req: 1
                    ... (the gop's tree for dns_req: 1) ..
                http_req: 1
                    ... (the gop's tree for http_req: 1) ..
                http_req of current frame: 2
```

We can filter on:

- **mate.http_use.Duration** time elapsed between the first frame of a Gog and the last one assigned to it.
- the attributes passed to the Gog
  - **mate.http_use.host**

AVPL Transforms

A Transform is a sequence of Match rules optionally completed with modification of the match result by an additional AVPL. Such modification may be an Insert (merge) or a Replace. Transforms can be used as helpers to manipulate an item's AVPL before it is processed further. They come to be very helpful in several cases.

Syntax

AVPL Transformations are declared in the following way:

```
Transform name {
  Match [Strict|Every|Loose] match_avpl [Insert|Replace] modify_avpl ;
  ...
};
```

The **name** is the handle to the AVPL transformation. It is used to refer to the transform when invoking it later.

The *Match* declarations instruct MATE what and how to match against the data AVPL and how to modify the data AVPL if the match succeeds. They will be executed in the order they appear in the config file whenever they are invoked.

The optional match mode qualifier (*Strict*, *Every*, or *Loose*) is used to choose the match mode as explained above; *Strict* is a default value which may be omitted.

The optional modification mode qualifier instructs MATE how the modify AVPL should be used:

- the default value *Insert* (which may be omitted) causes the *modify_avpl* to be **merged** to the existing data AVPL,
- the *Replace* causes all the matching AVPs from the data AVPL to be **replaced** by the *modify_avpl*.

The *modify_avpl* may be an empty one; this comes useful in some cases for both *Insert* and *Replace* modification modes.

Examples:

```
 Transform insert_name_and {
   Match Strict (host=10.10.10.10, port=2345) Insert (name=JohnDoe);
};
```

adds name=JohnDoe to the data AVPL if it contains host=10.10.10.10 **and** port=2345

```
Transform insert_name_or {
   Match Loose (host=10.10.10.10, port=2345) Insert (name=JohnDoe);
};
```

adds name=JohnDoe to the data AVPL if it contains host=10.10.10.10 **or** port=2345

```
Transform replace_ip_address {
   Match (host=10.10.10.10) Replace (host=192.168.10.10);
};
```

replaces the original host=10.10.10.10 by host=192.168.10.10

```
Transform add_ip_address {
   Match (host=10.10.10.10) (host=192.168.10.10);
};
```

adds (inserts) host=192.168.10.10 to the AVPL, keeping the original host=10.10.10.10 in it too

```
Transform replace_may_be_surprising {
  Match Loose (a=aaaa, b=bbbb) Replace (c=cccc, d=dddd);
};
```

gives the following results:

- (a=aaaa, b=eeee) gets transformed to (b=eeee, c=cccc, d=dddd) because a=aaaa did match so it got replaced while b=eeee did not match so it has been left intact,
- (a=aaaa, b=bbbb) gets transformed to (c=cccc, d=dddd) because both a=aaaa and b=bbbb did match.

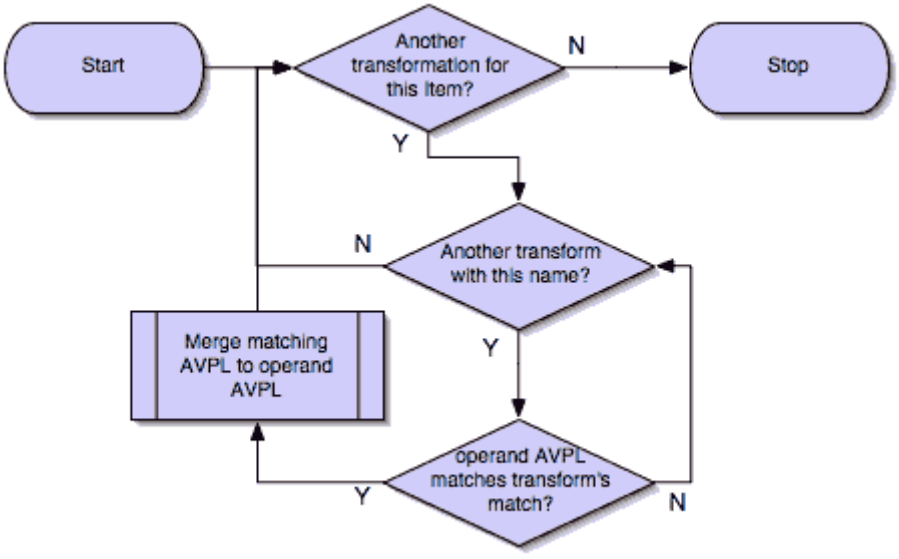
Usage

Once declared, Transforms can be added to the declarations of PDUs, Gops or Gogs. This is done by adding the *Transform name_list* statement to the declaration:

```
Pdu my_proto_pdu Proto my_proto Transport ip {
    Extract addr From ip.addr;
    ...
    Transform my_pdu_transform[, other_pdu_transform[, yet_another_pdu_transform]];
};
```

- In case of PDU, the list of transforms is applied against the PDU's AVPL after its creation.
- In case of Gop and Gog, the list of transforms is applied against their respective AVPLs when they are created and every time they change.

Operation



- A list of previously declared Transforms may be given to every Item (Pdu, Gop, or Gog), using the Transform statement.
- Every time the AVPL of an item changes, it will be operated against **all** the Transforms on the list given to that item. The Transforms on the list are applied left to right.
- Inside each of the Transforms, the item's AVPL will be operated against the Transform's Match clauses starting from the topmost one, until all have been tried or until one of them succeeds.

MATE's Transforms can be used for many different things, like:

Multiple Start/Stop conditions for a Gop

Using *Transforms* we can add more than one start or stop condition to a Gop.

```
Transform start_cond {
  Match (attr1=aaa,attr2=bbb) (msg_type=start);
  Match (attr3=www,attr2=bbb) (msg_type=start);
  Match (attr5^a) (msg_type=stop);
  Match (attr6$z) (msg_type=stop);
};

Pdu pdu ... {
  ...
  Transform start_cond;
}

Gop gop ... {
  Start (msg_type=start);
  Stop (msg_type=stop);
  ...
}
```

Marking Gops and Gogs to filter them easily

```
Transform marks {
  Match (addr=10.10.10.10, user=john) (john_at_host);
  Match (addr=10.10.10.10, user=tom) (tom_at_host);
}

...

Gop my_gop ... {
  ...
  Transform marks;
}
```

After that we can use a display filter **mate.gop.john_at_host** or **mate.gop.tom_at_host**

Adding direction knowledge to MATE

```
Transform direction_as_text {
  Match (src=192.168.0.2, dst=192.168.0.3) Replace (direction=from_2_to_3);
  Match (src=192.168.0.3, dst=192.168.0.2) Replace (direction=from_3_to_2);
};

Pdu my_pdu Proto my_proto Transport tcp/ip {
  Extract src From ip.src;
  Extract dst From ip.dst;
  Extract addr From ip.addr;
  Extract port From tcp.port;
  Extract start From tcp.flags.syn;
  Extract stop From tcp.flags.fin;
  Extract stop From tcp.flags.rst;
  Transform direction_as_text;
}

Gop my_gop On my_pdu Match (addr,addr,port,port) {
  ...
  Extra (direction);
}
```

NAT

NAT can create problems when tracing, but we can easily work around it by Transforming the NATed IP address and the Ethernet address of the router into the non-NAT address:

```
Transform denat {
  Match (addr=192.168.0.5, ether=01:02:03:04:05:06) Replace (addr=123.45.67.89);
  Match (addr=192.168.0.6, ether=01:02:03:04:05:06) Replace (addr=123.45.67.90);
  Match (addr=192.168.0.7, ether=01:02:03:04:05:06) Replace (addr=123.45.67.91);
}

Pdu my_pdu Proto my_proto transport tcp/ip/eth {
  Extract ether From eth.addr;
  Extract addr From ip.addr;
  Extract port From tcp.port;
  Transform denat;
}
```

About MATE

MATE was originally written by Luis Ontanon, a Telecommunications systems troubleshooter, as a way to save time filtering out the packets of a single call from huge capture files using just the calling number. Later he used the time he had saved to make it flexible enough to work with protocols other than the ones he was directly involved with.

MATE's configuration tutorial

We'll show a MATE configuration that first creates Gops for every DNS and HTTP request, then it ties the Gops together in a Gop based on the host. Finally, we'll separate into different Gogs request coming from different users.

With this MATE configuration loaded we can:

- use **mate.http_use.Duration** > **5.5** to filter frames based on the time it takes to load a complete page from the DNS request to resolve its name until the last image gets loaded.
- use **mate.http_use.client** == **"10.10.10.20" && mate.http_use.host** == **"www.example.com"** to isolate DNS and HTTP packets related to a visit of a certain user.
- use **mate.http_req.Duration** > **1.5** to filter all the packets of HTTP requests that take more than 1.5 seconds to complete.

The complete config file is available on the Wireshark Wiki: https://gitlab.com/wireshark/wireshark/-/wikis/Mate/Tutorial

Note: This example uses *dns.qry.name* which is defined since Wireshark version 0.10.9. Supposing you have a mate plugin already installed you can test it with the current Wireshark version.

A Gop for DNS requests

First we'll tell MATE how to create a Gop for each DNS request/response.

MATE needs to know what makes a DNS PDU. We describe it this using a Pdu declaration:

```
Pdu dns_pdu Proto dns Transport ip {
  Extract addr From ip.addr;
  Extract dns_id From dns.id;
  Extract dns_resp From dns.flags.response;
};
```

Using *Proto dns* we tell MATE to create Pdus every time it finds *dns*. Using *Transport ip* we inform MATE that some of the fields we are interested are in the *ip* part of the frame. Finally, we tell MATE to import *ip.addr* as *addr*, *dns.id* as *dns_id* and *dns.flags.response* as *dns_resp*.

Once we've told MATE how to extract *dns_pdus* we'll tell it how to match requests and responses

and group them into a Gop. For this we'll use a *Gop* declaration to define the Gop, and then, *Start* and *Stop* statements to tell it when the Gop starts and ends.

```
Gop dns_req On dns_pdu Match (addr,addr,dns_id) {
  Start (dns_resp=0);
  Stop (dns_resp=1);
};
```

Using the **Gop** declaration we tell MATE that the **Name** of the Gop is *dns_req*, that *dns_pdus* can become members of the Gop, and what is the key used to match the Pdus to the Gop.

The key for this Gop is *"addr, addr, dns_id"*. That means that in order to belong to the same Gop, *dns_pdus* have to have both addresses and the *request id* identical. We then instruct MATE that a *dns_req* starts whenever a *dns_pdu* matches *"dns_resp=0"* and that it stops when another *dns_pdu* matches *"dns_resp=1"*.

At this point, if we open a capture file using this configuration, we are able to use a display filter **mate.dns_req.Time > 1** to see only the packets of DNS requests that take more than one second to complete.

We can use a display filter **mate.dns_req && ! mate.dns_req.Time** to find requests for which no response was given. **mate.xxx.Time** is set only for Gops that have being stopped.

A Gop for HTTP requests

This other example creates a Gop for every HTTP request.

```
Pdu http_pdu Proto http Transport tcp/ip {
  Extract addr From ip.addr;
  Extract port From tcp.port;
  Extract http_rq From http.request.method;
  Extract http_rs From http.response;
  DiscardPduData true;
};

Gop http_req On http_pdu Match (addr, addr, port, port) {
  Start (http_rq);
  Stop (http_rs);
};
```

So, if we open a capture using this configuration

- filtering with **mate.http_req.Time > 1** will give all the requests where the response header takes more than one second to come

- filtering with **mate.http_req.Duration > 1.5** will show those request that take more than 1.5 seconds to complete.

You have to know that **mate.xxx.Time** gives the time in seconds between the pdu matching the GopStart and the Pdu matching the GopStop (yes, you can create timers using this!). On the other hand, **mate.xxx.Duration** gives you the time passed between the GopStart and the last pdu assigned to that Gop regardless whether it is a stop or not. After the GopStop, Pdus matching the Gop's Key will still be assigned to the same Gop as far as they don't match the GopStart, in which case a new Gop with the same key will be created.

Getting DNS and HTTP together into a Gog

We'll tie together to a single Gog all the http packets belonging to requests and responses to a certain host and the dns request and response used to resolve its domain name using the Pdu and Gop definitions of the previous examples

To be able to group DNS and HTTP requests together, we need to import into the Pdus and Gops some part of information that both those protocols share. Once the Pdus and Gops have been defined, we can use *Extract* (for Pdus) and *Extract* (for Gops) statements to tell MATE what other protocol fields are to be added to Pdus' and Gops' AVPLs. We add the following statements to the appropriate declarations:

```
Extract host From http.host; // to Pdu http_pdu as the last Extract in the list
Extra (host); // to Gop http_req after the Stop

Extract host From dns.qry.name; // to Pdu dns_pdu as the last Extract in the list
Extra (host); // to Gop dns_req after the Stop
```

Here we've told MATE to import *http.host* into *http_pdu* and *dns.qry.name* into *dns_pdu* as *host*. We also have to tell MATE to copy the *host* attribute from the Pdus to the Gops, we do this using *Extra*.

Once we have all the data we need in Pdus and Gops, we tell MATE what makes different Gops belong to a certain Gog.

```
Gog http_use {
  Member http_req (host);
  Member dns_req (host);
  Expiration 0.75;
};
```

Using the *Gog* declaration, we tell MATE to define a Gog type *Named http_use* whose expiration is 0.75 seconds after all the Gops that belong to it had been stopped. After that time, an eventual new Gop with the same key match will create a new Gog instead of been added to the previous Gog.

Using the *Member* statements, we tell MATE that **http_req*s with the same *host** belong to the same Gog, same thing for *dns_req*s.

So far we have instructed mate to group every packet related to sessions towards a certain host. At this point if we open a capture file and:

- a display filter **mate.http_use.Duration > 5** will show only those requests that have taken more than 5 seconds to complete starting from the DNS request and ending with the last packet of the http responses.
- a display filter **mate.http_use.host == "www.w3c.org"** will show all the packets (both DNS and HTTP) related to the requests directed to www.w3c.org

Separating requests from multiple users

"Houston: we've had a problem here."

This configuration works fine if used for captures taken at the client's side but deeper in the network we'd got a real mess. Requests from many users get mixed together into *http_uses*. Gogs are created and stopped almost randomly (depending on the timing in which Gops start and stop). How do we get requests from individual users separated from each other?

MATE has a tool that can be used to resolve this kind of grouping issues. This tool are the *Transforms*. Once defined, they can be applied against Pdus, Gops and Gogs and they might replace or insert more attributes based on what's there. We'll use them to create an attribute named client, using which we'll separate different requests.

For DNS we need the ip.src of the request moved into the Gop only from the DNS request.

So we first tell MATE to import ip.src as client:

```
Extract client From ip.src;
```

Next, we tell MATE to replace (**dns_resp=1, client**) with just **dns_resp=1** in the Pdu. That way, we'll keep the attribute **client** only in the DNS request Pdus (i.e., packets coming from the client).To do so, we have to add a *Transform* declaration (in this case, with just one clause) before the Pdu declaration which uses it:

```
Transform rm_client_from_dns_resp {
  Match (dns_resp=1, client) Replace (dns_resp=1);
};
```

Next, we invoke the transform by adding the following line after the *Extract* list of the dns_pdu Pdu:

```
  Transform rm_client_from_dns_resp;
```

HTTP is a little trickier. We have to remove the attribute carrying ip.src from both the response and the "continuations" of the response, but as there is nothing to filter on for the continuations, we have to add a fake attribute first. And then we have to remove client when the fake attribute appears. This is possible due to the fact that the *Match* clauses in the *Transform* are executed one by one until one of them succeeds. First, we declare another two *Transforms*:

```
Transform rm_client_from_http_resp1 {
  Match (http_rq); //first match wins so the request won't get the not_rq attribute
inserted
  Match Every (addr) Insert (not_rq); //this line won't be evaluated if the first one
matched so not_rq won't be inserted to requests
};

Transform rm_client_from_http_resp2 {
  Match (not_rq, client) Replace (); //replace "client and not_rq" with nothing (will
happen only in the response and eventual parts of it)
};
```

Next, we add another *Extract* statement to the *http_pdu* declaration, and apply both *Transforms* declared above in a proper order:

```
  Extract client From ip.src;
  Transform rm_client_from_http_resp1, rm_client_from_http_resp2;
```

In MATE, all the *Transform_s listed for an item will be evaluated, while inside a single _Transform*, the evaluation will stop at the first successful *Match* clause. That's why we first just match *http_rq* to get out of the first sequence before adding the *not_rq* attribute. Then we apply the second *Transform* which removes both *not_rq* and *client* if both are there. Yes, _Transform_s are cumbersome, but they are very useful.

Once we got all what we need in the Pdus, we have to tell MATE to copy the attribute *client* from the Pdus to the respective Gops, by adding client to *Extra* lists of both Gop declarations:

```
Extra (host, client);
```

On top of that, we need to modify the old declarations of Gop key to new ones that include both *client* and *host*. So we change the Gog **Member** declarations the following way:

```
  Member http_req (host, client);
  Member dns_req (host, client);
```

Now we got it, every "usage" gets its own Gog.

MATE configuration examples

The following is a collection of various configuration examples for MATE. Many of them are useless because the "conversations" facility does a better job. Anyway they are meant to help users understanding how to configure MATE.

TCP session

The following example creates a GoP out of every TCP session.

```
Pdu tcp_pdu Proto tcp Transport ip {
    Extract addr From ip.addr;
    Extract port From tcp.port;
    Extract tcp_start From tcp.flags.syn;
    Extract tcp_stop From tcp.flags.reset;
    Extract tcp_stop From tcp.flags.fin;
};

Gop tcp_ses On tcp_pdu Match (addr, addr, port, port) {
    Start (tcp_start=1);
    Stop (tcp_stop=1);
};

Done;
```

This probably would do fine in 99.9% of the cases but 10.0.0.1:20→10.0.0.2:22 and 10.0.0.1:22→10.0.0.2:20 would both fall into the same gop if they happen to overlap in time.

- filtering with **mate.tcp_ses.Time** > **1** will give all the sessions that last less than one second
- filtering with **mate.tcp_ses.NumOfPdus** < **5** will show all tcp sessions that have less than 5 packets.
- filtering with **mate.tcp_ses.Id** == **3** will show all the packets for the third tcp session MATE has found

a Gog for a complete FTP session

This configuration allows to tie a complete passive ftp session (including the data transfer) in a single Gog.

```
Pdu ftp_pdu Proto ftp Transport tcp/ip {
        Extract ftp_addr From ip.addr;
        Extract ftp_port From tcp.port;
        Extract ftp_resp From ftp.response.code;
        Extract ftp_req From ftp.request.command;
        Extract server_addr From ftp.passive.ip;
        Extract server_port From ftp.passive.port;

        LastPdu;
};

Pdu ftp_data_pdu Proto ftp-data Transport tcp/ip{
        Extract server_addr From ip.src;
        Extract server_port From tcp.srcport;

};

Gop ftp_data On ftp_data_pdu (server_addr, server_port) {
        Start (server_addr);
};

Gop ftp_ctl On ftp_pdu (ftp_addr, ftp_addr, ftp_port, ftp_port) {
        Start (ftp_resp=220);
        Stop (ftp_resp=221);
        Extra (server_addr, server_port);
};

Gog ftp_ses {
        Member ftp_ctl (ftp_addr, ftp_addr, ftp_port, ftp_port);
        Member ftp_data (server_addr, server_port);
};

Done;
```

Note: not having anything to distinguish between ftp-data packets makes this config to create one Gop for every ftp-data packet instead of each transfer. Pre-started Gops would avoid this.

using RADIUS to filter SMTP traffic of a specific user

Spying on people, in addition to being immoral, is illegal in many countries. This is an example meant to explain how to do it not an invitation to do so. It's up to the police to do this kind of job when there is a good reason to do so.

```
Pdu radius_pdu On radius Transport udp/ip {
    Extract addr From ip.addr;
    Extract port From udp.port;
    Extract radius_id From radius.id;
    Extract radius_code From radius.code;
    Extract user_ip From radius.framed_addr;
    Extract username From radius.username;
}

Gop radius_req On radius_pdu (radius_id, addr, addr, port, port) {
    Start (radius_code {1|4|7} );
    Stop (radius_code {2|3|5|8|9} );
    Extra (user_ip, username);
}

// we define the smtp traffic we want to filter
Pdu user_smtp Proto smtp Transport tcp/ip {
    Extract user_ip From ip.addr;
    Extract smtp_port From tcp.port;
    Extract tcp_start From tcp.flags.syn;
    Extract tcp_stop From tcp.flags.reset;
}

Gop user_smtp_ses On user_smtp (user_ip, user_ip, smtp_port!25) {
    Start (tcp_start=1);
    Stop (tcp_stop=1);
}

// with the following group of groups we'll group together the radius and the smtp
// we set a long expiration to avoid the session expire on long pauses.
Gog user_mail {
    Expiration 1800;
    Member radius_req (user_ip);
    Member user_smtp_ses (user_ip);
    Extra (username);
}

Done;
```

Filtering the capture file with **mate.user_mail.username** == **"theuser"** will filter the radius packets and smtp traffic for *"theuser"*.

H323 Calls

This configuration will create a Gog out of every call.

```
Pdu q931 Proto q931 Transport ip {
        Extract addr From ip.addr;
        Extract call_ref From q931.call_ref;
        Extract q931_msg From q931.message_type;
        Extract calling From q931.calling_party_number.digits;
        Extract called From q931.called_party_number.digits;
        Extract guid From h225.guid;
        Extract q931_cause From q931.cause_value;
};

Gop q931_leg On q931 Match (addr, addr, call_ref) {
        Start (q931_msg=5);
        Stop (q931_msg=90);
        Extra (calling, called, guid, q931_cause);
};

Pdu ras Proto h225.RasMessage Transport ip {
        Extract addr From ip.addr;
        Extract ras_sn From h225.requestSeqNum;
        Extract ras_msg From h225.RasMessage;
        Extract guid From h225.guid;
};

Gop ras_req On ras Match (addr, addr, ras_sn) {
        Start (ras_msg {0|3|6|9|12|15|18|21|26|30} );
        Stop (ras_msg {1|2|4|5|7|8|10|11|13|14|16|17|19|20|22|24|27|28|29|31});
        Extra (guid);
};

Gog call {
        Member ras_req (guid);
        Member q931_leg (guid);
        Extra (called,calling,q931_cause);
};

Done;
```

with this we can:

- filter all signalling for a specific caller: **mate.call.caller** == **"123456789"**
- filter all signalling for calls with a specific release cause: **mate.call.q931_cause** == **31**
- filter all signalling for very short calls: **mate.q931_leg.Time** < **5**

MMS

With this example, all the components of an MMS send or receive will be tied into a single Gog. Note that this example uses the *Payload* clause because MMS delivery uses MMSE over either HTTP or WSP. As it is not possible to relate the retrieve request to a response by the means of MMSE only (the request is just an HTTP GET without any MMSE), a Gop is made of HTTP Pdus but MMSE data need to be extracted from the bodies.

```
## WARNING: this example has been blindly translated from the "old" MATE syntax
## and it has been verified that Wireshark accepts it. However, it has not been
## tested against any capture file due to lack of the latter.

Transform rm_client_from_http_resp1 {
        Match (http_rq);
        Match Every (addr) Insert (not_rq);
};

Transform rm_client_from_http_resp2 {
        Match (not_rq,ue) Replace ();
};

Pdu mmse_over_http_pdu Proto http Transport tcp/ip {
        Payload mmse;
        Extract addr From ip.addr;
        Extract port From tcp.port;
        Extract http_rq From http.request;
        Extract content From http.content_type;
        Extract resp From http.response.code;
        Extract method From http.request.method;
        Extract host From http.host;
        Extract content From http.content_type;
        Extract trx From mmse.transaction_id;
        Extract msg_type From mmse.message_type;
        Extract notify_status From mmse.status;
        Extract send_status From mmse.response_status;
        Transform rm_client_from_http_resp1, rm_client_from_http_resp2;
};

Gop mmse_over_http On mmse_over_http_pdu Match (addr, addr, port, port) {
        Start (http_rq);
        Stop (http_rs);
        Extra (host, ue, resp, notify_status, send_status, trx);
};

Transform mms_start {
        Match Loose() Insert (mms_start);
};
```

```
Pdu mmse_over_wsp_pdu Proto wsp Transport ip {
        Payload mmse;
        Extract trx From mmse.transaction_id;
        Extract msg_type From mmse.message_type;
        Extract notify_status From mmse.status;
        Extract send_status From mmse.response_status;
        Transform mms_start;
};

Gop mmse_over_wsp On mmse_over_wsp_pdu Match (trx) {
        Start (mms_start);
        Stop (never);
        Extra (ue, notify_status, send_status);
};

Gog mms {
        Member mmse_over_http (trx);
        Member mmse_over_wsp (trx);
        Extra (ue, notify_status, send_status, resp, host, trx);
        Expiration 60.0;
};
```

MATE's configuration library

The MATE library (will) contains GoP definitions for several protocols. Library protocols are included in your MATE config using: _Action=Include; Lib=proto_name;_.

For Every protocol with a library entry, we'll find defined what from the PDU is needed to create a GoP for that protocol, eventually any criteria and the very essential GoP definition (i.e., *GopDef*, *GopStart* and *GopStop*).

NOTE	It seems that this code is written in the old syntax of MATE. So far it has not been transcribed into the new format. It may still form the basis to recreate these in the new format.

General use protocols

TCP

It will create a GoP for every TCP session, If it is used it should be the last one in the list. And every other proto on top of TCP should be declared with *Stop=TRUE;* so the a TCP PDU is not created where we got already one going on.

```
   Action=PduDef; Name=tcp_pdu; Proto=tcp; Transport=ip; addr=ip.addr; port=tcp.port;
tcp_start=tcp.flags.syn; tcp_stop=tcp.flags.fin; tcp_stop=tcp.flags.reset;
   Action=GopDef; Name=tcp_session; On=tcp_pdu; addr; addr; port; port;
   Action=GopStart; For=tcp_session; tcp_start=1;
   Action=GopStop; For=tcp_session; tcp_stop=1;
```

DNS

will create a GoP containing every request and its response (eventually retransmissions too).

```
Action=PduDef; Name=dns_pdu; Proto=dns; Transport=udp/ip; addr=ip.addr; port=udp.port;
dns_id=dns.id; dns_rsp=dns.flags.response;

Action=GopDef; Name=dns_req; On=dns_pdu; addr; addr; port!53; dns_id;
Action=GopStart; For=dns_req; dns_rsp=0;
Action=GopStop; For=dns_req; dns_rsp=1;
```

RADIUS

A Gop for every transaction.

```
Action=PduDef; Name=radius_pdu; Proto=radius; Transport=udp/ip; addr=ip.addr;
port=udp.port; radius_id=radius.id; radius_code=radius.code;

Action=GopDef; Name=radius_req; On=radius_pdu; radius_id; addr; addr; port; port;
Action=GopStart; For=radius_req; radius_code|1|4|7;
Action=GopStop; For=radius_req; radius_code|2|3|5|8|9;
```

RTSP

```
Action=PduDef; Name=rtsp_pdu; Proto=rtsp; Transport=tcp/ip; addr=ip.addr;
port=tcp.port; rtsp_method=rtsp.method;
Action=PduExtra; For=rtsp_pdu; rtsp_ses=rtsp.session; rtsp_url=rtsp.url;

Action=GopDef; Name=rtsp_ses; On=rtsp_pdu; addr; addr; port; port;
Action=GopStart; For=rtsp_ses; rtsp_method=DESCRIBE;
Action=GopStop; For=rtsp_ses; rtsp_method=TEARDOWN;
Action=GopExtra; For=rtsp_ses; rtsp_ses; rtsp_url;
```

VoIP/Telephony

Most protocol definitions here will create one Gop for every Call Leg unless stated.

ISUP

```
Action=PduDef; Name=isup_pdu; Proto=isup; Transport=mtp3; mtp3pc=mtp3.dpc;
mtp3pc=mtp3.opc; cic=isup.cic; isup_msg=isup.message_type;

Action=GopDef; Name=isup_leg; On=isup_pdu; ShowPduTree=TRUE; mtp3pc; mtp3pc; cic;
Action=GopStart; For=isup_leg; isup_msg=1;
Action=GopStop; For=isup_leg; isup_msg=16;
```

Q931

```
Action=PduDef; Name=q931_pdu; Proto=q931; Stop=TRUE; Transport=tcp/ip; addr=ip.addr;
call_ref=q931.call_ref; q931_msg=q931.message_type;

Action=GopDef; Name=q931_leg; On=q931_pdu; addr; addr; call_ref;
Action=GopStart; For=q931_leg; q931_msg=5;
Action=GopStop; For=q931_leg; q931_msg=90;
```

H225 RAS

```
Action=PduDef; Name=ras_pdu; Proto=h225.RasMessage; Transport=udp/ip; addr=ip.addr;
ras_sn=h225.RequestSeqNum; ras_msg=h225.RasMessage;
Action=PduExtra; For=ras_pdu; guid=h225.guid;

Action=GopDef; Name=ras_leg; On=ras_pdu; addr; addr; ras_sn;
Action=GopStart; For=ras_leg; ras_msg|0|3|6|9|12|15|18|21|26|30;
Action=GopStop; For=ras_leg;
ras_msg|1|2|4|5|7|8|10|11|13|14|16|17|19|20|22|24|27|28|29|31;
Action=GopExtra; For=ras_leg; guid;
```

SIP

```
Action=PduDef; Proto=sip_pdu; Transport=tcp/ip; addr=ip.addr; port=tcp.port;
sip_method=sip.Method; sip_callid=sip.Call-ID; calling=sdp.owner.username;

Action=GopDef; Name=sip_leg; On=sip_pdu; addr; addr; port; port;
Action=GopStart; For=sip; sip_method=INVITE;
Action=GopStop; For=sip; sip_method=BYE;
```

MEGACO

Will create a Gop out of every transaction.

To "tie" them to your call's GoG use: *Action=GogKey; Name=your_call; On=mgc_tr; addr!mgc_addr; megaco_ctx;*

```
Action=PduDef; Name=mgc_pdu; Proto=megaco; Transport=ip; addr=ip.addr;
megaco_ctx=megaco.context; megaco_trx=megaco.transid; megaco_msg=megaco.transaction;
term=megaco.termid;

Action=GopDef; Name=mgc_tr; On=mgc_pdu; addr; addr; megaco_trx;
Action=GopStart; For=mgc_tr; megaco_msg|Request|Notify;
Action=GopStop; For=mgc_tr; megaco_msg=Reply;
Action=GopExtra; For=mgc_tr; term^DS1; megaco_ctx!Choose one;
```

MATE's reference manual

Attribute Value Pairs

MATE uses AVPs for almost everything: to keep the data it has extracted from the frames' trees as well as to keep the elements of the configuration.

These "pairs" (actually tuples) are made of a name, a value and, in case of configuration AVPs, an operator. Names and values are strings. AVPs with operators other than '=' are used only in the configuration and are used for matching AVPs of Pdus, GoPs and GoGs in the analysis phase.

Name

The name is a string used to refer to a class of AVPs. Two attributes won't match unless their names are identical. Capitalized names are reserved for keywords (you can use them for your elements if you want but I think it's not the case). MATE attribute names can be used in Wireshark's display filters the same way like names of protocol fields provided by dissectors, but they are not just references to (or aliases of) protocol fields.

Value

The value is a string. It is either set in the configuration (for configuration AVPs) or by MATE while extracting interesting fields from a dissection tree and/or manipulating them later. The values extracted from fields use the same representation as they do in filter strings.

Operators

Currently only match operators are defined (there are plans to (re)add transform attributes but some internal issues have to be solved before that). The match operations are always performed between two operands: the value of an AVP stated in the configuration and the value of an AVP (or several AVPs with the same name) extracted from packet data (called "data AVPs"). It is not possible to match data AVPs to each other.

The defined match operators are:

- Equal = test for equality, that is: either the value strings are identical or the match will fail.
- Not Equal *!* will match only if the value strings aren't equal.
- One Of *{}* will match if one of the value strings listed is equal to the data AVP's string. Items inside the list's curly braces are separated with the | character.
- Starts With ^ will match if the configuration value string matches the first characters of the data AVP's value string.
- Ends With *$* will match if the configuration value string matches the last characters of the data AVP's value string.
- Contains ~ will match if the configuration value string matches a substring of the characters of the data AVP's value string.
- Lower Than < will match if the data AVP's value string is semantically lower than the configuration value string.
- Higher Than > will match if the data AVP's value string is semantically higher than the configuration value string.
- Exists *?* (can be omitted) will match if the AVP name matches, regardless what the value string is.

Equal AVP Operator

This operator tests whether the values of the operator and the operand AVP are equal.

Example

attrib=aaa **matches** attrib=aaa
attrib=aaa **does not match** attrib=bbb

Not equal AVP operator

This operator matches if the value strings of two AVPs are not equal.

Example

attrib=aaa matches attrib!bbb
attrib=aaa does not match attrib!aaa

"One of" AVP operator

The "one of" operator matches if the data AVP value is equal to one of the values listed in the "one of" AVP.

Example

attrib=1 matches attrib{1|2|3}
attrib=2 matches attrib{1|2|3}

attrib=4 does not match attrib{1|2|3}

"Starts with" AVP operator

The "starts with" operator matches if the first characters of the data AVP value are identical to the configuration AVP value.

Example

attrib=abcd matches attrib^abc
attrib=abc matches attrib^abc
attrib=ab does not match attrib^abc
attrib=abcd does not match attrib^bcd
attrib=abc does not match attrib^abcd

"Ends with" operator

The ends with operator will match if the last bytes of the data AVP value are equal to the configuration AVP value.

Example

attrib=wxyz matches attrib$xyz
attrib=yz does not match attrib$xyz
attrib=abc...wxyz does not match attrib$abc

Contains operator

The "contains" operator will match if the data AVP value contains a string identical to the configuration AVP value.

Example

attrib=abcde matches attrib~bcd
attrib=abcde matches attrib~abc
attrib=abcde matches attrib~cde
attrib=abcde does not match attrib~xyz

"Lower than" operator

The "lower than" operator will match if the data AVP value is semantically lower than the configuration AVP value.

Example

attrib=abc matches attrib<bcd
attrib=1 matches attrib<2
but beware: attrib=10 does not match attrib<9
attrib=bcd does not match attrib<abc
attrib=bcd does not match attrib<bcd

BUGS

It should check whether the values are numbers and compare them numerically

"Higher than" operator

The "higher than" operator will match if the data AVP value is semantically higher than the configuration AVP value.

Examples

attrib=bcd matches attrib>abc
attrib=3 matches attrib>2
but beware: attrib=9 does not match attrib>10
attrib=abc does not match attrib>bcd
attrib=abc does not match attrib>abc

BUGS

It should check whether the values are numbers and compare them numerically

Exists operator

The exists operator will always match as far as the two operands have the same name.

Examples

attrib=abc matches attrib?
attrib=abc matches attrib (this is just an alternative notation of the previous example)
obviously attrib=abc does not match other_attrib?

Attribute/Value Pair List (AVPL)

Pdus, GoPs and GoGs use an AVPL to contain the tracing information. An AVPL is an unsorted set of AVPs that can be matched against other AVPLs.

Operations between AVPLs

There are three types of match operations that can be performed between AVPLs. The Pdu's/GoP's/GoG's AVPL will be always one of the operands; the AVPL operator (match type) and the second operand AVPL will always come from the configuration. Note that a diverse AVP match operator may be specified for each AVP in the configuration AVPL.

An AVPL match operation returns a result AVPL. In Transforms, the result AVPL may be replaced by another AVPL. The replacement means that the existing data AVPs are dropped and the replacement AVPL from the configuration is Merged to the data AVPL of the Pdu/GoP/GoG.

- Loose Match: Will match if at least one of the AVPs of the two operand AVPLs match. If it

matches, it returns a result AVPL containing all AVPs from the data AVPL that did match the configuration's AVPs.

- "Every" Match: Will match if none of the AVPs of the configuration AVPL fails to match an AVP in the data AVPL, even if not all of the configuration AVPs have a match. If it matches, it returns a result AVPL containing all AVPs from the data AVPL that did match an AVP in the configuration AVPL.
- Strict Match: Will match if and only if each of the AVPs in the configuration AVPL has at least one match in the data AVPL. If it matches, it returns a result AVPL containing those AVPs from the data AVPL that matched.

Loose Match

A loose match between AVPLs succeeds if at least one of the data AVPs matches at least one of the configuration AVPs. Its result AVPL contains all the data AVPs that matched.

Loose matches are used in Extra operations against the Pdu's AVPL to merge the result into Gop's AVPL, and against Gop's AVPL to merge the result into Gog's AVPL. They may also be used in Criteria and Transforms.

NOTE	As of current (2.0.1), Loose Match does not work as described here, see issue 12184. Only use in Transforms and Criteria is effectively affected by the bug.

Loose Match Examples

(attr_a=aaa, attr_b=bbb, attr_c=xxx) Match Loose (attr_a?, attr_c?) ⇒ (attr_a=aaa, attr_c=xxx)

(attr_a=aaa, attr_b=bbb, attr_c=xxx) Match Loose (attr_a?, attr_c=ccc) ⇒ (attr_a=aaa)

(attr_a=aaa, attr_b=bbb, attr_c=xxx) Match Loose (attr_a=xxx; attr_c=ccc) ⇒ No Match!

Every Match

An "every" match between AVPLs succeeds if none of the configuration's AVPs that have a counterpart in the data AVPL fails to match. Its result AVPL contains all the data AVPs that matched.

These may only be used in Criteria and Transforms.

NOTE	As of current (2.0.1), Loose Match does not work as described here, see issue 12184.

"Every" Match Examples

(attr_a=aaa, attr_b=bbb, attr_c=xxx) Match Every (attr_a?, attr_c?) ⇒ (attr_a=aaa, attr_c=xxx)

(attr_a=aaa, attr_b=bbb, attr_c=xxx) Match Every (attr_a?, attr_c?, attr_d=ddd) ⇒ (attr_a=aaa, attr_c=xxx)

(attr_a=aaa, attr_b=bbb, attr_c=xxx) Match Every (attr_a?, attr_c=ccc) ⇒ No Match!

(attr_a=aaa; attr_b=bbb; attr_c=xxx) Match Every (attr_a=xxx, attr_c=ccc) =⇒ No Match!

Strict Match

A Strict match between AVPLs succeeds if and only if every AVP in the configuration AVPL has at least one counterpart in the data AVPL and none of the AVP matches fails. The result AVPL contains all the data AVPs that matched.

These are used between Gop keys (key AVPLs) and Pdu AVPLs. They may also be used in Criteria and Transforms.

Examples

(attr_a=aaa, attr_b=bbb, attr_c=xxx) Match Strict (attr_a?, attr_c=xxx) =⇒ (attr_a=aaa, attr_c=xxx)

(attr_a=aaa, attr_b=bbb, attr_c=xxx, attr_c=yyy) Match Strict (attr_a?, attr_c?) =⇒ (attr_a=aaa, attr_c=xxx, attr_c=yyy)

(attr_a=aaa, attr_b=bbb, attr_c=xxx) Match Strict (attr_a?, attr_c=ccc) =⇒ No Match!

(attr_a=aaa, attr_b=bbb, attr_c=xxx) Match Strict (attr_a?, attr_c?, attr_d?) =⇒ No Match!

AVPL Merge

An AVPL may be merged into another one. That would add to the latter every AVP from the former that does not already exist there.

This operation is done

- between the result of a key match and the Gop's or Gog's AVPL,
- between the result of an Extra match and the Gop's or Gog's AVPL,
- between the result of a Transform match and Pdu's/Gop's AVPL. If the operation specified by the Match clause is Replace, the result AVPL of the match is removed from the item's AVPL before the modify_avpl is merged into it.

Examples

(attr_a=aaa, attr_b=bbb) Merge (attr_a=aaa, attr_c=xxx) former becomes (attr_a=aaa, attr_b=bbb, attr_c=xxx)

(attr_a=aaa, attr_b=bbb) Merge (attr_a=aaa, attr_a=xxx) former becomes (attr_a=aaa, attr_a=xxx, attr_b=bbb)

(attr_a=aaa, attr_b=bbb) Merge (attr_c=xxx, attr_d=ddd) former becomes (attr_a=aaa, attr_b=bbb, attr_c=xxx, attr_d=ddd)

Transforms

A Transform is a sequence of Match rules optionally followed by an instruction how to modify the match result using an additional AVPL. Such modification may be an Insert (merge) or a Replace. The syntax is as follows:

```
Transform name {
    Match [Strict|Every|Loose] match_avpl [[Insert|Replace] modify_avpl] ; // may
occur multiple times, at least once
};
```

For examples of Transforms, check the Manual page.

TODO: migrate the examples here?

The list of Match rules inside a Transform is processed top to bottom; the processing ends as soon as either a Match rule succeeds or all have been tried in vain.

Transforms can be used as helpers to manipulate an item's AVPL before the item is processed further. An item declaration may contain a Transform clause indicating a list of previously declared Transforms. Regardless whether the individual transforms succeed or fail, the list is always executed completely and in the order given, i.e., left to right.

In MATE configuration file, a Transform must be declared before declaring any item which uses it.

Configuration AVPLs

Pdsu's configuration actions

The following configuration AVPLs deal with PDU creation and data extraction.

Pdu declaration block header

In each frame of the capture, MATE will look for source *proto_name*'s PDUs in the order in which the declarations appear in its configuration and will create Pdus of every type it can from that frame, unless specifically instructed that some Pdu type is the last one to be looked for in the frame. If told so for a given type, MATE will extract all Pdus of that type and the previously declared types it finds in the frame but not those declared later.

The complete declaration of a Pdu looks as below; the mandatory order of the diverse clauses is as shown.

```
Pdu name Proto proto_name Transport proto1[/proto2/proto3[/...]]] {
  Payload proto; //optional, no default value
  Extract attribute From proto.field ; //may occur multiple times, at least once
  Transform (transform1[, transform2[, ...]]); //optional
  Criteria [{Accept|Reject}] [{Strict|Every|Loose} match_avpl];
  DropUnassigned {true|false}; //optional, default=false
  DiscardPduData {true|false}; //optional, default=false
  LastExtracted {true|false}; //optional, default=false
};
```

Pdu name

The *name* is a mandatory attribute of a Pdu declaration. It is chosen arbitrarily, except that each *name* may only be used once in MATE's configuration, regardless the class of an item it is used for. The *name* is used to distinguish between different types of Pdus, Gops, and Gogs. The *name* is also used as part of the filterable fields' names related to this type of Pdu which MATE creates.

However, several Pdu declarations may share the same *name*. In such case, all of them are created from each source PDU matching their *Proto*, *Transport*, and *Payload* clauses, while the bodies of their declarations may be totally different from each other. Together with the *Accept* (or *Reject*) clauses, this feature is useful when it is necessary to build the Pdu's AVPL from different sets of source fields depending on contents (or mere presence) of other source fields.

Proto and Transport clauses

Every instance of the protocol *proto_name* PDU in a frame will generate one Pdu with the AVPs extracted from fields that are in the *proto_name*'s range and/or the ranges of underlying protocols specified by the *Transport* list. It is a mandatory attribute of a Pdu declaration. The *proto_name* is the name of the protocol as used in Wireshark display filter.

The Pdu's *Proto*, and its *Transport* list of protocols separated by / tell MATE which fields of a frame can get into the Pdu's AVPL. In order that MATE would extract an attribute from a frame's protocol tree, the area representing the field in the hex display of the frame must be within the area of either the *Proto* or its relative *Transport* s. *Transport* s are chosen moving backwards from the protocol area, in the order they are given.

Proto http Transport tcp/ip does what you'd expect it to - it selects the nearest tcp range that precedes the current http range, and the nearest ip range that precedes that tcp range. If there is another ip range before the nearest one (e.g., in case of IP tunneling), that one is not going to be selected. *Transport* tcp/ip/ip that "logically" should select the encapsulating IP header too doesn't work so far.

Once we've selected the *Proto* and *Transport* ranges, MATE will fetch those protocol fields belonging to them whose extraction is declared using the *Extract* clauses for the Pdu type. The *Transport* list is also mandatory, if you actually don't want to use any transport protocol, use

Transport mate. (This didn't work until 0.10.9).

Payload clause

Other than the Pdu's *Proto* and its *Transport* protocols, there is also a *Payload* attribute to tell MATE from which ranges of *Proto*'s payload to extract fields of a frame into the Pdu. In order to extract an attribute from a frame's tree the highlighted area of the field in the hex display must be within the area of the *Proto*'s relative payload(s). *Payload* s are chosen moving forward from the protocol area, in the order they are given. *Proto http Transport tcp/ip Payload mmse* will select the first mmse range after the current http range. Once we've selected the *Payload* ranges, MATE will fetch those protocol fields belonging to them whose extraction is declared using the *Extract* clauses for the Pdu type.

Extract clause

Each *Extract* clause tells MATE which protocol field value to extract as an AVP value and what string to use as the AVP name. The protocol fields are referred to using the names used in Wireshark display filters. If there is more than one such protocol field in the frame, each instance that fulfills the criteria stated above is extracted into its own AVP. The AVP names may be chosen arbitrarily, but to be able to match values originally coming from different Pdus (e.g., hostname from DNS query and a hostname from HTTP GET request) later in the analysis, identical AVP names must be assigned to them and the dissectors must provide the field values in identical format (which is not always the case).

Transform clause

The *Transform* clause specifies a list of previously declared *Transform* s to be performed on the Pdu's AVPL after all protocol fields have been extracted to it. The list is always executed completely, left to right. On the contrary, the list of Match clauses inside each individual *Transform* is executed only until the first match succeeds.

Criteria clause

This clause tells MATE whether to use the Pdu for analysis. It specifies a match AVPL, an AVPL match type (*Strict*, *Every*, or *Loose*) and the action to be performed (*Accept* or *Reject*) if the match succeeds. Once every attribute has been extracted and eventual transform list has been executed, and if the *Criteria* clause is present, the Pdu's AVPL is matched against the match AVPL; if the match succeeds, the action specified is executed, i.e., the Pdu is accepted or rejected. The default behaviors used if the respective keywords are omitted are *Strict* and *Accept*. Accordingly, if the clause is omitted, all Pdus are accepted.

DropUnassigned clause

If set to *TRUE*, MATE will destroy the Pdu if it cannot assign it to a Gop. If set to *FALSE* (the default if not given), MATE will keep them.

DiscardPduData clause

If set to *TRUE*, MATE will delete the Pdu's AVPL once it has analyzed it and eventually extracted some AVPs from it into the Gop's AVPL. This is useful to save memory (of which MATE uses a lot). If set to *FALSE* (the default if not given), MATE will keep the Pdu attributes.

LastExtracted clause

If set to *FALSE* (the default if not given), MATE will continue to look for Pdus of other types in the frame. If set to *TRUE*, it will not try to create Pdus of other types from the current frame, yet it will continue to try for the current type.

Gop's configuration actions

Gop declaration block header

Declares a Gop type and its prematch candidate key.

```
Gop name On pduname Match key {
  Start match_avpl; // optional
  Stop match_avpl; // optional
  Extra match_avpl; // optional
  Transform transform_list; // optional
  Expiration time; // optional
  IdleTimeout time; // optional
  Lifetime time; // optional
  DropUnassigned [TRUE|FALSE]; //optional
  ShowTree [NoTree|PduTree|FrameTree|BasicTree]; //optional
  ShowTimes [TRUE|FALSE]; //optional, default TRUE
};
```

Gop name

The *name* is a mandatory attribute of a Gop declaration. It is chosen arbitrarily, except that each *name* may only be used once in MATE's configuration, regardless the class of an item it is used for. The *name* is used to distinguish between different types of Pdus, Gops, and Gogs. The *name* is also used as part of the filterable fields' names related to this type of Gop which MATE creates.

On clause

The *name* of Pdus which this type of Gop is supposed to be groupping. It is mandatory.

Match clause

Defines what AVPs form up the *key* part of the Gop's AVPL (the Gop's *key* AVPL or simply the Gop's *key*). All Pdus matching the *key* AVPL of an active Gop are assigned to that Gop; a Pdu which contains the AVPs whose attribute names are listed in the Gop's *key* AVPL, but they do not strictly

match any active Gop's *key* AVPL, will create a new Gop (unless a *Start* clause is given). When a Gop is created, the elements of its key AVPL are copied from the creating Pdu.

Start clause

If given, it tells MATE what match_avpl must a Pdu's AVPL match, in addition to matching the Gop's *key*, in order to start a Gop. If not given, any Pdu whose AVPL matches the Gop's *key* AVPL will act as a start for a Gop. The Pdu's AVPs matching the match_avpl are not automatically copied into the Gop's AVPL.

Stop clause

If given, it tells MATE what match_avpl must a Pdu's AVPL match, in addition to matching the Gop's key, in order to stop a Gop. If omitted, the Gop is "auto-stopped" - that is, the Gop is marked as stopped as soon as it is created. The Pdu's AVPs matching the match_avpl are not automatically copied into the Gop's AVPL.

Extra clause

If given, tells MATE which AVPs from the Pdu's AVPL are to be copied into the Gop's AVPL in addition to the Gop's key.

Transform clause

The *Transform* clause specifies a list of previously declared *Transform* s to be performed on the Gop's AVPL after the AVPs from each new Pdu, specified by the key AVPL and the *Extra* clause's match_avpl, have been merged into it. The list is always executed completely, left to right. On the contrary, the list of *Match* clauses inside each individual *Transform* is executed only until the first match succeeds.

Expiration clause

A (floating) number of seconds after a Gop is *Stop* ped during which further Pdus matching the *Stop* ped Gop's key but not the *Start* condition will still be assigned to that Gop. The default value of zero has an actual meaning of infinity, as it disables this timer, so all Pdus matching the *Stop* ped Gop's key will be assigned to that Gop unless they match the *Start* condition.

IdleTimeout clause

A (floating) number of seconds elapsed from the last Pdu assigned to the Gop after which the Gop will be considered released. The default value of zero has an actual meaning of infinity, as it disables this timer, so the Gop won't be released even if no Pdus arrive - unless the *Lifetime* timer expires.

Lifetime clause

A (floating) of seconds after the Gop *Start* after which the Gop will be considered released regardless anything else. The default value of zero has an actual meaning of infinity.

DropUnassigned clause

Whether or not a Gop that has not being assigned to any Gog should be discarded. If *TRUE*, the Gop is discarded right after creation. If *FALSE*, the default, the unassigned Gop is kept. Setting it to *TRUE* helps save memory and speed up filtering.

TreeMode clause

Controls the display of Pdus subtree of the Gop:

- *NoTree*: completely suppresses showing the tree
- *PduTree*: the tree is shown and shows the Pdus by Pdu Id
- *FrameTree*: the tree is shown and shows the Pdus by the frame number in which they are
- *BasicTree*: needs investigation

ShowTimes clause

Whether or not to show the times subtree of the Gop. If *TRUE*, the default, the subtree with the timers is added to the Gop's tree. If *FALSE*, the subtree is suppressed.

Gog's configuration actions

Gop declaration block header

Declares a Gog type and its prematch candidate key.

```
Gog name {
  Member gopname (key); // mandatory, at least one
  Extra match_avpl; // optional
  Transform transform_list; // optional
  Expiration time; // optional, default 2.0
  GopTree [NoTree|PduTree|FrameTree|BasicTree]; // optional
  ShowTimes [TRUE|FALSE]; // optional, default TRUE
};
```

Gop name

The *name* is a mandatory attribute of a Gog declaration. It is chosen arbitrarily, except that each *name* may only be used once in MATE's configuration, regardless the class of an item it is used for. The *name* is used to distinguish between different types of Pdus, Gops, and Gogs. The *name* is also used as part of the filterable fields' names related to this type of Gop which MATE creates.

Member clause

Defines the *key* AVPL for the Gog individually for each Gop type *gopname*. All *gopname* type Gops whose *key* AVPL matches the corresponding *key* AVPL of an active Gog are assigned to that Gog; a

Gop which contains the AVPs whose attribute names are listed in the Gog's corresponding *key* AVPL, but they do not strictly match any active Gog's *key* AVPL, will create a new Gog. When a Gog is created, the elements of its *key* AVPL are copied from the creating Gop.

Although the *key* AVPLs are specified separately for each of the Member *gopname* s, in most cases they are identical, as the very purpose of a Gog is to group together Gops made of Pdus of different types.

Extra clause

If given, tells MATE which AVPs from any of the Gop's AVPL are to be copied into the Gog's AVPL in addition to the Gog's key.

Expiration clause

A (floating) number of seconds after all the Gops assigned to a Gog have been released during which new Gops matching any of the session keys should still be assigned to the existing Gog instead of creating a new one. Its value can range from 0.0 to infinite. Defaults to 2.0 seconds.

Transform clause

The *Transform* clause specifies a list of previously declared *Transform* s to be performed on the Gog's AVPL after the AVPs from each new Gop, specified by the *key* AVPL and the *Extra* clause's match_avpl, have been merged into it. The list is always executed completely, left to right. On the contrary, the list of *Match* clauses inside each individual *Transform* is executed only until the first match succeeds.

TreeMode clause

Controls the display of Gops subtree of the Gog:

- *NoTree*: completely suppresses showing the tree
- *BasicTree*: needs investigation
- *FullTree*: needs investigation

ShowTimes clause

Whether or not to show the times subtree of the Gog. If *TRUE*, the default, the subtree with the timers is added to the Gog's tree. If *FALSE*, the subtree is suppressed.

Settings Config AVPL

The **Settings** config element is used to pass to MATE various operational parameters. the possible parameters are

GogExpiration

How long in seconds after all the gops assigned to a gog have been released new gops matching any of the session keys should create a new gog instead of being assigned to the previous one. Its value can range from 0.0 to infinite. Defaults to 2.0 seconds.

DiscardPduData

Whether or not the AVPL of every Pdu should be deleted after it was being processed (saves memory). It can be either *TRUE* or *FALSE*. Defaults to *TRUE*. Setting it to *FALSE* can save you from a headache if your config does not work.

DiscardUnassignedPdu

Whether Pdus should be deleted if they are not assigned to any Gop. It can be either *TRUE* or *FALSE*. Defaults to *FALSE*. Set it to *TRUE* to save memory if unassigned Pdus are useless.

DiscardUnassignedGop

Whether GoPs should be deleted if they are not assigned to any session. It can be either *TRUE* or *FALSE*. Defaults to *FALSE*. Setting it to *TRUE* saves memory.

ShowPduTree

ShowGopTimes

Debugging Stuff

The following settings are used to debug MATE and its configuration. All levels are integers ranging from 0 (print only errors) to 9 (flood me with junk), defaulting to 0.

Debug declaration block header

```
Debug {
  Filename "path/name"; //optional, no default value
  Level [0-9]; //optional, generic debug level
  Pdu Level [0-9]; //optional, specific debug level for Pdu handling
  Gop Level [0-9]; //optional, specific debug level for Gop handling
  Gog Level [0-9]; //optional, specific debug level for Gog handling
};
```

Filename clause

The {{{path/name}}} is a full path to the file to which debug output is to be written. Non-existent file will be created, existing file will be overwritten at each opening of a capture file. If the statement is missing, debug messages are written to console, which means they are invisible on Windows.

Level clause

Sets the level of debugging for generic debug messages. It is an integer ranging from 0 (print only errors) to 9 (flood me with junk).

Pdu Level clause

Sets the level of debugging for messages regarding Pdu creation. It is an integer ranging from 0 (print only errors) to 9 (flood me with junk).

Gop Level clause

Sets the level of debugging for messages regarding Pdu analysis (that is how do they fit into ?GoPs). It is an integer ranging from 0 (print only errors) to 9 (flood me with junk).

Gog Level clause

Sets the level of debugging for messages regarding GoP analysis (that is how do they fit into ?GoGs). It is an integer ranging from 0 (print only errors) to 9 (flood me with junk).

Settings Example

```
Action=Settings; SessionExpiration=3.5; DiscardPduData=FALSE;
```

Action=Include

Will include a file to the configuration.

```
Action=Include; {Filename=filename;|Lib=libname;}
```

Filename

The filename of the file to include. If it does not begin with '/' it will look for the file in the current path.

Lib

The name of the lib config to include. will look for libname.mate in wiresharks_dir/matelib.

Include Example

```
Action=Include; Filename=rtsp.mate;
```

This will include the file called "rtsp.mate" into the current config.

Appendix A: Wireshark Messages

Wireshark provides you with additional information generated out of the plain packet data or it may need to indicate dissection problems. Messages generated by Wireshark are usually placed in square brackets (“[]”).

Packet List Messages

These messages might appear in the packet list.

[Malformed Packet]

Malformed packet means that the protocol dissector can’t dissect the contents of the packet any further. There can be various reasons:

- *Wrong dissector*: Wireshark erroneously has chosen the wrong protocol dissector for this packet. This will happen e.g., if you are using a protocol not on its well known TCP or UDP port. You may try Analyze|Decode As to circumvent this problem.
- *Packet not reassembled*: The packet is longer than a single frame and it is not reassembled, see Packet Reassembly for further details.
- *Packet is malformed*: The packet is actually wrong (malformed), meaning that a part of the packet is just not as expected (not following the protocol specifications).
- *Dissector is buggy*: The corresponding protocol dissector is simply buggy or still incomplete.

Any of the above is possible. You’ll have to look into the specific situation to determine the reason. You could disable the dissector by disabling the protocol on the Analyze menu and check how Wireshark displays the packet then. You could (if it’s TCP) enable reassembly for TCP and the specific dissector (if possible) in the Edit|Preferences menu. You could check the packet contents yourself by reading the packet bytes and comparing it to the protocol specification. This could reveal a dissector bug. Or you could find out that the packet is indeed wrong.

[Packet size limited during capture]

The packet size was limited during capture, see “Limit each packet to n bytes” at the The “Capture Options” Dialog Box. While dissecting, the current protocol dissector was simply running out of packet bytes and had to give up. There’s nothing else you can do now, except to repeat the whole capture process again with a higher (or no) packet size limitation.

Packet Details Messages

These messages might appear in the packet details.

[Response in frame: 123]

The current packet is the request of a detected request/response pair. You can directly jump to the corresponding response packet by double clicking on the message.

[Request in frame: 123]

Same as “Response in frame: 123” above, but the other way round.

[Time from request: 0.123 seconds]

The time between the request and the response packets.

[Stream setup by PROTOCOL (frame 123)]

The session control protocol (SDP, H225, etc.) message which signaled the creation of this session. You can directly jump to the corresponding packet by double clicking on this message.