



# Learn Wireshark

Second Edition

---

A definitive guide to expertly analyzing protocols and  
troubleshooting networks using Wireshark

Lisa Bock



# Learn Wireshark

## *Second Edition*

A definitive guide to expertly analyzing protocols and troubleshooting networks using Wireshark

**Lisa Bock**



BIRMINGHAM—MUMBAI

# Learn Wireshark

## *Second Edition*

Copyright © 2022 Packt Publishing

*All rights reserved.* No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

**Group Product Manager:** Vijin Boricha  
**Publishing Product Manager:** Prachi Sawant  
**Content Development Editor:** Romy Dias  
**Technical Editor:** Rajat Sharma  
**Copy Editor:** Safis Editing  
**Project Coordinator:** Ashwin Dinesh Kharwa  
**Proofreader:** Safis Editing  
**Indexer:** Sejal Dsilva  
**Production Designer:** Roshan Kawale  
**Marketing Coordinator:** Sanjana Gupta

First Published: August 2019

Second Edition: June 2022

Production reference: 1010722

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-80323-167-9

[www.packt.com](http://www.packt.com)

*To all dreamers, know that there isn't always a clear path to achieving your dream.  
In addition to celebrating and rejoicing each milestone, there will be times of  
great sorrow and despair along the way. Nonetheless, keep moving toward your  
dream while being authentic, harmonious, and true to yourself. One day you'll  
see a sign, and you'll say to yourself with a smile, "I have arrived."*



# Contributors

## About the author

**Lisa Bock** is an experienced author with a demonstrated history of working in the e-learning industry. She is a security ambassador with a broad range of IT skills and knowledge, including Cisco security, CyberOps, Wireshark, biometrics, ethical hacking, and the IoT. Lisa is an author for LinkedIn Learning and an award-winning speaker who has presented at several national conferences. She holds an MS in computer information systems/information assurance from UMGC. Lisa was an associate professor in the IT department at Pennsylvania College of Technology (Williamsport, PA) from 2003 until her retirement in 2020. She is involved with various volunteer activities, and she and her husband, Mike, enjoy bike riding, watching movies, and traveling.

*I want to thank my friends and family for their ongoing support. I am also grateful to the entire Packt team, who work very hard to create an exceptional product. Finally, I'd like to thank my students, who push me to deliver the very best educational content.*

## About the reviewer

**Nick Parlow** is a Fujitsu Fellow and Distinguished Engineer, and has been an escalation engineer for Fujitsu in the UK for nearly 20 years, specializing in messaging technologies and networks. He has fixed stuff for central government, the Ministry of Defence, and his local school. He has master's degrees in network engineering from Sheffield Hallam University and software engineering from the University of Northumbria.

Nick is a Microsoft Certified Trainer and holds many other credentials, but is most proud of being a Raspberry Pi Certified Educator and Code Club volunteer. When he's not working, writing books, reviewing books, soldering things, or taking blurry photos of the night sky, he likes to play with chainsaws.

*I'd like to thank the author, Lisa Bock, and the team at Packt for giving me the opportunity to do something that has been wholly enjoyable – reviewing this great book. Most thanks, however, go to my long-suffering family and colleagues for giving me the time and support to do so. Thank you, Chris, Bryn, Jon, Caroline, Craig, and everybody else. You're brilliant.*



# Table of Contents

## Preface

---

## Part 1 Traffic Capture Overview

### 1

#### Appreciating Traffic Analysis

---

Reviewing packet analysis	4	Identifying where to use packet analysis	17
Exploring early packet sniffers	5	Analyzing traffic on a LAN	17
Evaluating devices that use packet analysis	6	Outlining when to use packet analysis	19
Capturing network traffic	7	Troubleshooting latency issues	19
Recognizing who benefits from using packet analysis	8	Testing IoT devices	20
Assisting developers	8	Monitoring for threats	20
Helping network administrators monitor the network	9	Baselining the network	21
Educating students on protocols	12	Getting to know Wireshark	22
Alerting security analysts to threats	13	Summary	23
Arming hackers with information	14	Questions	24

### 2

#### Using Wireshark

---

Examining the Wireshark interface	28	Finding information	34
Streamlining the interface	28	Understanding the phases of packet analysis	34
Discovering keyboard shortcuts	31	Gathering network traffic	34
Recognizing the Wireshark authors	32	Decoding the raw bits	37

Displaying the captured data	38	Dissecting protocols	44
Analyzing the packet capture	41	<b>Summary</b>	<b>45</b>
<b>Using CLI tools with Wireshark</b>	<b>42</b>	<b>Questions</b>	<b>46</b>
Exploring tshark	42		

## 3

### Installing Wireshark

---

<b>Discovering support for different OSes</b>	<b>50</b>	Beginning the installation	58
Using Wireshark on Windows	50	Choosing components	58
Running Wireshark on Unix	50	Creating shortcuts and selecting an install location	62
Installing Wireshark on macOS	51	Capturing packets and completing the installation	63
Deploying Wireshark on Linux	51	<b>Reviewing available resources</b>	<b>65</b>
Working with Wireshark on other systems	52	Viewing news and help topics	65
<b>Comparing different capture engines</b>	<b>54</b>	Evaluating download options	67
Understanding libpcap	54	<b>Summary</b>	<b>69</b>
Examining WinPcap	54	<b>Questions</b>	<b>69</b>
Grasping Npcap	55	<b>Further reading</b>	<b>71</b>
<b>Performing a standard Windows installation</b>	<b>58</b>		

## 4

### Exploring the Wireshark Interface

---

<b>Opening the Wireshark welcome screen</b>	<b>74</b>	Printing packets and closing Wireshark	82
Selecting a file	74	<b>Discovering the Edit menu</b>	<b>84</b>
Capturing traffic	75	Copying items and finding packets	84
<b>Exploring the File menu</b>	<b>76</b>	Marking or ignoring packets	88
Opening a file, closing, and saving	77	Setting a time reference	89
Exporting packets, bytes, and objects	78	Personalizing your work area	90
		<b>Exploring the View menu</b>	<b>91</b>

Enhancing the interface	91	Refreshing the view	98
Formatting time and name resolution	93	<b>Summary</b>	<b>101</b>
Modifying the display	96	<b>Questions</b>	<b>101</b>

## Part 2 Getting Started with Wireshark

### 5

#### Tapping into the Data Stream

<b>Reviewing network architectures</b>	<b>108</b>	Comparing conversations and endpoints	119
Comparing different types of networks	108	<b>Realizing the importance of baselining</b>	<b>123</b>
Exploring various types of media	110	Planning the baseline	123
<b>Learning various capture methods</b>	<b>113</b>	Capturing traffic	123
Providing input	114	Analyzing the captured traffic	124
Directing output	114	Saving the baselines	125
Selecting options	116	<b>Summary</b>	<b>126</b>
<b>Tapping into the stream</b>	<b>118</b>	<b>Questions</b>	<b>127</b>

### 6

#### Personalizing the Interface

<b>Personalizing the layout</b>	<b>130</b>	Adding, editing, and deleting columns	141
Altering the appearance	130	Refining the font and colors	145
Changing the layout	132	<b>Adding comments</b>	<b>148</b>
<b>Creating a tailored configuration profile</b>	<b>136</b>	Attaching comments to files	148
Customizing a profile	136	Entering packet comments	148
Crafting buttons	139	Viewing and saving comments	149
<b>Adjusting columns, font, and colors</b>	<b>141</b>	<b>Summary</b>	<b>150</b>
		<b>Questions</b>	<b>151</b>

## 7

### Using Display and Capture Filters

---

<b>Filtering network traffic</b>	<b>154</b>	<b>Understanding the expression builder</b>	<b>168</b>
Analyzing traffic	154	Building an expression	170
Comparing the filters' files	156		
<b>Comprehending display filters</b>	<b>159</b>	<b>Discovering shortcuts and handy filters</b>	<b>172</b>
Editing display filters	160	Embracing filter shortcuts	172
Using bookmarks	161	Applying useful filters	175
<b>Creating capture filters</b>	<b>162</b>	<b>Summary</b>	<b>177</b>
Modifying capture filters	164	<b>Questions</b>	<b>177</b>
Bookmarking a filter	168	<b>Further reading</b>	<b>179</b>

## 8

### Outlining the OSI Model

---

<b>An overview of the OSI model</b>	<b>182</b>	<b>Traveling over the Physical layer</b>	<b>197</b>
Developing the framework	182	<b>Exploring the encapsulation process</b>	<b>198</b>
Using the framework	183	Viewing the data	199
<b>Discovering the purpose of each layer, the protocols, and the PDUs</b>	<b>183</b>	Identifying the segment	199
Evaluating the Application layer	185	Characterizing the packet	200
Dissecting the Presentation layer	186	Forming the frame	200
Learning about the Session layer	188	<b>Demonstrating frame formation in Wireshark</b>	<b>201</b>
Appreciating the Transport layer	190	Examining the network bindings	202
Explaining the Network layer	193	<b>Summary</b>	<b>203</b>
Examining the Data Link layer	196	<b>Questions</b>	<b>203</b>

## Part 3 The Internet Suite TCP/IP

### 9

#### Decoding TCP and UDP

---

Reviewing the transport layer	210	Dissecting the window size	229
Describing TCP	211	Viewing additional header values	232
Establishing and maintaining a connection	211	<b>Understanding UDP</b>	<b>234</b>
Exploring a single TCP frame	214	Studying a single UDP frame	235
<b>Examining the 11-field TCP header</b>	<b>219</b>	<b>Discovering the four-field UDP header</b>	<b>236</b>
Exploring TCP ports	220	Analyzing the UDP header fields	236
Sequencing bytes	222	<b>Summary</b>	<b>237</b>
Acknowledging data	225	<b>Questions</b>	<b>238</b>
Following the flags	228	<b>Further reading</b>	<b>239</b>

### 10

#### Managing TCP Connections

---

<b>Dissecting the three-way handshake</b>	<b>242</b>	Permitting SACK	257
Isolating a single stream	243	Using timestamps	259
Identifying the handshake packets	248	<b>Understanding TCP protocol preferences</b>	<b>260</b>
<b>Learning TCP options</b>	<b>252</b>	Modifying TCP preferences	262
Grasping the EOL option	254	<b>Tearing down a connection</b>	<b>264</b>
Using NOP	254	<b>Summary</b>	<b>266</b>
Defining the MSS	255	<b>Questions</b>	<b>266</b>
Scaling the WS	256	<b>Further reading</b>	<b>268</b>



## 11

### Analyzing IPv4 and IPv6

---

Reviewing the network layer	270	Editing protocol preferences	287
Understanding the purpose of IP	271	Reviewing IPv4 preferences	287
Outlining IPv4	272	Adjusting preferences for IPv6	290
Dissecting the IPv4 header	273	Discovering tunneling protocols	291
Modifying options for IPv4	282	Summary	292
Exploring IPv6	282	Questions	293
Navigating the IPv6 header fields	283	Further reading	295

## 12

### Discovering ICMP

---

Understanding the purpose of ICMP	298	Providing information using ICMPv6	312
Understanding the ICMP header	299	Evaluating type and code values	315
Investigating the data payload	302	Reviewing ICMP type and code values	315
Dissecting ICMP and ICMPv6	305	Defining ICMPv6 type and code values	317
Reviewing ICMP	305	Configuring firewall rules	318
Outlining ICMPv6	306	Acting maliciously	318
Sending ICMP messages	307	Allowing only necessary types	323
Reporting errors on the network	308	Summary	324
Issuing query messages	311	Questions	324
		Further reading	326

## Part 4 Deep Packet Analysis of Common Protocols

## 13

### Diving into DNS

---

Recognizing the purpose of DNS	330	Mapping an IP address	330
		Types of DNS servers	333

Transporting DNS	335	<b>Evaluating queries and responses</b>	<b>345</b>
<b>Comparing types and classes of RRs</b>	<b>336</b>	Caching a response	346
Breaking down DNS types	336	Calculating response times	347
Examining the RR structure	337	Testing using nslookup	351
<b>Reviewing the DNS packet</b>	<b>338</b>	Securing DNS	353
Examining the header	339	<b>Summary</b>	<b>354</b>
Dissecting the packet structure	343	<b>Questions</b>	<b>354</b>
Outlining the query section	344	<b>Further reading</b>	<b>356</b>

## 14

### Examining DHCP

<b>Recognizing the purpose of DHCP</b>	<b>360</b>	Understanding DHCP messages	375
Configuring the client's IP address	361	Comparing DHCP options	376
Using a DHCP relay agent	361	<b>Following a DHCP example</b>	<b>377</b>
Working with IPv6 addresses	363	Releasing an IP address	377
Addressing security issues	365	Broadcasting a discover packet	379
<b>Stepping through the DORA process</b>	<b>366</b>	Delivering an offer	380
Moving through DHCP states	366	Requesting an IP address	382
Obtaining an IP address	367	Acknowledging the offer	383
Leasing an IP address	370	<b>Summary</b>	<b>384</b>
<b>Dissecting a DHCP header</b>	<b>372</b>	<b>Questions</b>	<b>385</b>
Examining DHCP field values	373	<b>Further reading</b>	<b>387</b>

## 15

### Decoding HTTP

<b>Describing HTTP</b>	<b>390</b>	<b>Keeping track of the connection</b>	<b>394</b>
Dissecting a web page	390	Evaluating connection types	395
Understanding HTTP versions	393	Maintaining state with cookies	396
Recognizing HTTP methods	394	<b>Comparing request and response messages</b>	<b>398</b>

Viewing an HTTP request	398	Responding to the client	407
Responding to the client	400	Ending the conversation	412
<b>Following an HTTP stream</b>	<b>402</b>	<b>Summary</b>	<b>412</b>
Beginning the conversation	405	<b>Questions</b>	<b>413</b>
Requesting data	406	<b>Further reading</b>	<b>414</b>

## 16

### Understanding ARP

---

<b>Understanding the role and purpose of ARP</b>	<b>418</b>	Reversing ARP	427
Resolving MAC addresses	419	Evaluating InARP	428
Investigating an ARP cache	421	Issuing a gratuitous ARP	430
Replacing ARP with NDP in IPv6	423	Working on behalf of ARP	430
<b>Exploring ARP headers and fields</b>	<b>423</b>	<b>Comparing ARP attacks and defense methods</b>	<b>432</b>
Identifying a standard ARP request/reply	423	Comparing ARP attacks and tools	432
Breaking down the ARP header fields	425	Defending against ARP attacks	435
<b>Examining different types of ARP</b>	<b>427</b>	<b>Summary</b>	<b>436</b>
		<b>Questions</b>	<b>437</b>
		<b>Further reading</b>	<b>438</b>

## Part 5 Working with Packet Captures

## 17

### Determining Network Latency Issues

---

<b>Analyzing latency issues</b>	<b>442</b>	Common transmission errors	450
Grasping latency, throughput, and packet loss	442	<b>Discovering expert information</b>	<b>454</b>
Learning the importance of time values	446	Viewing the column headers	456
<b>Understanding coloring rules</b>	<b>447</b>	Assessing the severity	457
<b>Exploring the Intelligent Scrollbar</b>	<b>449</b>	Organizing the information	458
		<b>Summary</b>	<b>461</b>
		<b>Questions</b>	<b>462</b>

## 18

### Subsetting, Saving, and Exporting Captures

---

Discovering ways to subset traffic	466	Recognizing ways to export components	477
Dissecting by an IP address	467	Selecting specified packets	478
Narrowing down by conversations	470	Exporting various objects	480
Minimizing by port number	471	Identifying why and how to add comments	482
Breaking down by protocol	472	Providing file and packet comments	482
Subsetting by stream	473	Saving and viewing comments	484
Understanding options to save a file	474	Summary	487
Using Save as	476	Questions	487

## 19

### Discovering I/O and Stream Graphs

---

Discovering the Statistics menu	492	Comparing TCP stream graphs	506
Viewing general information	493	Using time sequence graphs	506
Assessing protocol effectiveness	494	Determining throughput	512
Graphing capture issues	497	Assessing Round Trip Time	514
Creating I/O graphs	499	Evaluating window scaling	515
Examining errors	500	Summary	517
Graphing duplicate ACKs	501	Questions	517
Modifying the settings	502		
Exploring other options	504		

## 20

### Using CloudShark for Packet Analysis

---

Discovering CloudShark	522	Outlining the various filters and graphs	532
Modifying the preferences	523	Displaying data using filters	533
Uploading captures	525	Viewing data using graphs	534
Working with capture files	526		

---

<b>Evaluating the different analysis tools</b>	<b>537</b>	<b>Locating sample captures</b>	<b>544</b>
Following the stream and viewing conversations	538	Examining captures	544
Viewing packet lengths and VoIP activity	540	Finding more captures	546
Exploring HTTP analysis and wireless traffic	541	<b>Summary</b>	<b>546</b>
Monitoring possible threats	542	<b>Questions</b>	<b>547</b>
		<b>Further reading</b>	<b>548</b>

**Assessments**

---

**Index**

---

**Other Books You May Enjoy**

---

# Preface

In the early 2000s, a coworker introduced me to Ethereal, the precursor to Wireshark. I remember looking at the screen as my laptop gobbled up traffic and thinking, "I don't know what this is, but I want to know!" Over the next few years, I immersed myself in learning as much as possible about packet analysis using Wireshark. I attended training, watched videos, and read books that helped me compile and curate my knowledge and respect for what the packets tell us.

I have taught network and security courses and presented at conferences about the many benefits of using Wireshark. In this second edition of *Learn Wireshark*, I want to share my knowledge with you. Each chapter has multiple opportunities for a hands-on approach. Using the examples, you will make sense of the data and understand what the packets are telling you. I'll outline how to conduct a detailed search, follow the data stream, and identify endpoints so that you can troubleshoot latency issues and actively recognize network attacks. Join me on this journey, and you'll soon realize that the ability to understand what's happening on the network is a superpower!

## Who this book is for

This book is for network administrators, security analysts, students, teachers, and anyone interested in learning about packet analysis using Wireshark. Basic knowledge of network fundamentals, devices, and protocols, along with an understanding of different topologies, will be beneficial as you move through the material.

## What this book covers

*Chapter 1, Appreciating Traffic Analysis*, describes the countless places and reasons to conduct packet analysis. In addition, we'll cover the many benefits of using Wireshark, an open source protocol analyzer that includes many rich features.

*Chapter 2, Using Wireshark*, starts with an overview of the beginnings of today's Wireshark. We'll examine the interface and review the phases of packet analysis. Finally, we'll cover the built-in tools, with a closer look at `tshark` (or terminal-based Wireshark), a lightweight alternative to Wireshark.

*Chapter 3, Installing Wireshark*, illustrates how Wireshark provides support for different operating systems. We'll compare the different capture engines, such as WinPCap, LibPcap, and Npcap, walk through a standard Windows installation, and then review the resources available at <https://www.wireshark.org/>.

*Chapter 4, Exploring the Wireshark Interface*, provides a deeper dive into some of the common elements of Wireshark to improve your workflow. We'll investigate the welcome screen and common menu choices, such as **File**, **Edit**, and **View**, so that you can easily navigate the interface during an analysis.

*Chapter 5, Tapping into the Data Stream*, starts with a comparison of the different network architectures and then moves on to the various capture options. You'll discover the conversations and endpoints you'll see when tapping into the stream, and then learn about the importance of baselining network traffic.

*Chapter 6, Personalizing the Interface*, helps you to realize all the ways you can customize the many aspects of the interface. You'll learn how to personalize the layout and general appearance, create a tailored configuration profile, adjust the columns, font, and color, and create buttons.

*Chapter 7, Using Display and Capture Filters*, helps you to make examining a packet capture less overwhelming. We'll take a look at how to narrow your scope by filtering network traffic. We'll compare and contrast display and capture filters, discover the shortcuts used to build filters, and conclude with a review of the expression builder.

*Chapter 8, Outlining the OSI Model*, provides an overview of the **Open Systems Interconnection (OSI)** model, a seven-layer framework that outlines how the OS prepares data for transport on the network. We'll review the purpose, protocols, and **Protocol Data Units (PDUs)** of each layer, explore the encapsulation process, and demonstrate the frame formation in Wireshark.

*Chapter 9, Decoding TCP and UDP*, is a deep dive into two of the key protocols in the transport layer – the **Transmission Control Protocol (TCP)** and the **User Datagram Protocol (UDP)**. We'll review the purpose of the transport layer and then evaluate the header and field values of both the TCP and the UDP.

*Chapter 10, Managing TCP Connections*, begins by examining the three-way handshake. We'll discover the TCP options, get a better understanding of the TCP protocol preferences, and then conclude with an overview of the TCP teardown process.

*Chapter 11, Analyzing IPv4 and IPv6*, provides a breakdown of the purpose of the **Internet Protocol (IP)**. We'll outline IPv4 and the header fields and then explore the streamlined header of IPv6. We'll summarize with a discussion of the protocol preferences and see how IPv4 and IPv6 can coexist by using tunneling protocols.

*Chapter 12, Discovering ICMP*, details the purpose of the **Internet Control Message Protocol (ICMP)**. We'll dissect ICMP and ICMPv6, compare query and error messages, and discuss the ICMP type and code values. We'll cover how ICMP can be used in malicious ways and outline the importance of configuring firewall rules.

*Chapter 13, Diving into DNS*, outlines the significance of the **Domain Name System (DNS)**. You'll learn how DNS works when resolving a hostname to an IP address. We'll compare the different types of records, step through a query and response, review the DNS header, and calculate the DNS response time using Wireshark.

*Chapter 14, Examining DHCP*, begins by explaining the need for the **Dynamic Host Configuration Protocol (DHCP)**. We'll then outline the **DORA** process – **Discover Offer Request Acknowledge**. We'll dissect a DHCP header and review all the field values, flags, and port numbers, and then finish by stepping through a DHCP example.

*Chapter 15, Decoding HTTP*, highlights the **Hypertext Transfer Protocol (HTTP)**, an application layer protocol used when browsing the web. We'll learn the details of HTTP, explore common methods of transport, and dissect the header and fields. We'll then compare request and response messages, and then summarize by following an HTTP stream.

*Chapter 16, Understanding ARP*, takes a closer look at the **Address Resolution Protocol (ARP)**, which is a significant protocol in delivering data. We'll outline the role and purpose of ARP, explore the header and fields, describe the different types of ARP, and take a brief look at ARP attacks.

*Chapter 17, Determining Network Latency Issues*, outlines how even a beginner can diagnose network problems. We'll explore coloring rules and the Intelligent Scrollbar, and then conclude with an overview of the expert information, which divides the alerts into categories and guides you through a more targeted evaluation.

*Chapter 18, Subsetting, Saving, and Exporting Captures*, helps you to explore the many different ways in which to break down a packet capture into smaller files for analysis. We'll cover the different options when saving a file, discover ways to export components such as objects, session keys, and packet bytes, and then outline why and how to add comments.

*Chapter 19, Discovering I/O and Stream Graphs*, begins by covering the many ways the statistics menu can help us when analyzing a capture file. We'll create basic I/O graphs to help visualize network issues and summarize by comparing how the different TCP stream graphs provide a visual representation of the streams.



*Chapter 20, Using CloudShark for Packet Analysis*, covers CloudShark, an online application that is similar to Wireshark. You'll learn how to filter traffic and generate graphs. We'll then review how you can share captures with colleagues and outline where you can find sample captures so that you can continue improving your skills.

## To get the most out of this book

To prepare for working with Wireshark, download and install the latest version on your system. Detailed instructions are listed in *Chapter 3, Installing Wireshark*.

To get the most out of each chapter, when there is a reference to a packet capture, download the files so that you can follow along with the lessons.

In addition to this, practice your skills on your own and, in particular, review the common protocols in the TCP/IP suite so that you can deepen your knowledge and become more proficient in packet analysis.

## Download the example code files

All Wireshark capture files are referenced within the book. Download the appropriate capture files from the online repositories so that you can follow along with the lessons.

## Download the color images

We also provide a PDF file that has color images of the screenshots and diagrams used in this book. You can download it here: <https://packt.link/iF8Fj>.

## Conventions used

There are a number of text conventions used throughout this book.

**Code in text:** Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "To write to a file, use `-w`, then the filename and path."

Any command-line input or output is written as follows:

```
C:\Program Files\Wireshark>tshark -i "ethernet 2" -w Test-  
Tshark.pcap -a duration:10
```

**Bold:** Indicates a new term, an important word, or words that you see onscreen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: "Once you're in CloudShark, select the **Export | Download File** drop-down menu."

**Tips or Important Notes**

Appear like this.

## Get in touch

Feedback from our readers is always welcome.

**General feedback:** If you have questions about any aspect of this book, email us at [customer-care@packtpub.com](mailto:customer-care@packtpub.com) and mention the book title in the subject of your message.

**Errata:** Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit [www.packtpub.com/support/errata](http://www.packtpub.com/support/errata) and fill in the form.

**Piracy:** If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at [copyright@packt.com](mailto:copyright@packt.com) with a link to the material.

**If you are interested in becoming an author:** If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit [authors.packtpub.com](http://authors.packtpub.com).

## Share Your Thoughts

Once you've read *Learn Wireshark - Second Edition*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.



# Part 1

# Traffic Capture Overview

In this section, we'll outline the value of traffic analysis, learn about the evolution of Wireshark, and step through the phases of packet analysis. We'll then discuss some of the command-line interface tools, outline how to download and install Wireshark, and explore the interface along with commonly accessed menu choices.

The following chapters will be covered under this section:

- *Chapter 1, Appreciating Traffic Analysis*
- *Chapter 2, Using Wireshark*
- *Chapter 3, Installing Wireshark*
- *Chapter 4, Exploring the Wireshark Interface*



# 1

# Appreciating Traffic Analysis

Today's networks are complex, and many times, when faced with issues, the only way you can solve the problem is if you can see the problem. For that very reason, packet analysis, using tools such as Wireshark, has been around for many years. In addition to manually conducting packet analysis using Wireshark, today's devices incorporate the ability to pull data from the network and examine its contents. This function helps the network administrator to troubleshoot, test, baseline, and monitor the network for threats.

This chapter will help you to recognize the many benefits of using Wireshark for packet analysis. You'll learn about its history as an exceptional open source software product, which includes many rich features. You'll discover how various groups can benefit from using packet analysis, such as network administrators, students, and security analysts. In addition, we'll cover the many places in which to conduct packet analysis, including on a **Local Area Network (LAN)**, on a host, or in the real world. Finally, you'll learn how Wireshark has the ability to decode hundreds of different protocols and is constantly being improved, making it the optimal tool for monitoring the network.

In this chapter, we will address all of this by covering the following topics:

- Reviewing packet analysis
- Recognizing who benefits from using packet analysis
- Identifying where to use packet analysis
- Outlining when to use packet analysis
- Getting to know Wireshark

## Reviewing packet analysis

Packet analysis examines packets to understand the characteristics and structure of the traffic flow, either during a live capture or by using a previously captured file. The analyst can complete packet analysis by either studying one packet at a time or as a complete capture.

When monitoring the network for analysis, we capture traffic using specialized software such as Wireshark or `tshark`. Once the data is captured and we save the file, the software stores the data in a file that is commonly called a **packet capture** or **PCAP** file.

Packet analysis benefits many groups, including the following:

- **Network administrators:** Use packet analysis to gain information about current network conditions.
- **Security analysts:** Use packet analysis to determine whether there is anything unusual or suspicious about the traffic when carrying out a forensic investigation.
- **Students:** Use packet analysis as a learning tool to better understand the workings of different protocols.
- **Hackers:** Use packet analysis to sniff network traffic while conducting footprinting and reconnaissance in order to gain valuable information about the network.

We use packet analysis in many places, including on a LAN, on a host, or in the real world. Additionally, we use packet analysis when troubleshooting latency issues, testing **Internet of Things (IoT)** devices, and as a tool when baselining the network.

Today, packet analysis using Wireshark is a valuable skill. However, analyzing packets has been around in the networking world for many years. As early as the 1990s, various tools enabled analysts to carry out packet analysis on the network to troubleshoot errors and to monitor server behavior. In the next section, we'll examine some of the early tools used to monitor network activity.

## Exploring early packet sniffers

Packet analysis has been around in some form for over 20 years, as a diagnostic tool, to observe data and other information traveling across the network. Packet analysis is also referred to as **sniffing**. The term refers to early packet sniffers, which *sniffed* or captured traffic as it traveled across the network. In the 1990s, Novell, a software company, developed the **Novell LANalyzer**, which had a graphical UI and dashboard to examine network traffic. Concurrently, Microsoft introduced its Network Monitor.

Over the last 20 years, there have been many other packet analyzers and tools to sniff traffic, including the following:

Tool	Description
Cain and Abel	This tool can gather passwords and record Voice over Internet Protocol (VoIP) conversations.
NarusInsight	Formerly known as Carnivore, this was used to monitor all internet traffic.
dSniff	This passively monitors a network for interesting traffic.
Ettercap	This eavesdrops to capture passwords, emails, and files.
Tcpdump	This is a protocol analyzer that runs from the command line.
Security Onion	This is an open source tool that combines packet capture with an Intrusion Detection System (IDS).
Wireshark	This is a packet sniffer used to analyze network traffic.

Table 1.1 – Packet analyzers and tools

Most packet analyzers work in a similar manner. They capture data and then decode the raw bits in the field values according to the appropriate **Request for Comment (RFC)** or other specifications. Once done, the data is presented in a meaningful fashion.

Packet analysis tools range in appearance and functionality, as follows:

- They provide simple text-based analysis, such as terminal-based Wireshark (`tshark`).
- They deliver a rich graphical UI with advanced **artificial intelligence (AI)**-based expert systems that guide the analyst through a more targeted evaluation.

In the next section, we'll take a look at the various devices that use packet analysis today.



## Evaluating devices that use packet analysis

Packet analysis and traffic sniffing are used by many devices on the network, including routers, switches, and firewall appliances. As data flows across the network, the devices gather and interpret the packet's raw bits and examine the field values in each packet to decide on what action should be taken.

Devices examine network traffic in the following manner:

- A **router** captures the traffic and examines the IP header to determine where to send the traffic, as part of the routing process.
- An **IDS** examines the traffic and alerts the network administrator if there is any unusual or suspicious behavior.
- A **firewall** monitors all traffic and will drop any packets that are not in line with the **Access Control List (ACL)**.

For example, when data passes through a firewall, the device examines the traffic and determines whether to allow or deny the packets according to the ACL.

### Using an ACL

When using a firewall, an ACL governs the type of traffic that is allowed on the network. For example, an ACL has the following entries:

- Allow outbound SYN packets. The destination port is 80.
- Allow inbound SYN-ACK packets. The source port is 80.

To decide whether to allow or deny a packet, the firewall must check each header as it passes through the device. It will determine variables such as IP addresses, **Transmission Control Protocol (TCP)** flags, and port numbers that are in use. If the packet does not meet the ACL entry, the firewall will drop the packet. As shown in the following diagram, an inbound SYN packet with a destination port of 80 is blocked because it does not match the rule:

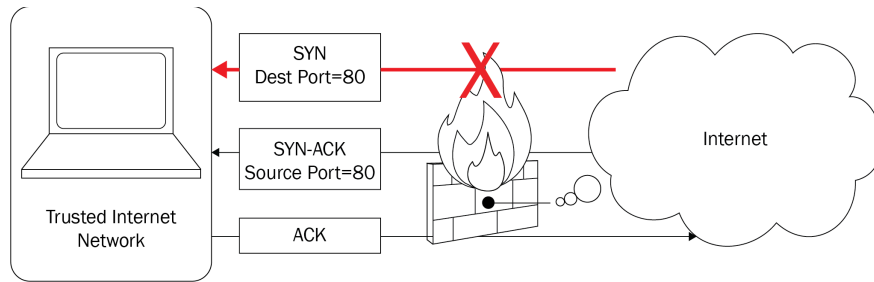


Figure 1.1 – A firewall with an ACL

It's important to note that a packet sniffer examines traffic but doesn't modify the contents in any way. It simply gathers the traffic for analysis as it travels across the network.

As you can see, packet sniffing and analysis have been influential for many years as elements of managing networks. However, the first step of analysis is to capture traffic, which we will explore next.

## Capturing network traffic

On today's networks, a **Network Interface Card (NIC)** will *only* monitor traffic that is addressed to that host. However, we can put the card into a state called **promiscuous mode**, which will allow the adapter to gather all the traffic that is on the network. Therefore, to capture and monitor all network traffic, the NIC must be in promiscuous mode.

On a Windows machine, you can check to see whether the interface card is in promiscuous mode by running the following command in PowerShell:

Windows PowerShell

Copyright (C) 2014 Microsoft Corporation. All rights reserved.

```
PS C:\Users\Admin> Get-NetAdapter | Format-List -Property
PromiscuousMode
```

```
PromiscuousMode : False
```

We use packet analysis to understand the characteristics of the traffic flow. Although you can conduct packet analysis during a live capture, it's common to capture traffic and save it for further analysis. Common steps to capture packets for analysis include the following:

1. Install Wireshark and the appropriate packet capture engine.
2. Launch Wireshark and select the capture options.
3. Start the capture and run until you capture 2,000–3,000 packets.
4. Stop the capture and save the trace file in the appropriate format.
5. Analyze the capture by studying one packet at a time, or as a complete capture.

In some cases, you might need to send a packet capture to the corporate or security analyst for further analysis.

Wireshark allows us to capture, display, and filter data live from a single or multiple network interface(s). In addition, you can examine pre-captured packets, search with granular details, and follow the data stream. As a result, packet analysis is advantageous as it helps you to understand the nature of the network. The following section outlines the many different individuals who can benefit from using Wireshark for packet analysis.

## **Recognizing who benefits from using packet analysis**

Nearly everyone can benefit from using packet analysis, including developers, network administrators, students, and security analysts. Let's look at each group and explore the benefits that can be reaped through packet analysis. We'll start with developers, as they can see how their program responds to requests on the network in real time.

### **Assisting developers**

Application performance issues can affect the bottom line, especially in a mission-critical situation. Developers diligently strive to produce elegant and efficient software. Prior to releasing an application, developers run functional and regression tests, along with stressing the server to ensure an optimized application.

Typically, developers test applications in a perfect environment, with high bandwidth and low latency. However, once the application moves from the local (or test) environment to the production network, clients may complain about the slow response times. The programmers will carefully check the application; however, on many occasions, they are unable to find anything unusual.

The developer must determine the reasons for the slow response times. Once further testing determines that it is not the application that is causing the issue, a packet analysis tool such as Wireshark can assist the developer.

By using packet analysis, the developer can uncover common problems in transmissions and help determine the root cause of the delayed response times. Problems such as delayed round-trip time and signs of congestion within an organization can occur in a network and impact response time.

Simply optimizing an application is not enough. All development life cycles should include checking what is happening on the network, as issues can affect overall performance.

In addition to developers, network administrators commonly use Wireshark to troubleshoot the network, as we will see next.

## Helping network administrators monitor the network

Network administrators use packet analysis to gain information about current network conditions. Wireshark can help identify errors and/or problems on the network that might require device tuning and/or replacement to improve overall performance.

A powerful feature in Wireshark is the ability to quickly detect issues in the capture. The network administrator can use both the expert system and the intelligent scroll bar, which color codes potential problems and helps with analysis, as we'll see in the next section.

### Expert system and intelligent scroll bar

Wireshark allows us to visualize issues while performing an analysis. The expert system categorizes various traffic conditions. It has a color code for each level that allows for easy identification of the general workflow and possible critical events:

- **Chat color (blue):** It provides information about typical workflows, such as a TCP window update or connection finish.
- **Note color (cyan):** It indicates items of interest, such as duplicate acknowledgments and TCP keepalive segments.
- **Warn color (yellow):** It indicates a warning, such as a TCP zero window or connection reset.
- **Error color (red):** It is the highest level as there might be a serious problem, such as a retransmission or a malformed packet.

The visual for the expert system is in the lower-left corner, as shown in the following screenshot:

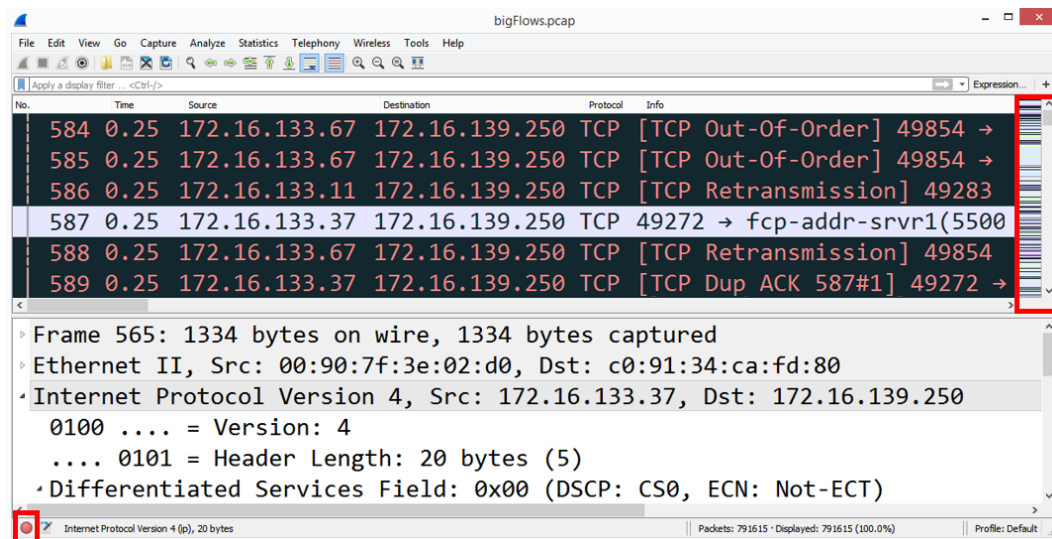


Figure 1.2 – Expert system and intelligent scroll bar

Wireshark also has an intelligent scroll bar, which provides a visual to detect issues. In the preceding screenshot, we can see a distinct coloring pattern on the right-hand side based on the coloring rules set in the application.

With the intelligent scroll bar, the administrator can easily click on a color band to zero in on a possible problem. Bear in mind that the intelligent scroll bar is only visible if the coloring rules are active; however, coloring rules are on by default.

Once any problems have been identified, you can subset traffic, add comments, save, and export the packet captures.

## Subsetting traffic, commenting, saving, and exporting

There are times when the network administrator might only want to share a small subset of traffic with other members of the team. Wireshark can subset large captures so that you can focus on the problem areas.

For example, in addition to data, a large packet capture will most likely have several different types of traffic, such as management and 802.11 control frames. You can easily apply a filter using the **...and not selected** option to exclude packets that are not relevant to the analysis.

Once you have created a smaller file, you can export the specified packets and save them in a wide variety of formats. Formats include the default PCAPNG, along with PCAP, Sun Snoop, DMP, and more.

Within the newly created subset, you can include comments. You can find comments in a couple of different ways:

- Select the comments icon that looks like a pad and pencil in the lower-left corner to add a comment for a single packet.
- Navigate to the **Edit | Packet comment** menu choice to add a comment for a single packet.
- Navigate to the **Statistics | Capture file properties** menu choice and include comments for an entire packet capture in the comment area at the bottom of the window.

**Note**

If you do add comments, then you must save the file in PCAPNG format, as not all file formats support the use of comments.

In addition to network administrators, students will gain valuable insight into what is actually happening on the network by using Wireshark to examine the headers and field values of the protocols.

## Educating students on protocols

Students can use packet analysis as a learning tool to better understand protocols. For example, when reviewing the **Dynamic Host Configuration Protocol (DHCP)**, a textbook will display the four stages of the process: **D**iscover, **O**ffer, **R**quest, and **A**cknowledge (**DORA**). Take a look at the following diagram:

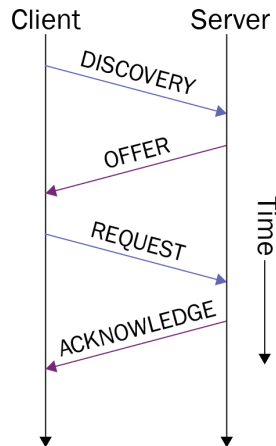


Figure 1.3 – The DORA process

While the preceding diagram displays each of the four-part transactions, it does not show the details of each part of the four-packet exchange.

In the following screenshot, we can see an actual DHCP transaction in Wireshark. In addition to this, the student can see the specifics of each exchange, including the transport protocol, the IP, the **Media Access Control (MAC)** addresses, and the DHCP header flags:

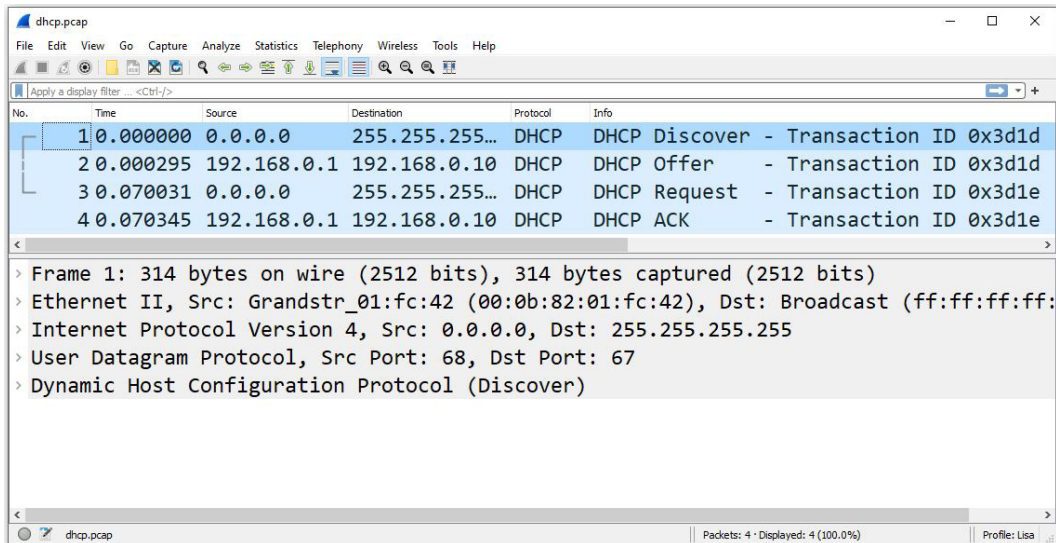


Figure 1.4 – The DORA process in Wireshark

By learning the normal behavior and purposes of common protocols, students will be able to troubleshoot any problems that might occur in the future.

As you can see, packet analysis has many benefits for many people. Because of the ability to really examine what is happening on the network, another key group that uses packet analysis is security analysts.

## Alerting security analysts to threats

To effectively discover potential problems, a security analyst must be an expert at packet analysis, as they use packet analysis in various ways:

- Determine whether there is anything unusual or suspicious about the traffic.
- Discover what transpired on the network when completing a forensic investigation.

Wireshark can help the security analyst better understand specific types of attacks so that they can craft firewall rules. To hone security analysis skills, the analyst can discover and download many PCAPs on various repositories. The Honeynet project, which is located at <https://www.honeynet.org>, is a great place to start. Navigate to the section on **CHALLENGES**, which offers many examples of forensic exercises to review and learn about many common threats found on today's networks.



Once you are on the **CHALLENGES** page, search for Challenge 12 - Hiding in Plain Sight, and read the details regarding the challenge. Then, to strengthen your analysis skills, download the files found at the bottom of the page and work through the questions. The answers can also be found at the bottom of the page, along with other files of interest.

Security analysts feel that Wireshark is a valuable tool as it provides insight into what is happening on the network. Because of its ability to have so much insight into what is happening on the network, Wireshark is also used by hackers for reconnaissance in order to gather and analyze traffic. This could be many times prior to an attack or during an active attack, which we will discuss next.

## Arming hackers with information

Malicious actors use packet analysis to sniff network traffic, with the goal of obtaining sensitive information. In addition, they can use the information gathered to launch an active attack.

When used as a precursor to an attack, hackers gather information during reconnaissance, which is also called **footprinting**. Let's take a look at a couple of ways in which hackers use Wireshark as part of a passive attack.

### Outlining passive attacks

Using Wireshark (or a similar tool), a malicious actor will try to obtain confidential information traveling through the network to achieve the following goals:

- **Footprinting and reconnaissance:** As a precursor to an active attack, malicious actors capture traffic to gather as much information about the target as possible. In addition to this, Wireshark can be used to gather additional information such as IP and MAC addresses, open ports and services, and possible defense methods that are in place.
- **Sniffing plain text:** Another use of packet sniffing is looking for passwords that are sent in plain text. In addition, protocols such as SNMP, HTTP, FTP, Telnet, and VoIP that are sent in plain text are susceptible to packet sniffers. Once captured, the protocol can expose information about the network and/or system(s).

An organization can defend against unauthorized packet sniffing in a couple of ways. There is anti-sniffer software that can detect sniffers on the network. However, one of the best ways to prevent data exposure is to use encryption. If someone captures the traffic, then the encrypted data will appear meaningless.

Next, we'll take a look at how hackers can also use Wireshark by actively sniffing and monitoring traffic as part of an **Address Resolution Protocol (ARP)** spoofing attack.

## Understanding active attacks

Malicious actors launch many different types of attacks on the network, such as **Denial of Service (DoS)**, phishing, or **Structured Query Language (SQL)** injection attacks. Next, let's take a look at another type of attack: an ARP cache poison attack.

### Poisoning the cache

ARP cache poisoning, also known as **ARP spoofing**, is used in a **Man-in-the-Middle (MitM)** attack. In order to understand why this is an effective attack, let's walk through the normal use of ARP on a LAN.

On a LAN, hosts are identified by their MAC (or physical) addresses. In order to communicate with the correct host, each device keeps track of all LAN hosts' MAC addresses in an ARP or MAC address table, also known as an **ARP cache table**.

Entries in the ARP or MAC address table will time out after a while. Under normal circumstances, when the device needs to communicate with another device on the network, it needs its own MAC address. First, the device will check the ARP cache and, if there is no entry in the table, the device will send an ARP request broadcast out to all hosts on the network.

The ARP request asks the following question: who has (the requested) IP address? Tell me (the requesting) IP address. The device will then wait for an ARP reply, as shown in the following screenshot:

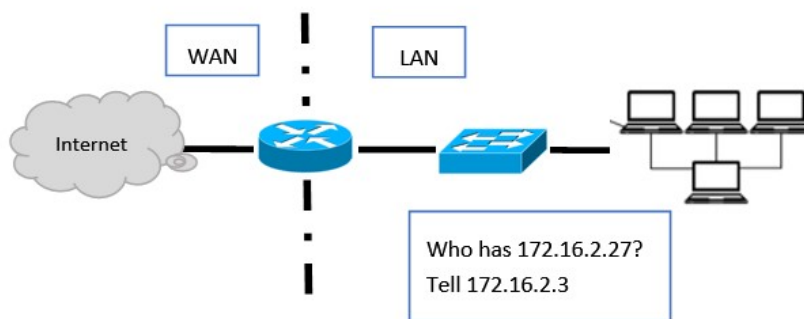


Figure 1.5 – ARP broadcast on a network

The ARP reply is a response that holds information on the host's IP address and the requested MAC address. Once received, the ARP cache is updated to reflect the MAC address.

In an ARP spoofing attack, a malicious actor will do the following:

1. Send an unsolicited ARP reply message that contains a spoofed MAC address for the attacker's machine to all hosts on the LAN.
2. After the ARP reply is received, all devices on the LAN will update their ARP (or MAC address) tables with the incorrect MAC address. This effectively *poisons* the cache on the end devices.
3. Once the ARP tables are poisoned, this will allow an intruder to impersonate another host to gain access to sensitive information.

ARP spoofing is done during a MitM attack, which allows a malicious actor to obtain traffic that is normally destined to go to another host.

In the following diagram, a bogus ARP reply was sent by the malicious actor, which then poisoned the cache in all of the network devices. All hosts on the network now think that 10.40.10.103 is at 46:89:FF:4C:57:BB, instead of 00:80:68:B4:87:EF, and will go to the attacker with the spoofed MAC address:

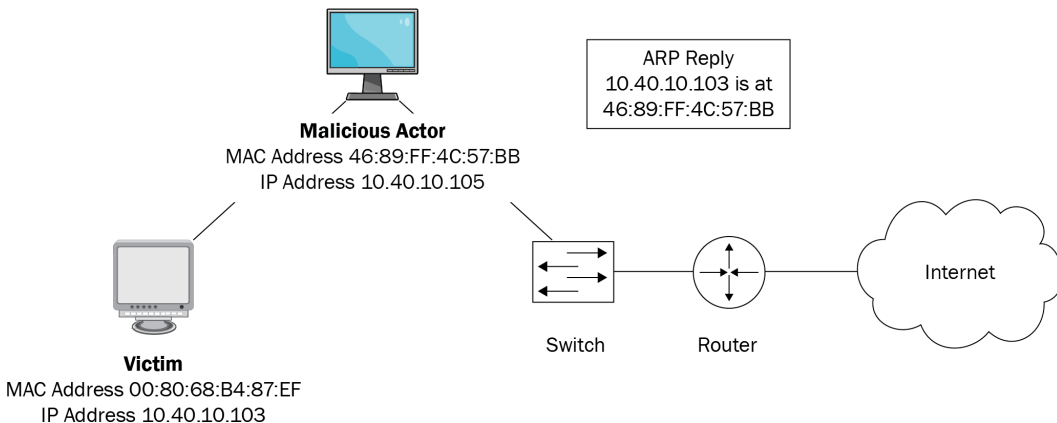


Figure 1.6 – An ARP spoof attack

The malicious actor will then use *active* sniffing to gather the misdirected traffic in an attempt to obtain sensitive information. In most cases, the traffic sent to the malicious actor is forwarded to the victim, who has no idea that anything is amiss.

Now we have seen the many individuals who can benefit from using packet analysis. In the next section, we will examine where packet analysis is most effective.

## Identifying where to use packet analysis

To conduct an effective packet analysis, the first step is to get a good capture. There are many places in which to conduct packet analysis, including on a LAN, on a host, or in the real world. Let's start with using packet analysis on a LAN.

### Analyzing traffic on a LAN

Today's networks are complex. An enterprise network provides connectivity, data applications, and services to the clients on the network, as shown in the following diagram:

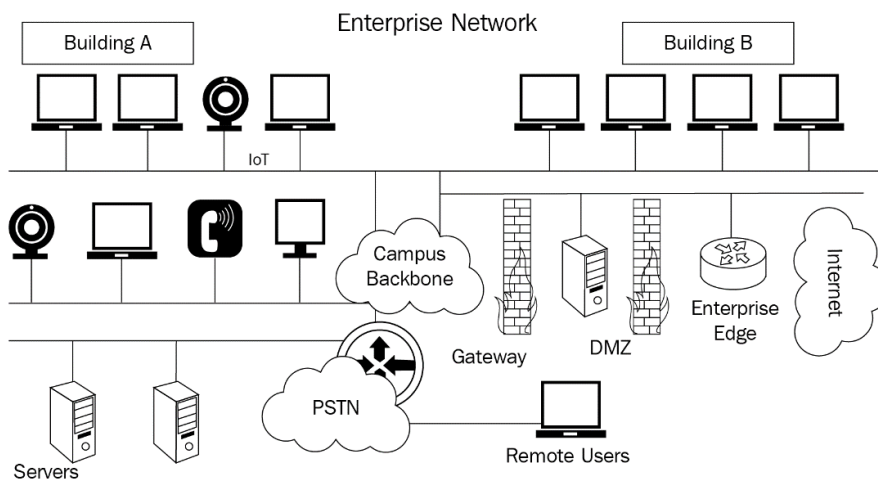


Figure 1.7 – A LAN

Most LANs are heterogeneous, with various operating systems such as Windows, Linux, and macOS, along with a mixture of devices such as softphones, tablets, laptops, and mobile devices. Depending on the business requirements, the network might include wide area network connectivity along with telephony.

To effectively use packet analysis, placement is the key. Not all traffic is created equally. Depending on placement, you might only capture a portion of the total network traffic. If the packet sniffer is on a host or end device, then it will be able to see the traffic on the segment's collision domain. If the sniffer is mirroring all traffic on a backbone, then it will be able to see all the traffic.

In certain instances, you might need to perform packet analysis on an individual host, such as a PC, to only monitor traffic destined to that host. In other cases, you might need to gather traffic on a switch to see the traffic as it passes through the switch ports.

## Sniffing network traffic

Packet analysis can be done on an individual host, within a switch, or in line with the traffic. The difference is as follows:

- If the protocol analyzer is installed on a client device attached to a switch, then the view of network traffic is limited. While sniffing traffic on a single switch port, you will only see broadcasts, multicasts, and your own unicast traffic.
- To see all the traffic on a switch, the network administrator can use port monitoring or **Switched Port Analyzer (SPAN)**. In some cases, you may be able to monitor within the switch, as Wireshark is built into the Cisco Nexus 7000 series and many other devices.
- Another option is to use a full-duplex tap in line with traffic. The tap makes a copy or mirror of the traffic, which is pulled into the device for analysis. If this option is used, then you might require a special adapter.

In addition to using packet analysis on a LAN or a host, packet analysis can be used in the real world to monitor traffic for threats.

## Using packet analysis in the real world

Packet analysis is used in the real world in many forms. One example is the **Department of Homeland Security (DHS)** EINSTEIN system, which has an active role in federal government cybersecurity. The United States government is constantly at risk of many types of attacks, including DoS attacks, malware, unauthorized access, and active scanning and probing.

The EINSTEIN system actively monitors the traffic for threats. Its two main functions are as follows:

- To observe and report possible cyber threats
- To detect and block attacks from compromising federal agencies

The EINSTEIN system provides the situational awareness that is necessary to take a proactive approach against an active attack. The intelligence gathered helps agencies to defend against ongoing threats.

As illustrated, packet analysis is effective in many locations. The following section provides guidance on what circumstances packet analysis will reap the most benefits under.

# Outlining when to use packet analysis

We use packet analysis in many ways. We can troubleshoot latency issues, test IoT devices, monitor for threats, and baseline the network. Let's evaluate some of this activity, starting with troubleshooting, which is a common use of packet analysis.

## Troubleshooting latency issues

Wireshark can be a valuable asset when troubleshooting issues on the network. There are many built-in tools designed to gather and report network statistics. We can analyze network problems and monitor bandwidth usage per application and process. The information gathered can help identify choke points and maintain efficient network data transmission.

Protocol analysis enables the network administrator to monitor the traffic on the network, unearthing problems that determine where performance can be fine-tuned. For example, if you suspect latency, you can obtain a capture in the area where you suspect trouble, and then run a Stevens graph, as shown in the following screenshot:

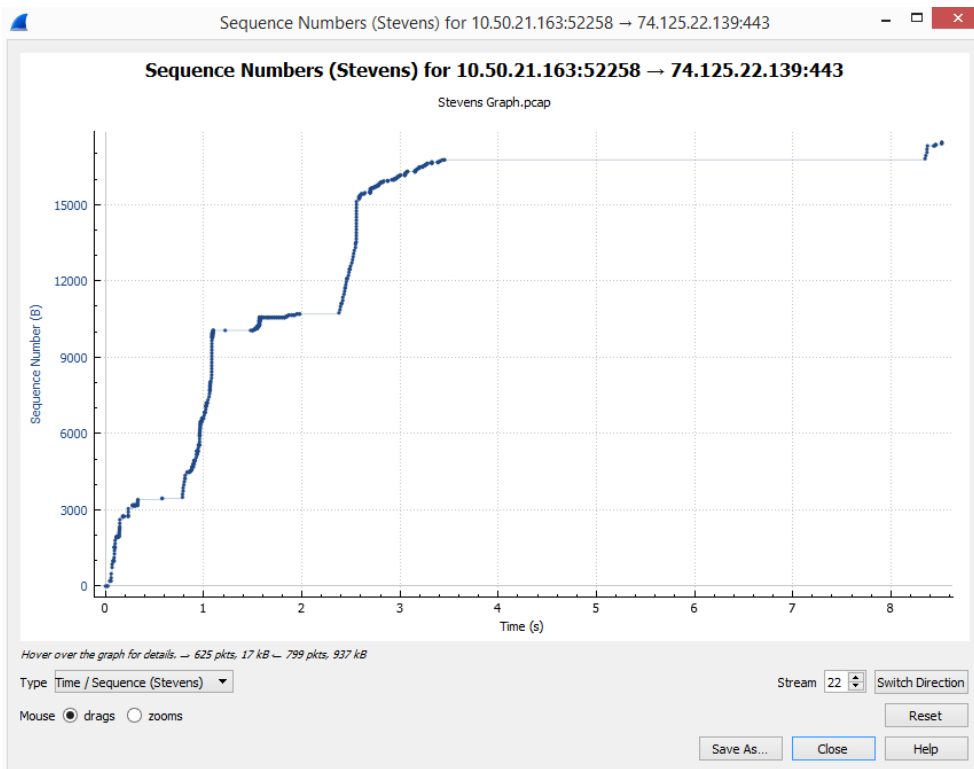


Figure 1.8 – A Stevens graph

Once the graph is complete, you can examine details that can highlight errors in the communication stream. For example, along the top of the graph, we see a straight line that continues for approximately four (4) seconds. The line represents a gap in transmission and may warrant further investigation.

In addition to troubleshooting the network, many are discovering how Wireshark can be a valuable asset in testing IoT devices prior to their implementation in an organization.

## Testing IoT devices

The IoT is a ubiquitous transformation of intelligent devices embedded in everyday objects that connect to the internet, enabling them to send and receive data. The IoT has several components: people, infrastructure, things, processes, and data. IoT has become a billion-dollar industry as consumers, along with industries, are seeing the benefits.

Even with all of the benefits, prior to connecting an IoT device to the network, it's best to run some tests. Using Wireshark can help you see what happens when you plug the device into the network. The following is a list of questions that Wireshark can help determine:

- How do the devices communicate once they are active? Do they phone home without being prompted?
- What information do they communicate? Are the username and password sent in plain text?

The only way you can understand the behavior of these devices is by plugging one in, capturing the data exchange, and analyzing the packet capture. The information obtained can provide valuable insights into the vulnerabilities of IoT devices.

Along with troubleshooting and testing, Wireshark can be instrumental in proactive threat assessment.

## Monitoring for threats

Monitoring for threats occurs in one of three ways:

- **Proactive:** Monitoring your systems and preventing threats by using a device such as an IDS.
- **Active:** Proactively seeking threats by conducting packet analysis and monitoring log files.
- **Reactive:** A system has fallen victim to an attack and the incident response team manages the attack, followed by a forensic exercise.

Wireshark can help the security analyst take an active role in monitoring for threats. While Wireshark does not provide any alerts, it can be used in conjunction with an IDS to investigate possible malicious network activity.

For example, while using snort (an open source IDS), the sensor produced the following alert, which could be an indication of malicious activity on the protected network:

```
DELETED WEB-MISC text/html content-type without HTML - possible  
malware C&C (Detection of a non-standard protocol or event)  
[16460]
```

This alert indicates that an infected host might be communicating with an external entity and sending information gathered on the network to a botmaster. The security analyst should take immediate action by running a capture in different segments of the network to identify and mitigate the threat.

Industries also see the value in using Wireshark for threat monitoring. For example, in the *Cisco Certified CyberOps Associate* certification prep course, students learn how to observe and monitor for unusual traffic patterns using Wireshark, as they hone their skills in preparing to work alongside cybersecurity analysts within a **Security Operations Center (SOC)**.

In order to determine what traffic is unusual, or to properly troubleshoot the network, you must be able to determine what constitutes normal network activity. This is achieved by conducting a baseline, as outlined in the following section.

## Baselining the network

A network baseline is a set of parameters that define normal activity. The baseline provides a snapshot of network traffic during a window of time using Wireshark or tshark.

Key characteristics for baseline can include utilization, network protocols, effective throughput, forwarding rates, and network latency. The network team can use the baseline for forecasting and planning, along with optimization, tuning, and troubleshooting.

The baseline process goes through several stages: plan, capture, save, and analyze. Once the baseline is complete, the network analyst can review the captured data in order to assess general performance for end-to-end communications. Baselining the network helps to gain valuable information regarding the health of the network, and possibly identify current network problems. In addition to this, subsequent baselining exercises can help predict future problems.

Whenever the installation of new equipment is planned, it's best to do a baseline prior to the change. After implementation, do another capture so you can identify possible issues in the traffic flow and then fine-tune the configuration.



As you can see, there are many ways we can use packet analysis to monitor, test, baseline, and troubleshoot. However, because of the ability to obtain sensitive information or as a precursor to an attack, packet analysis should only be done in the following circumstances:

- The network is your own, or you have received explicit permission to conduct packet analysis for security scans.
- It is completed during troubleshooting network connectivity issues.

In addition, consideration should be given to maintain the privacy of the data collected, and have a proper method to obtain, analyze, and retain any packet captures.

As outlined, we now know the many reasons to use packet analysis. Let's summarize by embracing Wireshark, which is one of the most powerful packet analysis tools available today.

## Getting to know Wireshark

In the late 1990s, Gerald Combs needed a tool to analyze network problems. Portable sniffers were available at the time, but they were costly. Gerald developed Ethereal with the help of some friends, and this later became Wireshark. It has been around for over 20 years and continues to evolve and improve over time.

Wireshark's strength is the ability to decode the captured bits into a readable format by using decoders or dissectors.

Dissectors provide information on how to break down the protocols into the proper format according to the appropriate RFC, or other specifications.

Wireshark can decode hundreds of different protocols. New dissectors are periodically added to the library. In addition, you can decode proprietary and specialty protocols by developing your own dissector.

Wireshark is compatible with many other sniffers and has a wide range of file formats for importing and exporting. Some of the other features include the following:

- Merge packet captures.
- Provide a detailed analysis of VoIP traffic.
- Create basic and advanced I/O graphs.

Wireshark can be installed on most OSes, including Windows, Solaris, Linux, and macOS.

After using Wireshark for any length of time, you can observe how it can help network administrators to understand traffic flows, troubleshoot performance problems, or conduct a network baseline.

## Summary

With the variety and amount of data that travels on today's networks, it's easy to understand why packet analysis using Wireshark should be in everyone's skill set. In this chapter, we took a brief look at how packet analysis began in the 1990s with the use of hardware sniffers. Fast forward to today, and we can see that packet analysis is used by nearly every device on the network to gather traffic, examine the contents, and then decide what action to take.

We learned how developers, network administrators, students, and security analysts can all benefit from using packet analysis. We examined the many places where we conduct packet analysis: on a LAN, on a host, and in the real world. In addition to this, we discovered how packet analysis has a variety of uses within today's networks, including troubleshooting, testing IoT devices, monitoring threats, and baselining. We can now appreciate how Wireshark is an exceptional open source software product that includes rich features and a variety of tools available to easily solve problems and analyze network traffic.

In the next chapter, we'll examine the Wireshark interface and review the phases of packet analysis. We'll also review the built-in **Command-Line Interface (CLI)** tools, such as `dumpcap` and `editcap`. Additionally, because Wireshark can be resource-intensive, we will learn how `tshark` (or terminal-based Wireshark) can provide a lightweight alternative to Wireshark.

## Questions

Now it's time to check your knowledge. Select the best response and then check your answers, which can be found in the *Assessments* appendix:

1. Packet analysis has been around in some form since the \_\_\_\_\_ as a diagnostic tool to observe data and other information traveling across the network.
  - A. 1950s
  - B. 1960s
  - C. 1970s
  - D. 1990s
2. Packet analysis is used in the real world in many forms. One is the DHS \_\_\_\_\_ system, which monitors for threats.
  - A. CARVER
  - B. Packet
  - C. EINSTEIN
  - D. DESTINY3
3. In the expert system, \_\_\_\_\_ provides information about typical workflows such as TCP window updates or connection finishes.
  - A. Note
  - B. Chat
  - C. Error
  - D. Warn
4. A \_\_\_\_\_ provides a snapshot of network traffic during a window of time using Wireshark or `ts shark`. Characteristics can include utilization, network protocols, and effective throughput forwarding rates.
  - A. Round Robin
  - B. DORA process
  - C. Baseline
  - D. WinCheck

5. Monitoring for threats occurs in one of three ways. \_\_\_\_\_ is when a system has fallen victim to an attack and the incident response team manages the attack, followed by a forensic exercise.
- A. Proactive
  - B. Reactive
  - C. Active
  - D. Redactive
6. When testing \_\_\_\_\_ using Wireshark, you will be able to determine how they communicate once active and see whether they phone home without being prompted.
- A. ACLs
  - B. Expert systems
  - C. IoT devices
  - D. IDSes
7. When obtaining an IP address, DHCP will go through a four-part transaction called the \_\_\_\_\_.
- A. Round Robin
  - B. DORA process
  - C. Baseline
  - D. WinCheck



# 2

## Using Wireshark

**Wireshark** is a protocol analyzer that can capture traffic and then present it in a human-readable format. In this chapter, you'll gain an insight into the overall functionality of Wireshark, and we'll see how to troubleshoot network traffic, monitor for security issues, and debug applications. We'll begin by taking a look at some of the features of the interface. We'll also cover where you can find a list of shortcuts, so you can confidently and quickly capture and analyze packets.

So that you can better appreciate the work that goes into Wireshark, we'll also review a list of the many authors that contribute to this project and help make Wireshark an exceptional tool. We'll also see where you can find links on the interface that provide helpful information on how to better use Wireshark.

In order to better understand the packet analysis process, we'll briefly review each of the phases involved: *gather*, *decode*, *display*, and *analyze*. We'll then review the built-in **command-line interface (CLI)** tools that complement Wireshark's basic functionality. We'll then finish with a closer look at `tshark`, a lightweight CLI application, which you can use when you need to capture traffic without the resource-intensive overhead of using Wireshark.

This chapter will address all of these by covering the following topics:

- Examining the Wireshark interface
- Understanding the phases of packet analysis
- Learning how to use the Wireshark CLI tools

## Examining the Wireshark interface

Wireshark offers a comprehensive framework for analyzing network traffic, and it performs well on most operating systems. The interface is streamlined and intuitive, with shortcuts and methods to make navigation easier and get you up and running with analyzing traffic. In this section, we'll discover how Wireshark presents information, along with where to find a list of keyboard shortcuts. We'll also take a look at the many authors who have made this application possible, and describe some ways you can obtain help and learn more about Wireshark.

Let's start with a brief look at the Wireshark interface.

### Streamlining the interface

When you first launch Wireshark, you will see a list of active interfaces. Some have a sparkline (or a moving graph symbol) next to the interface(s). When present, a sparkline represents actively exchanging data, and you can select that interface and begin capturing traffic.

As shown in the following screenshot, both the Wi-Fi and **VMware** network interfaces have active sparklines:

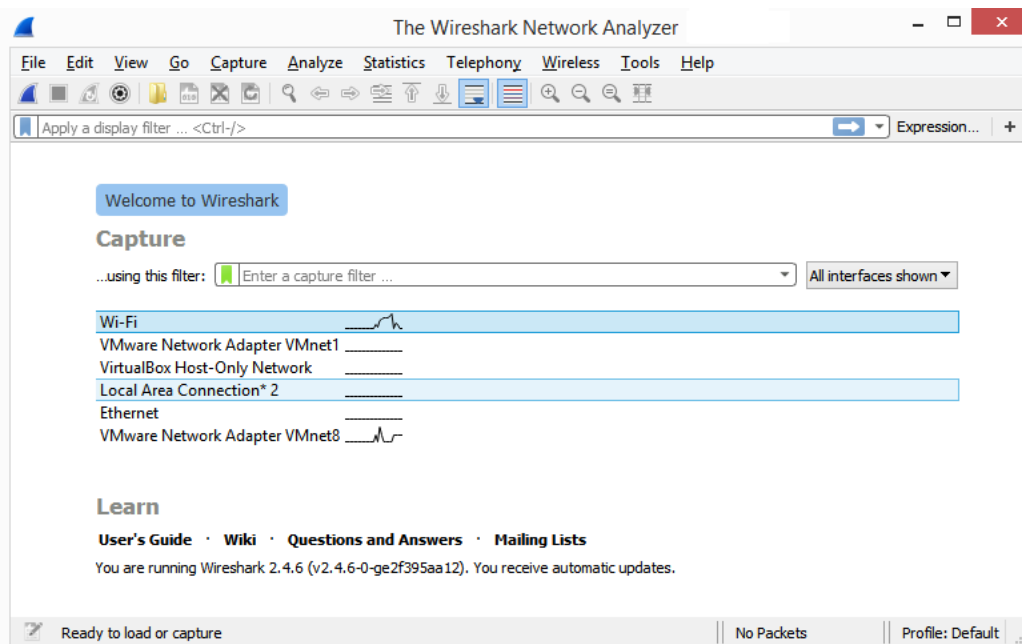


Figure 2.1 – The Wireshark interface

During analysis, Wireshark has many ways to improve your experience. For example, when working with a packet capture within Wireshark, you can easily add columns to the interface. Simply right-click on a value in the packet details area and select **Apply as Column**, as shown in the following screenshot. Selecting this option will add the `Flags` field as a column:

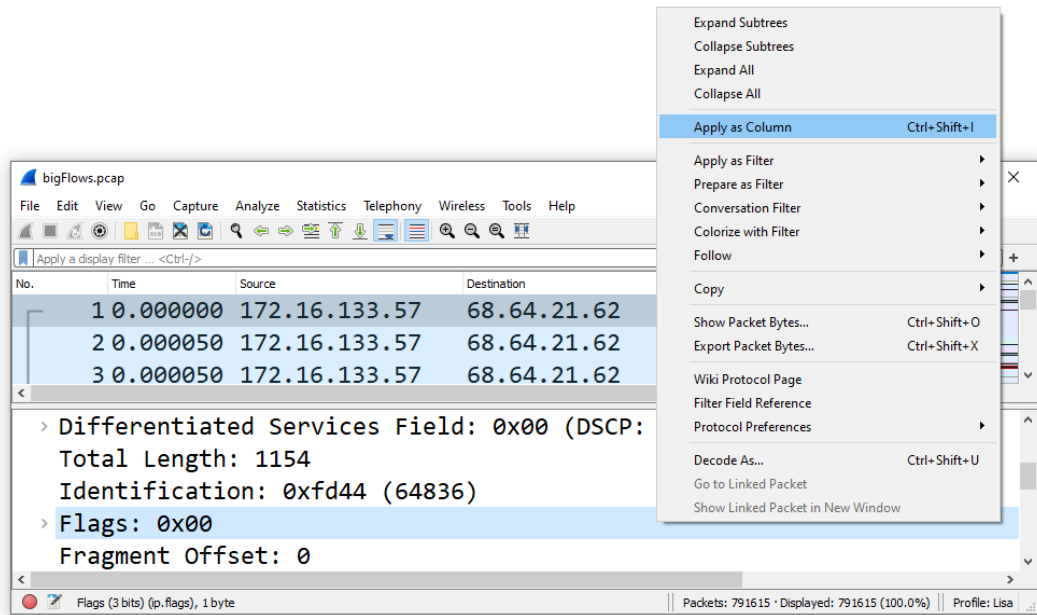


Figure 2.2 – Applying as a column



Wireshark also includes the **intelligent scrollbar**, which is on the right-hand side of the packet list. When the coloring rules are on, you can see an indication of problems and quickly go to trouble spots, which appear as dark black lines. This is highlighted by 1 in the following screenshot:

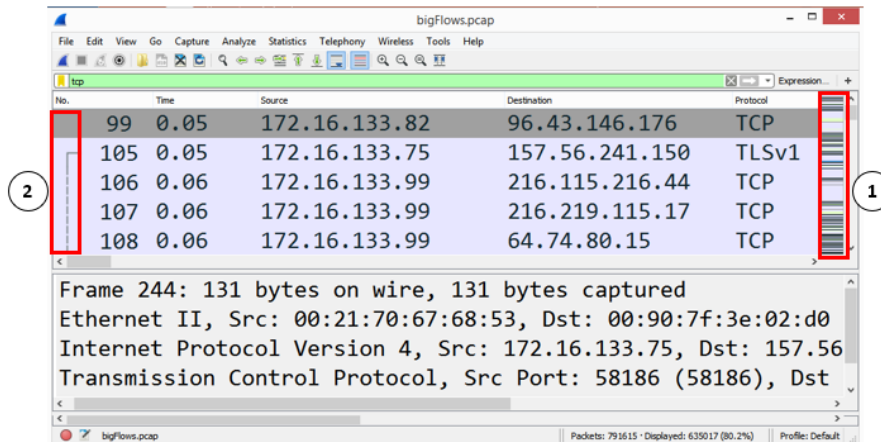


Figure 2.3 – The Wireshark interface with enhanced features

Some of the other features include the following:

- There are enhanced graphs – for example, flow graphs and I/O graphs that are easy to use.
- There are coloring rules, which are easy to create and edit.
- Related packets are displayed – you can simply click to see related packets (that is, the dotted line, as highlighted by 2 on the left-hand side of the preceding screenshot).
- It is capable of being translated into several different languages.

With millions of downloads per year, Wireshark has become a significant tool. It has proven to be flexible as an open source utility and encourages developers to add functionality, as well as improve the overall appearance.

Each new version improves the application. Improvements can include fixing a simple visual or display issue to more significant problems that can cause an application to crash, such as faulty dissectors. When you update Wireshark, take the time to read the notes, which will include information such as the following:

- What's new
- Bug fixes
- New and updated features
- New protocol support
- Updated protocol support
- New and updated capture file support
- New and updated capture interfaces support
- Getting help
- Frequently asked questions

In addition to the many benefits and features, Wireshark also provides a way to navigate using the keyboard. Let's investigate this concept next.

## Discovering keyboard shortcuts

Everyone has preferences as to how they interact with Wireshark. Some individuals prefer using a keyboard, as it's faster and more intuitive than using a mouse.

Wireshark has a list of keyboard shortcuts that can be found by selecting the **Help** menu choice, then clicking **About Wireshark** and selecting the **Keyboard shortcuts** tab, as shown here:

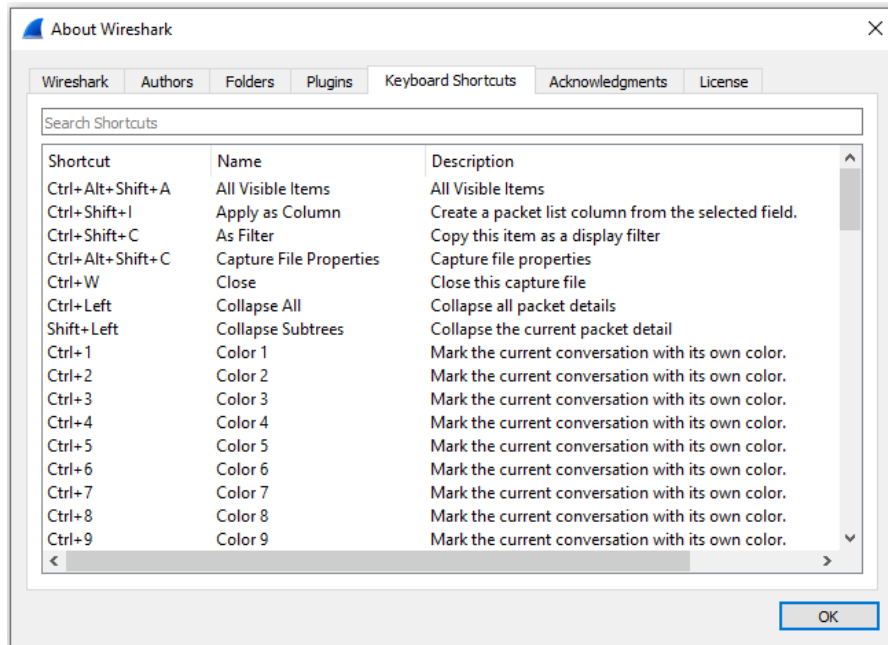


Figure 2.4 – Keyboard shortcuts

For example, when working with a packet capture, many times I'll select *Ctrl* + +, which will zoom in on the main text in the interface.

All of these improvements over the years have been possible because of the generosity of the open source community. The following section outlines how to see who is involved in creating Wireshark.

## Recognizing the Wireshark authors

Wireshark is open source and distributed under **GNU General Public License (GPL)**. Its success is attributed to the many developers that have contributed their time to it over the years.

When Gerald Combs and the original development team first released **Ethereal**, it had limited functionality and could decode less than six protocols. However, over the years, developers have added dissectors, functionality, and ease of use. As a result, Wireshark has become one of the most predominant network protocol analyzers in use today.

Many authors have contributed to the success of Wireshark by providing ongoing development and maintenance of the application. Some consistently jump in to add their expertise, and others contribute only when they need a specific protocol dissector.

Anyone can be involved, as there is plenty of documentation on how to add a basic dissector. If you do modify Wireshark to add a dissector or visual enhancement, share your work with the Wireshark team so that everyone can benefit.

To see a current list of Wireshark authors, go to the **Help** menu choice, click **About Wireshark** and select the **Authors** tab, as shown in the following screenshot:

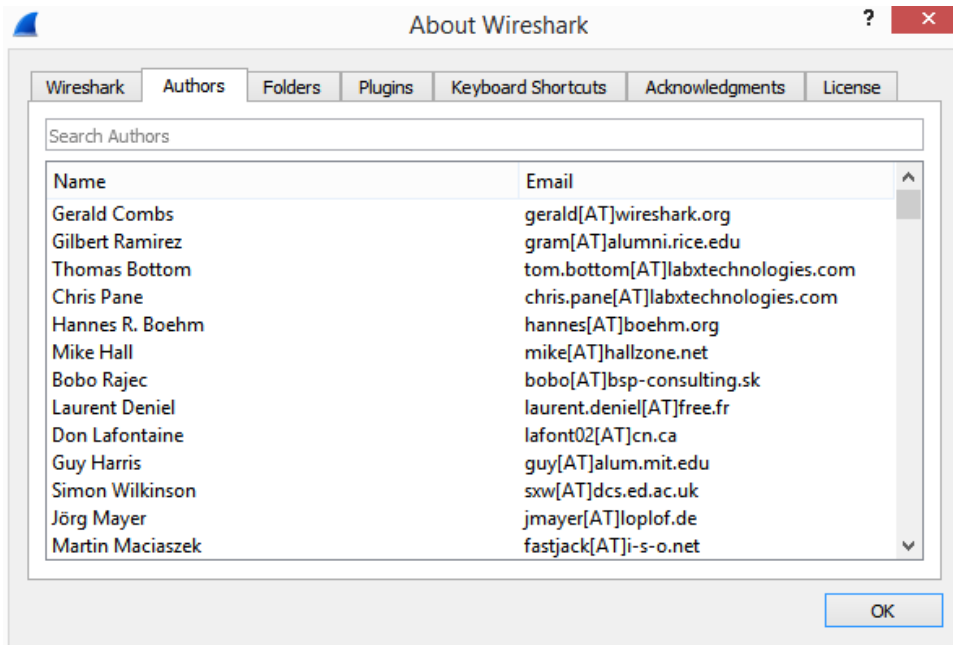


Figure 2.5 – The list of Wireshark authors

The Wireshark developer's goal today is to ensure functionality on **Windows**, **macOS**, and **Linux**. You can use Wireshark on any number of computers as necessary. All the source code is available under the GPL and can be found in the current Wireshark source code repository.

At times, you might need a little more information or help to complete a task. Within the interface, there is a list of links that will take you to external resources.

## Finding information

Across the bottom on the right-hand side of the Wireshark welcome interface, you will see the **Learn** label, where you will find the **User's Guide**, **Wiki**, **Questions and Answers**, and **Mailing Lists** links. Below the links, Wireshark will list what version you are running and whether or not you are receiving automatic updates.

So that you can better understand what goes on when you are using Wireshark, the next section walks through the process of analyzing packets.

## Understanding the phases of packet analysis

**Packet analysis** is the process of gathering traffic on the network, decoding it and dissecting the raw bits, and presenting it in a human-readable format for analysis, as shown in the following diagram:

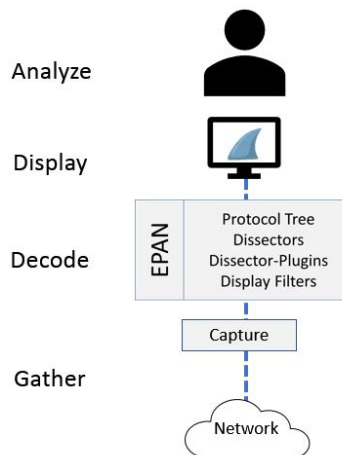


Figure 2.6 – The phases of packet analysis

Regardless of the software used, there are four main phases of packet analysis: *gather*, *decode*, *display*, and *analyze*. In this section, we'll review each of the phases, starting with the first step, *gather*, where we collect the data from the network.

## Gathering network traffic

When you launch Wireshark, a welcome screen displays a list of available network connections on your current device. In most cases, you will have more than one interface. To begin capturing immediately, you can select an active sparkline and begin the capture.

Alternatively, you can go to the **Capture** menu, and then go to the **Options** tab. This will open the following window:

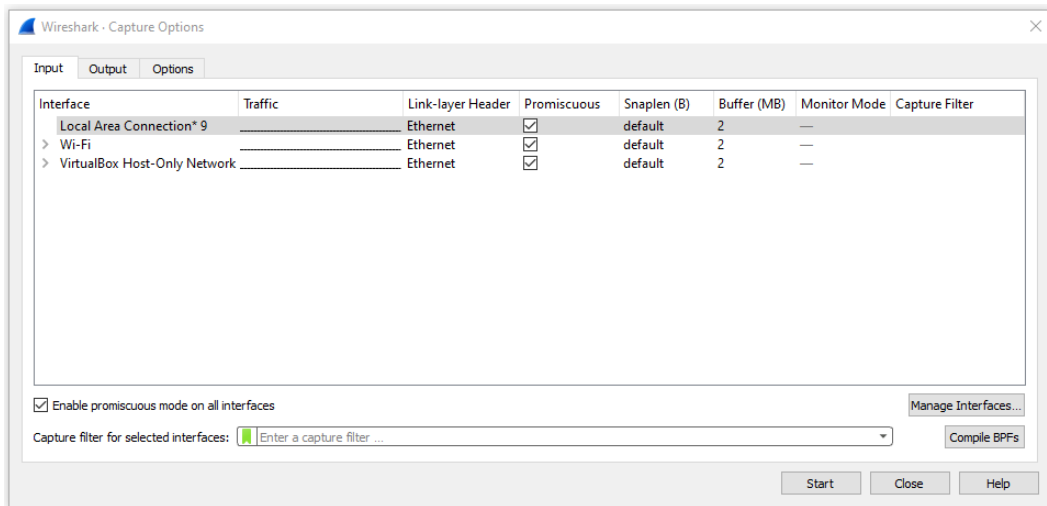


Figure 2.7 – The Capture Options window

Keep in mind that there are two key areas that will enable you to gather traffic: capturing in promiscuous mode and using a capture engine. Let's first discuss why it's important to enable promiscuous mode prior to capture.

## Capturing in promiscuous mode

When gathering traffic with Wireshark, you can capture on all interfaces. However, so that you can see all the traffic that is coming into the network interface card, make sure you select one of the following when in the **Input** tab of the **Capture Options** dialog box:

- Check the box next to the interface under the **Promiscuous** column header, as shown in the middle of the **Capture Options** dialog box.
- Check the **Enable promiscuous mode on all interfaces** box, as shown in the lower left-hand corner of the **Capture Options** dialog box.

After choosing an interface to listen on and placing it in promiscuous mode, the interface gathers up network traffic. To achieve this, a capture engine is required. Let's explore this concept next.

## Using a capture engine

Part of effectively capturing traffic is having an appropriate **packet capture (pcap)** engine installed. A pcap engine provides an **application programming interface (API)** that can capture traffic from the network so that it can be processed by the operating system.

As a result, when installing Wireshark, you will see a window appear that prompts you to install **Npcap**. Frequently, people aren't really sure if they should install Npcap. However, as shown in the following screenshot, Wireshark requires either Npcap or **WinPcap** to capture data:

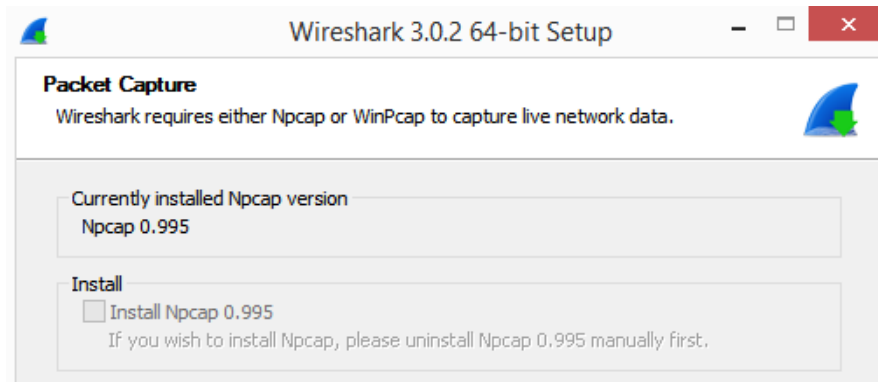


Figure 2.8 – The Wireshark prompt to install Npcap

If you don't install either Npcap or WinPcap, Wireshark won't run as expected.

### Note

While WinPcap is available for download, it is no longer actively maintained. As a result, Npcap is the preferred capture engine to use when installing Wireshark on a Windows machine.

Once you have gathered the traffic, the next step is to convert the raw bits and decode them into the proper protocol.

## Decoding the raw bits

Traffic enters a network interface card in binary form, one frame at a time. The following diagram illustrates converting bits into a human-readable format:

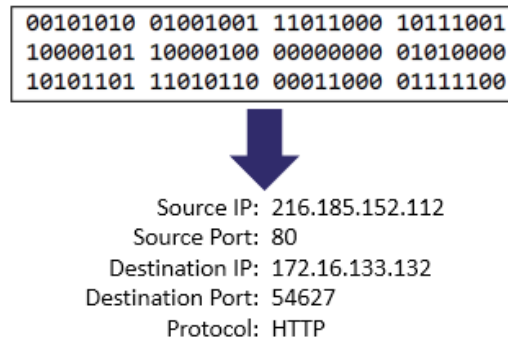


Figure 2.9 – Converting bits into a human-readable format

While in this phase, Wireshark uses the **Enhanced Packet Analyzer (EPAN)**, which decodes the bits into human-readable form.

## Stepping through the EPAN

Prior to 2006, Wireshark was called Ethereal. The name has since changed, however, the main core is the same. EPAN is the packet-analyzing engine for Wireshark that uses dissectors (also known as decoders). The dissectors provide information on how to recreate the protocols in the proper format according to the appropriate **Request for Comments (RFC)** or other specification. EPAN contains four main APIs, as shown in the following diagram:

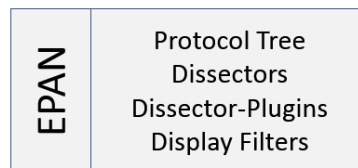


Figure 2.10 – The contents of the EPAN

The EPAN components are described as follows:

- **Protocol tree:** This displays the detailed analysis of a single packet.
- **Dissectors:** These provide information on how to break down the protocols into the proper format.



- **Dissector-plugins:** These use dissectors as separate functions.
- **Display filters:** These allow you to filter captured data.

In most cases, Wireshark is able to correctly identify and dissect the protocol. However, there are times when you will need to help Wireshark decode the protocol. You can achieve this by right-clicking the frame and selecting **Decode As...**, which will bring up the following window:

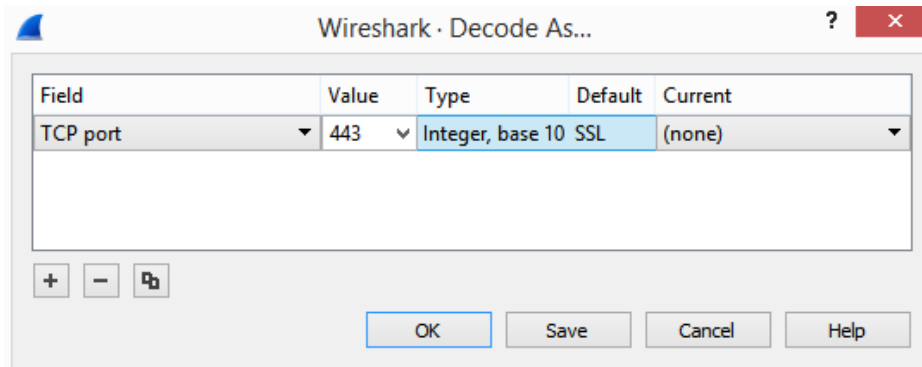


Figure 2.11 – The Decode As... window

Once in this window, you can modify the values to match the appropriate protocol. This function is very useful when protocols either don't have a dedicated port or they're running on a different port compared to usual. For example, you should use **Decode As...** when HTTP is running on port 8080 instead of port 80.

Once the bits have been converted into the proper format, the next step is to display the results in a human-readable format.

## Displaying the captured data

In Wireshark, along with many other packet analysis tools, there are many options to enhance your graphical experience. When you open a packet capture in Wireshark, the default layout for the main display is in three panels, as shown in the following screenshot:

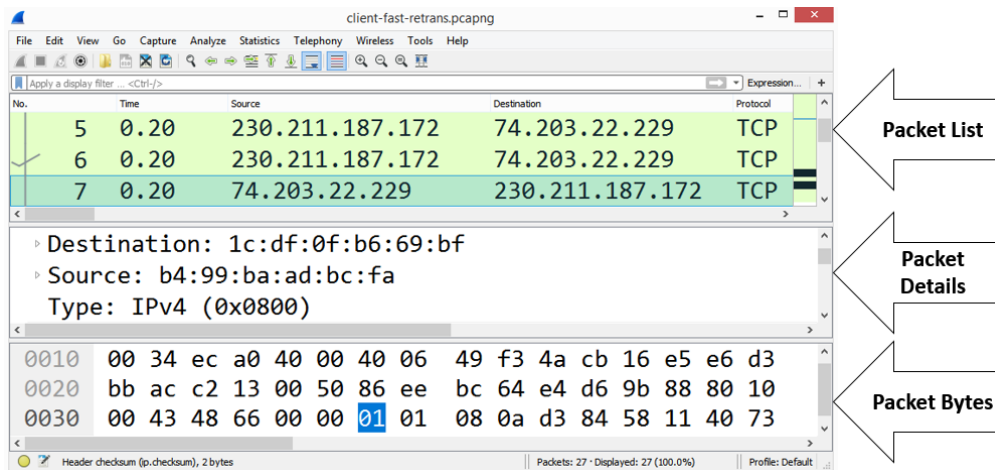


Figure 2.12 – Wireshark's main display, showing three panels

The three panels are defined as follows:

- **Packet list:** This is a list of all the captured packets, where each line represents a single packet. If there are too many packets to fit in the panel, the user can use the scroll bar on the right to navigate through the capture.
- **Packet details:** This displays the details of a single packet and includes the protocols and field values. It also displays Wireshark-specific hints. For example, when examining a **Transmission Control Protocol (TCP)** header, you will see [Stream index: n] listed below the source and destination ports. However, there is no field value called *stream index*. A *stream* is a communication between two endpoints that comprises the following:
  - Endpoint A's socket [IP address and port]
  - Endpoint B's socket [IP address and port]

To help keep track of all the streams, Wireshark lists each stream in a TCP header. This is shown in the following screenshot:

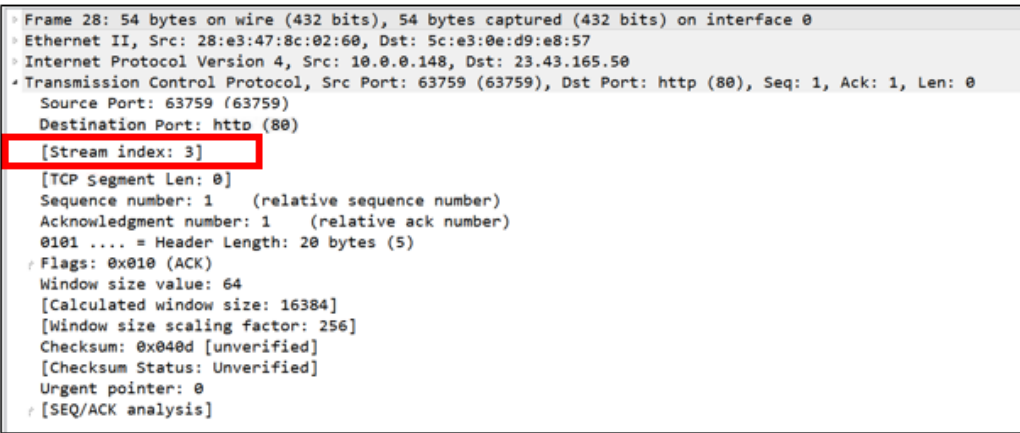


Figure 2.13 – A packet details panel

- **Packet bytes:** This is a hexadecimal representation of a single packet, as shown in the packet details panel. Any plain text data is displayed on the right-hand side, as shown in the following screenshot:

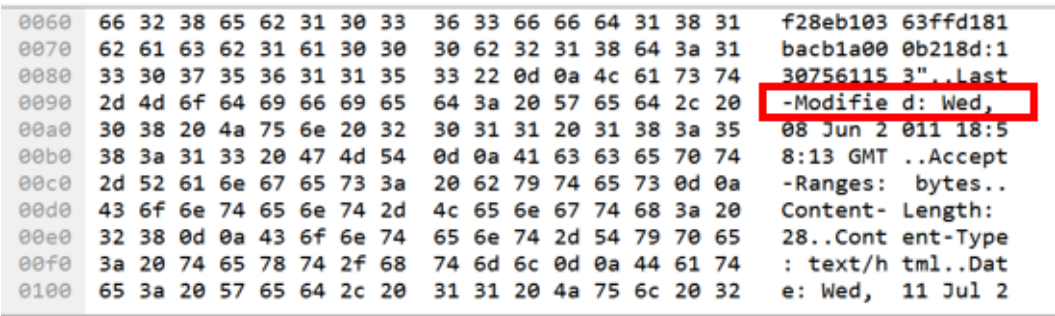


Figure 2.14 – A packet bytes panel

**Note**

The default view for the packet bytes panel is hexadecimal. However, you can change the view to bits by right-clicking anywhere in the panel and selecting **Show bytes as bits**.

While the default layout is three panels, you can personalize this view at any time.

## Changing the layout

To change the appearance of the main window, go to the **Edit** menu choice and then select **Preferences**. Once in the **Preferences** window, select **Layout**, where you can change your layout to one of many different configurations, as shown here:

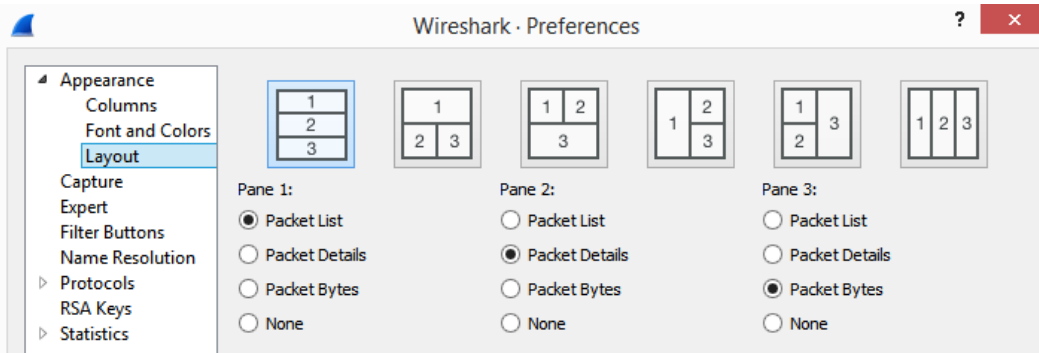


Figure 2.15 – Preferences | Layout

After Wireshark displays the results of the capture, we then move to the final stage of packet analysis: *analyze*.

## Analyzing the packet capture

The analysis phase can be in either real time or by using a pre-captured file to troubleshoot and examine the captured traffic. Analysis can be done by using the many built-in tools in Wireshark, which include the following:

- Filtering traffic to display specific types of flows, such as DNS or HTTP traffic
- Searching for specific packets, for example, `tcp.port == 443`
- Turning on the coloring rules or using the expert system to easily spot problems
- Following the stream to see the details of a single conversation
- Doing a deep packet analysis of individual frames and examining the field values of each of the headers

Wireshark's **Statistics** menu choices can range from basic information features such as **Capture File Properties** to more detailed analysis such as **Conversations**, **Flow Graphs**, and **Stream Graphs**.

Along with the tools within Wireshark, you can also subset the data to share a smaller file with coworkers and add comments to the entire file or within an individual frame.

Although the GUI is easy to use and understand, the Wireshark interface, with all its enhancements, coloring rules, and shortcuts, can be resource-intensive. As a result, it's best to become familiar with some of the CLI tools, which we will cover in the next section.

## Using CLI tools with Wireshark

Wireshark has several CLI tools that complement its basic functionality and will allow you to perform several tasks, such as editing, splitting, and manipulating packet captures. The following table is a summary of some of the tools available:

Tool	Function
<code>dumpcap</code>	A program used to capture network traffic
<code>editcap</code>	Can edit and subset capture files
<code>capinfos</code>	Provides basic statistics on the capture file
<code>mergcap</code>	Can merge multiple capture files into one
<code>text2pcap</code>	Converts a hexdump of <b>ASCII</b> (short for <b>American Standard Code for Information Interchange</b> ) packets into a capture file
<code>tshark</code>	A lightweight CLI equivalent of Wireshark

Table 2.1 – Wireshark's built-in CLI tools

All of the CLI tools are baked into Wireshark, however, they are also available to use as a lightweight solution when working work with packet captures.

Next, let's take a look at `tshark`, which is a great alternative to use when you need to conserve resources.

## Exploring tshark

Part of the Ethernet development process included **Terminal Ethernet (Tethereal)**, which was a CLI tool. Tethereal was later renamed **Terminal Wireshark (tshark)**.

To capture using `tshark` on a Windows machine, go into the CLI and build a command as the following example shows:

```
C:\Program Files\Wireshark>tshark -i "ethernet 2" -w Test-
Tshark.pcap -a duration:10
```

Keep in mind, if you have multiple interfaces, you will need to find which interface is active by using `ipconfig`.

**Note**

Commands on a Windows machine are not case-sensitive.

To run the `tshark` example, follow these steps:

1. Begin the command with `tshark`.
2. Identify the interface by using `-i`, then entering the interface name.
3. To write to a file, use `-w`, then enter the filename and path. Make sure you add the extension.
4. To set the duration, use `-a`, which is capture auto-stop, and set the duration in seconds.
5. Press *Enter* to begin the capture.

When complete, locate and open the `pcap` file in Wireshark. If you don't send the output to a file, you will see a list of packets captured on the screen:

```
C:\Program Files\Wireshark>tshark -i "wi-fi" -a duration:10
Capturing on 'Wi-Fi'
  1  0.000000 2603:1036:404:f2::2 → 2601:98b:4402:20cd:b819:45e2:8cb1:bf75 TL
Sv1.2 Application Data
  2  0.034029 2601:98b:4402:20cd:b819:45e2:8cb1:bf75 → 2603:1036:404:f2::2 TC
P 59203 → https(443) [ACK] Seq=1 Ack=86 Win=66 Len=0
  3  0.132176 2a01:111:f100:2002::8975:2da8 → 2601:98b:4402:20cd:b819:45e2:8c
b1:bf75 TLSv1.2 Application Data
  4  0.156495 2601:98b:4402:20cd:b819:45e2:8cb1:bf75 → 2a01:111:f100:2002::89
75:2da8 TCP 59576 → https(443) [ACK] Seq=1 Ack=70 Win=63 Len=0
  5  1.127444 fe80::5ee3:eff:fed9:e857 → ff02::1      ICMPv6 Router Advertise
ment from 5c:e3:0e:d9:e8:57
  6  1.200726 10.0.0.59 → 10.0.0.148      TCP 49627 → 59655 [PSH, ACK] Seq=1
Ack=1 Win=4096 Len=314
  7  1.208793 10.0.0.148 → 10.0.0.59      TCP 59655 → 49627 [PSH, ACK] Seq=1
Ack=315 Win=64 Len=314
  8  1.209189 10.0.0.148 → 10.0.0.59      TCP 59655 → 49627 [FIN, ACK] Seq=31
5 Ack=315 Win=64 Len=0
  9  1.216924 10.0.0.59 → 10.0.0.148      TCP 49627 → 59655 [ACK] Seq=315 Ack
=315 Win=4091 Len=0
```

Figure 2.15 – Output from running `tshark`

The Wireshark documentation found at [https://www.wireshark.org/docs/wsug\\_html\\_chunked/ChCustCommandLine.html](https://www.wireshark.org/docs/wsug_html_chunked/ChCustCommandLine.html) lists a number of switches to use with `tshark`. Many are the same options that you can use while using the GUI, such as adding filters and specific field values, and include the following:

Output	
<code>-w &lt;outfile -&gt;</code>	Set the output filename (or <code>-</code> for <code>stdout</code> )
<code>-i &lt;interface&gt;</code>	Enter the name of the interface (the default is the first non-loopback), for example, <code>"wi-fi"</code> .
Capture stop conditions	
<code>-c &lt;packet count&gt;</code>	Stop after $n$ packets (the default is infinite)
<code>-a &lt;autostop cond.&gt; ...</code>	<ul style="list-style-type: none"> <li><code>duration:NUM</code>: Sets the duration to stop the capture after <code>NUM</code> seconds</li> <li><code>filesize:NUM</code>: Stops the capture once it hits <code>NUM</code> KB</li> <li><code>files:NUM</code>: Sets the capture to stop after <code>NUM</code> files</li> </ul>

Table 2.2 – tshark options

The `tshark` CLI tool can evaluate any protocol that can be dissected by Wireshark. If you aren't sure if a specific protocol is supported, you can check the protocol list in the **Preferences** dialog box.

## Dissecting protocols

Wireshark is loaded with hundreds of protocols to dissect, with new protocols added every year. To see whether a specific protocol is supported, go to **Edit**, then **Preferences** (as shown in the following screenshot), and scroll to see the desired protocol:

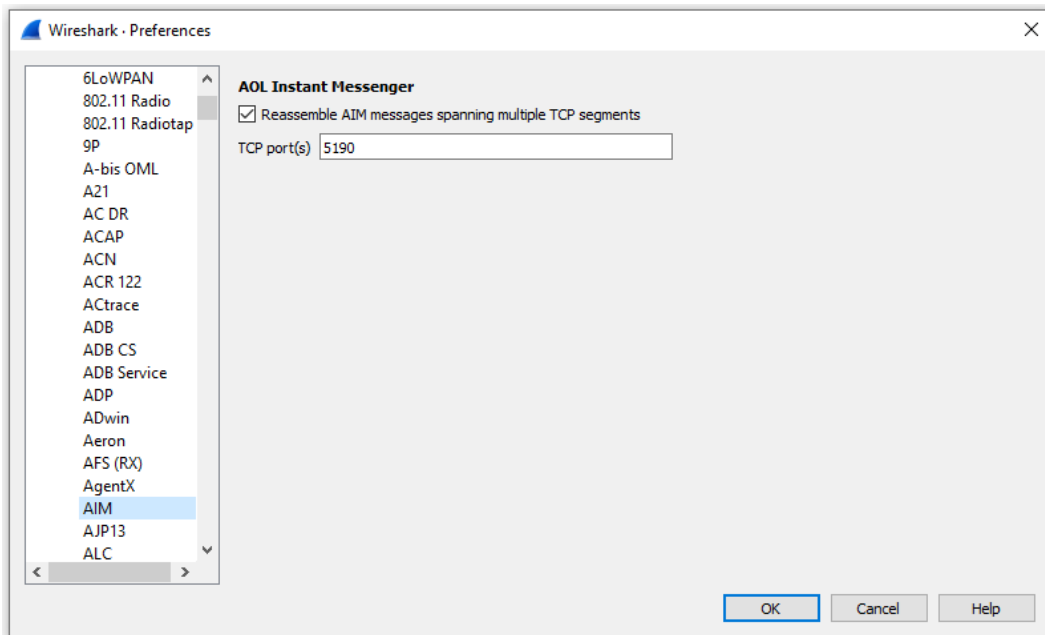


Figure 2.17 – The Wireshark Preferences dialog with the list of protocols

Once in the protocol **Preference** window, you can modify the way Wireshark dissects the protocol. For example, in the preceding screenshot, AOL Instant Messenger uses TCP port 5190, which can be changed if another port is to be used.

## Summary

In this chapter, we took a brief look at the Wireshark interface. We learned how you can quickly begin capturing by clicking on a sparkline, easily add columns to the interface, and use the intelligent scrollbar coloring to identify trouble spots. You can now appreciate how each new version of Wireshark improves with the help of the many developers constantly updating the software. We also covered where you can find keyboard shortcuts and additional information.

We then explored the phases of packet capture, from gathering network traffic to processing it into a human-readable format for analysis. Finally, we saw how because Wireshark can be resource-intensive, it's sometimes better to use a CLI tool such as `tshark`, a lightweight application for capturing packets.



In the next chapter, we will explore downloading and installing Wireshark on various operating systems such as Windows, macOS, and Linux. We'll then take the time to explore the different capture engines, and we'll walk through a Windows installation and compare the different download options. Finally, we will look at the various resources that are available at <https://www.wireshark.org/>.

## Questions

Now, it's time to check your knowledge. Select the best response, then check your answers, which can be found in the *Assessments appendix*:

1. A(n) \_\_\_\_\_ (or a moving graph symbol) represents actively exchanging data. If present, you can select that interface and begin capturing traffic.
  - A. GPL
  - B. sparkline
  - C. API
  - D. Wiki
2. So that you can see all the traffic that is coming into the network interface card, make sure the card is in \_\_\_\_\_ mode.
  - A. sparkline
  - B. RFC
  - C. generous
  - D. promiscuous
3. A \_\_\_\_\_ engine provides an API to capture traffic from the network before the traffic is processed by the operating system.
  - A. CACE
  - B. pcap
  - C. tcap
  - D. capinfos

4. \_\_\_\_\_ provides information on how to break down the protocols into the proper format, according to the appropriate RFC or other specification.
  - A. Protocol tree
  - B. Dissector filters
  - C. Capinfos
  - D. Dissectors
5. Wireshark has several CLI tools that complement the basic functionality, \_\_\_\_\_ can merge multiple capture files into one.
  - A. tshark
  - B. capinfos
  - C. mergecap
  - D. text2pcap
6. If you don't install the appropriate \_\_\_\_\_, such as Npcap or WinPcap, Wireshark won't run as expected.
  - A. capture engine
  - B. captap
  - C. EPAN
  - D. byte wrangler
7. Prior to running tshark on a Windows machine, you'll need to identify which interface is active using the \_\_\_\_\_ command.
  - A. netstat
  - B. ipconfig
  - C. ping
  - D. powercfg



# 3

## Installing Wireshark

To start capturing and analyzing packets, you'll first have to download and install Wireshark on your computer or laptop. The good news is that Wireshark can be installed on a variety of different platforms. In this chapter, we'll start by learning how Wireshark provides support for different **Operating Systems (OSes)**. We'll also review the importance of a capture engine, and how it is a critical component when gathering network traffic.

When installing Wireshark on a PC, you'll be presented with a variety of choices. So that you can confidently navigate the installation and make the correct selections, we'll review the different options. Once installed, you can then begin capturing and analyzing traffic. Wireshark is open source with constant enhancements and improvements. You'll discover many online resources where you can see the latest news and updates that can help improve your workflow, along with several download options.

This chapter will cover the following:

- Discovering support for different OSes
- Comparing the different capture engines
- Performing a standard Windows installation
- Reviewing the resources available at [Wireshark.org](https://www.wireshark.org)

## Discovering support for different OSes

Wireshark is an open source packet analysis tool developed as a cross-platform application. Wireshark uses the Qt **Graphical User Interface (GUI)** library, which can run on a variety of hardware and software platforms with little or no modification to the underlying code. For most OSes, it can be installed with ease using a standard installation.

Wireshark is capable of working with most modern default system capabilities. For example, a system with a 64-bit AMD64/x86-64 processor along with ample memory and disk space will perform well with minimal problems. However, gathering captures that are larger than a few hundred **megabytes (MB)** may consume too much memory and cause the system to crash.

In this section, we'll outline how Wireshark can be used on either Microsoft Windows, Linux, or macOS. Let's start with how developers provide support for the Windows OS family.

## Using Wireshark on Windows

Of all the OSes today, Windows has the highest market share. Wireshark will run on most Windows systems, as it natively interacts with the Windows **Application Programming Interface (API)**. Currently, Wireshark can run on Windows 8.1 and above, along with Server 2012 R2, 2016, and 2019.

### Important Note

Wireshark is no longer supported on older Windows OSes. Because of this, if you are still using Wireshark on an older version of Windows, it may not perform as expected.

Now that we have discussed how Wireshark operates in a Windows environment, let's take our discussion further and explore how Wireshark functions on a Unix platform.

## Running Wireshark on Unix

In addition to standard Windows options, Wireshark can be installed and run on several Unix systems, such as Oracle Solaris 11, FreeBSD, and NetBSD. For those and other OSes that do not have a standard install, you can access Wireshark packages for most platforms by going to <https://www.wireshark.org/download.html>. Once there, scroll to third-party packages where you can see the options available for many other platforms.

In addition to Unix, the following section outlines how Wireshark provides support for macOS.

## Installing Wireshark on macOS

On Wireshark.org, you will find an installation for macOS 10.13 and later. You can download and install Wireshark on a macOS just like any other OS. The installer will guide you through the process in much the same manner as installing on a Windows machine. This has made Wireshark more user-friendly to the growing population of macOS users.

### Important Note

Prior to capturing packets on your macOS, you may need to install **Change mode Berkeley Packet Filter (ChmodBPF)**, which will allow Wireshark to access the capture devices.

Because of the widespread use of Linux, the following section provides information on how Wireshark is also easy to install and use on a Linux machine.

## Deploying Wireshark on Linux

Wireshark is supported on many Linux platforms, including Ubuntu, Debian, SUSE, and Red Hat. However, you may run into errors during the build and installation phases.

Common problems that can arise include the following:

- You don't have the necessary development package on your system.
- The development package is outdated.
- You are missing the libpcap capture engine.

If you are able to install Wireshark, there may be an issue with capturing packets and the system might display an error, as shown here:

```
No interface can be used for capturing in this system with the
current configuration. (Couldn't run /usr/bin/dumpcap in child
process: Permission denied)
```

If you see this error, this is most likely because `dumpcap` needs elevated permissions and advanced configuration is required to capture traffic. Running the following command often resolves this issue:

```
dumpcap setuid root
```

In addition, running Wireshark as a root user can also cause problems. Linux systems defend themselves against what is perceived as risky behavior, which can cause harm to the OS. As a result, Wireshark may not run while in root mode, and further configuration may be necessary to make this possible.

As new OSes enter the market, it's nice to know that Wireshark evolves to keep up with the changing demands in today's networked environment.

The following section outlines how versatile Wireshark is when working with a variety of OSes.

## Working with Wireshark on other systems

Wireshark can be used on network devices and servers to monitor and analyze traffic. For example, several Cisco devices are Wireshark-capable and provide the network specialist with comprehensive documentation on best practices while capturing network traffic. Some guidelines while on a Cisco networking device include the following:

- Prior to capture, make sure the CPU is not overburdened and that you have at least 200 MB of free memory.
- When possible, limit captures by either size or duration.

In addition to Cisco, IBM provides extensive documentation on how to obtain a Wireshark trace file. When done, technicians are encouraged to send their trace files to IBM support for further analysis.

Many other companies have found the value of packet analysis using Wireshark and have integrated the software within their respective products.

If you need to become familiar with working with Wireshark on a Linux machine, then there are other options. The following section provides guidance on how to easily download and begin using a premade Linux **Virtual Machine (VM)**.

## Downloading premade virtual images

A VM can be used to get a feel of how to use Wireshark on a Linux OS for training or testing purposes. Premade virtual images are available at <https://www.osboxes.org/>, where you'll find you can choose from many Linux OSes.

In some cases, you can find a Linux OS that has Wireshark preinstalled and ready to run. Examples include BackBox and Kali Linux, which include not only Wireshark but other tools used during penetration testing.

Using the premade images for testing on a production network is not practical, as the VM doesn't have the same visibility as the host. However, using a VM is beneficial when learning about how to use Wireshark on a Linux OS in a classroom setting for training or testing purposes.

Another option when using Wireshark is to capture using a portable **application (app)**.

## Utilizing a portable app

Wireshark has had an option to download Windows PortableApps for a 32-bit operating system for many years. As time has passed, developers have now made Windows PortableApps for a 64-bit OS available.

The portable app option makes it easy to run Wireshark on portable media. Instead of installing on a traditional computer, you can use the portable app on a thumb drive and/or a cloud drive, such as Google Docs.

Once you download, install, and launch the portable app, it will appear the same as using the standard installation. For example, I selected `WiresharkPortable64_3.6.0.paf` and uploaded the app to my cloud drive. Once installed, I opened the app, as shown in the following screenshot:

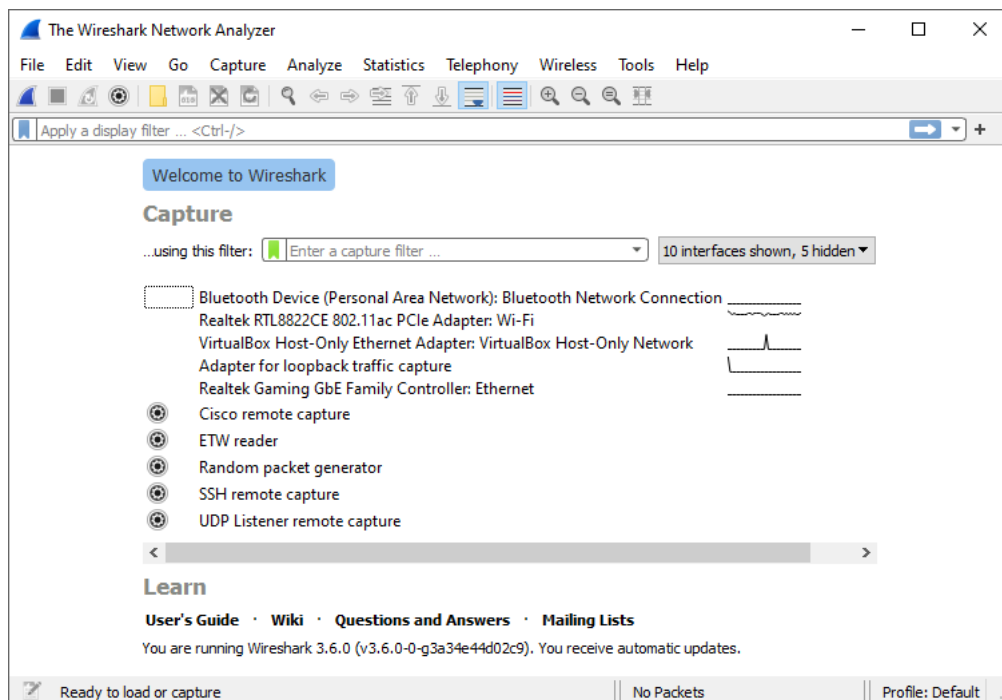


Figure 3.1 – Using the Wireshark portable app



Using the portable app makes it convenient to run Wireshark on a Windows system from either a removable drive or cloud storage. Once you are done, you can disconnect without leaving any data.

**Important Note**

Even though you don't have to install Wireshark on the host, you will need to install Npcap in order to capture packets.

Regardless of what OS Wireshark runs on, the OS will need a way to gather or capture the raw bits from the network. A capture engine pulls or captures the network traffic so that it can be sent to the OS for dissection and analysis. The next section provides a comparison of the capture engines available today.

## Comparing different capture engines

To effectively capture and analyze traffic, there must be a way to gather the raw traffic from the network before being processed by the OS. A **Packet Capture (PCAP)** engine provides an API to capture traffic. Wireshark uses one of several capture engines, such as libpcap, WinPCap, and Npcap. Let's begin with outlining libpcap.

## Understanding libpcap

Libpcap is a capture engine that was originally developed for a Unix-like OS. Libpcap is incorporated into tcpdump, Snort, and other packet analyzers to grab packets as they come off the network interface.

Wireshark and tshark work with libpcap and generate pcapng files by default. Libpcap and tcpdump are developed and maintained at <http://www.tcpdump.org/>. In the late 1990s, a version of libpcap was adapted for Windows called WinPcap, as we'll discuss next.

## Examining WinPcap

WinPcap is a capture engine that uses drivers specific to a Windows OS and can be found at <https://www.winpcap.org/>. WinPcap enables packet capture right from a network adapter and presents it to Wireshark before any processing is done by the OS.

WinPcap has worked well in a Windows environment, specifically the Windows NT family, for many years. When installing Wireshark, you will see the option to use WinPcap. However, it has not been recently updated and may not perform well on certain versions of Windows 10.

To overcome performance issues, users are directed to use Npcap, which might perform better, as outlined in the next section.

## Grasping Npcap

Prior to using Wireshark on a Windows OS to capture packets, you must install a capture engine. As a result, during the installation process, users will see an option to install Npcap. Npcap comes from the Nmap project and is the packet sniffing library for Windows. Npcap is based on WinPcap/libpcap but has improved features and enhanced ability to capture.

Npcap provides support for **Network Driver Interface Specification (NDIS) 6.0**, which is a major version enhancement. Having this support overcomes the limitations of WinPcap and will most likely improve capture on Windows 8.1 and later machines.

Npcap is compatible with WinPcap and can run alongside it, or you can uninstall WinPcap and use the Npcap driver exclusively. However, Wireshark documentation suggests using Npcap if you are using Windows 10 or later versions.

Npcap also includes several features that help improve functionality, as we'll learn next.

## Recognizing Npcap features

Some of the other Npcap features include loopback packet capture, which can be helpful during troubleshooting, along with support for the libpcap API. Npcap can also ensure enhanced security in that it can be set to restrict access to admin only on a Windows machine. If this option is set, then the user will have to authorize using the driver in the Windows **User Account Control (UAC)** dialog box.

A standard Wi-Fi card on a Windows machine can *only* be put into promiscuous mode, not monitor mode. As a result, when capturing traffic, you won't see raw 802.11 traffic or radiotap headers. Wireshark will wrap the traffic so that they look like an Ethernet packet; as such, they are sometimes called **fake Ethernet packets**.

However, with Npcap, users can capture raw 802.11 packets when using a supported wireless adapter. This is easily achieved by selecting the following option during the installation of Npcap:

### **Support raw 802.11 traffic (and monitor mode) for wireless adapters**

Once selected, Npcap will then have two modes:

- **Managed mode:** Captures Ethernet packets only
- **Monitor mode:** Uses `wlanhelper.exe`, which will allow you to switch into monitor mode and gather all 802.11 traffic, including data, control, and management packets that have radiotap headers

Because wireless networks are so prevalent, radiotap headers give us the ability to visualize information while troubleshooting wireless traffic. Let's explore this concept.

### Troubleshooting radiotap headers

Radiotap headers can be used when troubleshooting Wi-Fi, as they can provide a lot of information associated with each 802.11 frame. For example, you can assess signal strength and antennae noise, which are both shown in negative decibels per milliwatt (-dBm). Other indicators found in radiotap headers include the following:

- Channel frequency
- Data rate

To see an example of what you might see in a radiotap header, go to: [https://www.cloudshark.org/captures/ca7828d13464?filter=frame%20and%20radiotap%20and%20wlan%20and%20wlan\\_aggregate](https://www.cloudshark.org/captures/ca7828d13464?filter=frame%20and%20radiotap%20and%20wlan%20and%20wlan_aggregate).

Once you're on CloudShark, select **Export | Download File** from the menu found on the right-hand side of the screen, as shown in the following screenshot:

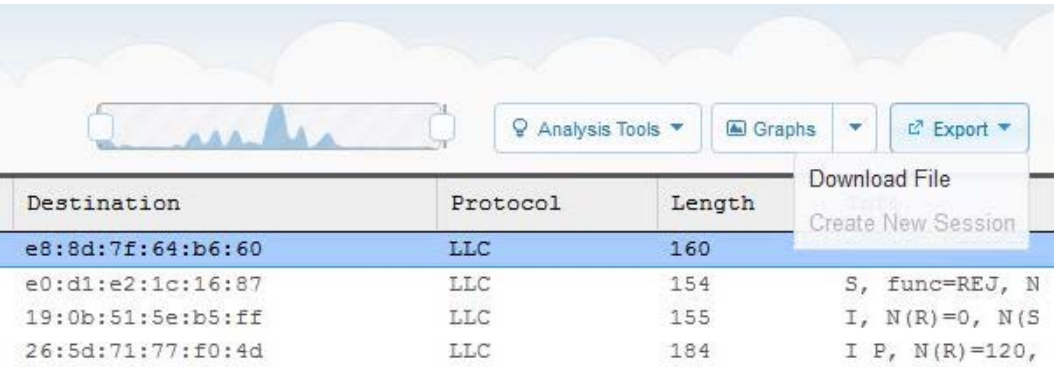


Figure 3.2 – Downloading the file from CloudShark

When the **Download** window opens, choose **Download the original file** and open it in Wireshark. Select **Frame 1** and expand the radiotap header to see the details, as shown in the following screenshot:

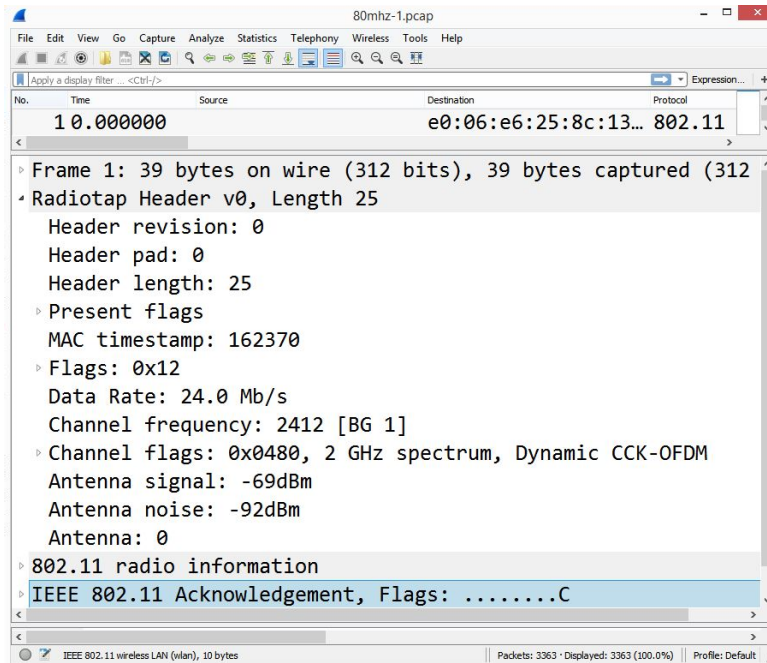


Figure 3.3 – Radiotap header

When troubleshooting a wireless connection, the signal strength will attenuate (or weaken) as the device moves farther away from the access point. If the signal is too weak, the device will be unable to effectively receive traffic.

Some suggested values for signal strength would be as follows:

- -30 dBm is an optimal level and the data rate will most likely be strong.
- -67 dBm is an appropriate level and the data rate is acceptable for most applications.
- -70 dBm is a less optimal signal, and the data rate will most likely suffer.
- Greater than -80 dBm is unacceptable.

As shown in *Figure 3.3*, the antennae signal is -69 dBm, which is at the edge of a less optimal signal and can result in a diminished data rate.

#### Important Note

Keep in mind when evaluating values such as signal strength and noise that these will represent an estimation and not an exact value.

Now that we have learned about the different capture engines, let's explore the various options to choose from while installing Wireshark on a Windows OS.

## Performing a standard Windows installation

The Windows installation is a straightforward process that presents the user with a series of prompts, which offer default values that the user may choose to accept or decline. Prior to installing, make sure you meet any system requirements. In most cases, **User Access Control** (UAC) will dim the screen and ask for confirmation to run the program.

With each new version, the components, options, and order of installation may change. The following is a list of dialog boxes you should expect to see when doing a routine setup. We'll start with the first two you will typically see, the welcome and the license agreement.

## Beginning the installation

As you begin the installation, Wireshark displays a series of prompts. The following are generally the first two screens you will see:

- i. **Welcome screen:** The Wireshark installation begins with a warning to make sure Wireshark is not running before launching the wizard, which will guide you through the installation.
- ii. **License agreement:** The next screen outlines the terms of the license, which must be read and agreed upon before moving on to the next step.

### Note

It might be worthwhile to read the license, as it provides a detailed overview of the license agreement, specifically that Wireshark is distributed under the **GNU General Public License** and not Unix's.

The first two prompts are fairly straightforward. This next section provides detailed information on what components to select during installation.

## Choosing components

During the installation, you may be given the choice to accept or reject certain components. **Choose Components** has a number of choices. The user may accept all choices or select specific components to install, as shown in the following screenshot:

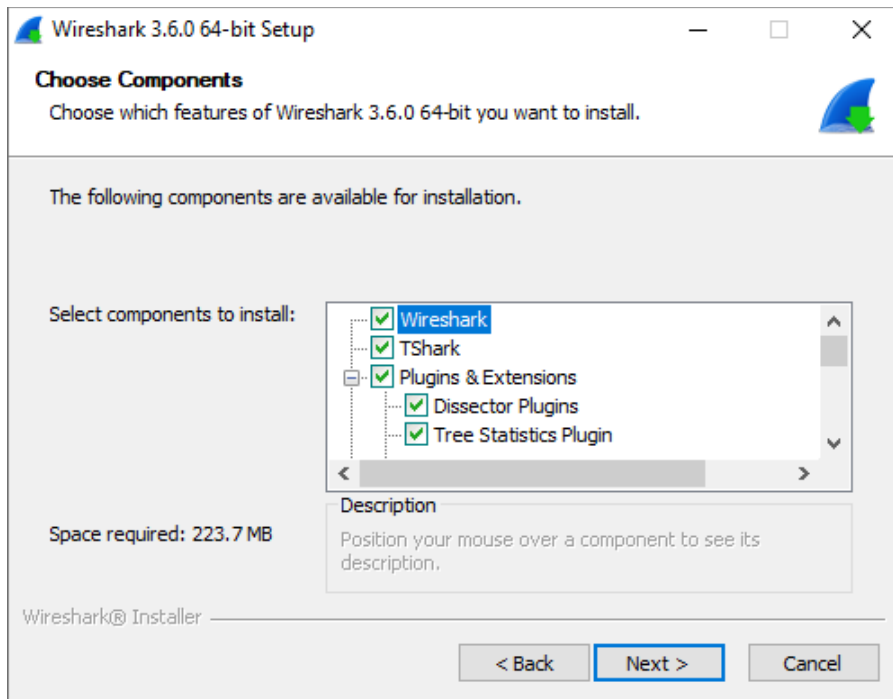


Figure 3.4 – The Choose Components screen

The first choice when selecting a component to install is whether to install **Wireshark**. While this may be obvious, the user may *only* want to install `tshark`, a lightweight CLI tool that is not as resource-intensive as the full Wireshark GUI.

The next section includes extra features and protocol dissectors for Wireshark and `tshark`. Let's take a look.

## Selecting plugins and extensions

The section includes plugins and extensions that can improve functionality. Within this section, you'll find many options, outlined as follows:

- **Dissector plugins:** Plugins that include extended dissections
- **Tree statistics plugins:** Provides extended statistics
- **Meta Analysis and Tracing Engine (MATE):** Provides the ability to filter frames that are related to one another
- **Transum:** Computes response time with a number of different protocols
- **File type plugins:** Capture file support

- **Codec plugins:** Provides additional support for codecs
- **Configuration profiles:** Create a customized profile that is specific to a user or protocol
- **Simple Network Monitor Protocol (SNMP) MIBs:** Provide a more extensive dissection of SNMP.

In addition to plugins, Wireshark includes a set of tools, which we'll learn about next.

## Choosing tools

Wireshark has a list of over a dozen command-line tools to select. The first part of the list includes tools that enhance functionality when working with packets, as follows:

- **Editcap:** Allows you to adjust timestamps, delete packets, and convert file formats.
- **Text2pcap:** Provides the ability to take an ASCII hexdump and convert the file to a libpcap-format capture file.
- **Mergecap:** Used when you need to combine capture files, as it merges two or more files into one, either by appending or by merging by timestamp.
- **Reordercap:** Rearranges packets from an input file by sorting the timestamps and converting them to an output file.
- **DFTest:** Used when you have to debug a **display filter (dfilter)**, DFTest will show the display filter byte code.
- **Capinfos:** Provides information such as the number of packets, duration, and other information about a capture file.
- **CapType:** Will recognize one of the many supported file formats and then print the type of file
- **Rawshark:** Will output and analyze raw pcap data when required for external (third-party) integration or exports.
- **MMDBResolve:** Will identify and print a packet's geolocation by an IPv4 and IPv6 address using a GeoLite2 database

The second half of the list has tools that work with external data, which include the following:

- **Androiddump:** Used when it's necessary to capture from an Android device. You'll need to have the Android **Software Development Kit (SDK)** along with permission to access the device.
- **Sshdump and Ciscodump:** These provide an interface to remotely capture traffic from a Cisco router via a **Secure Shell (SSH)** connection.
- **UDPDump:** Offers a capture interface that pulls **User Datagram Protocol (UDP)** packets from network devices when debugging applications that run over UDP.
- **RandpktDump:** Enables access to the **random packet generator (randpkt)** during testing or for educational purposes.
- **Etwdump:** Provides access to an **Event Trace Log (ETL)** file, which is a log file generated by Microsoft Tracelog.

In addition to the plugins, extensions, and tools, you will have the option to obtain the user guide.

## Obtaining the user guide

While there are plenty of resources available, selecting this option will include a copy of the user guide that you can access offline.

As you can see, there are many components that you can select during the Wireshark installation. Keep in mind that these options periodically change.

The next two prompts offer choices on shortcuts, outlining file extensions, and deciding where to house the install folder.



## Creating shortcuts and selecting an install location

Within the installation, you'll have choices on whether you would like some shortcuts for **Wireshark Start Menu Item**, **Wireshark Desktop Icon**, or **Wireshark Quick Launch Icon**, as shown here:

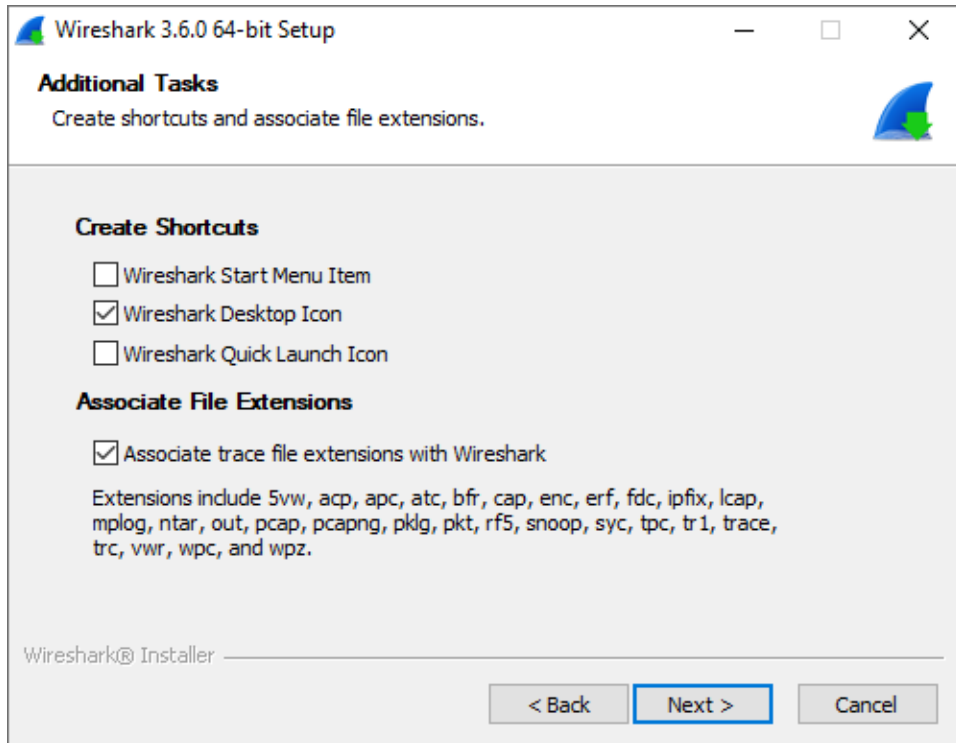


Figure 3.5 – The Additional Tasks screen

In addition, you will be able to include all of the available file extensions.

The next prompt will give you a **Choose install location** option. At this point, you can select the default location or browse to a user-defined folder. Wireshark will provide information on how much space is required.

In addition to those listed earlier, you will have a few more decisions to make. Those include selecting a capture engine and whether or not to use USB capture before completing the installation.

## Capturing packets and completing the installation

Wireshark needs a capture engine to gather network traffic and will query the system to see whether one is present.

The following prompts deal with capturing traffic, along with what you should expect to see when Wireshark completes the installation.

### Checking for a capture engine

At this point, Wireshark will check whether Npcap or WinPcap is installed. You are then presented with a **Packet Capture** screen that states **Wireshark requires either Npcap or WinPcap to capture live network data**, as shown in the following screenshot:

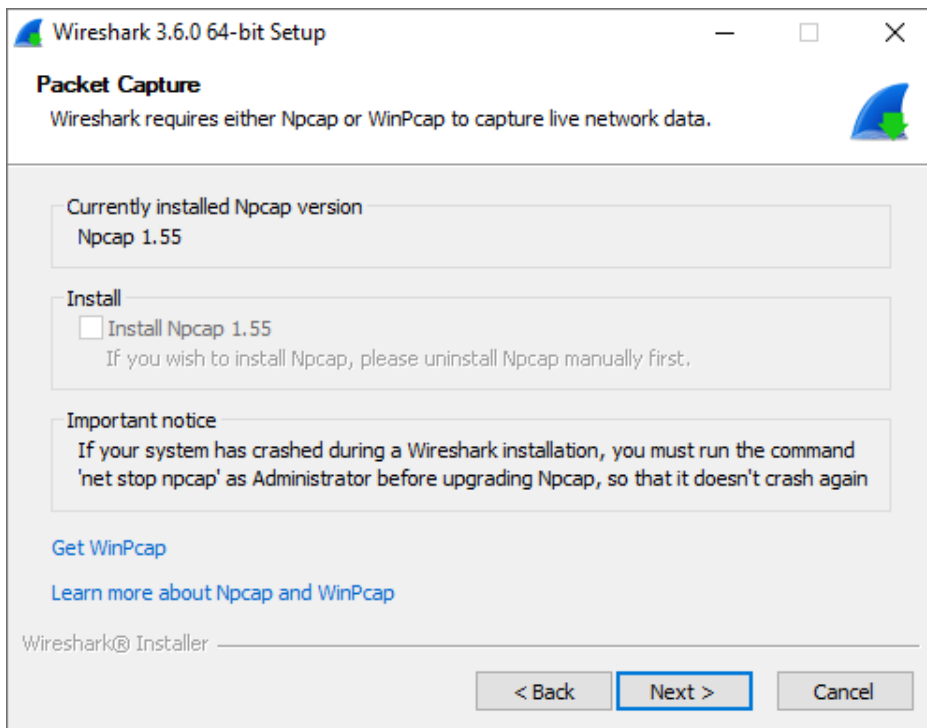


Figure 3.6 – The Packet Capture screen

If you have Windows 8.1 or higher, then Npcap is most likely an appropriate choice. Wireshark presents links for the user to do the following:

- Get Npcap if needed
- Learn more about Npcap and WinPcap

Wireshark also offers **USB Capture**, which is optional.

## Using a USB interface

At times, it is necessary to capture USB traffic. This option checks to make sure you have USBPcap currently installed and, if not, gives you an option to install it, as shown in the following screenshot:

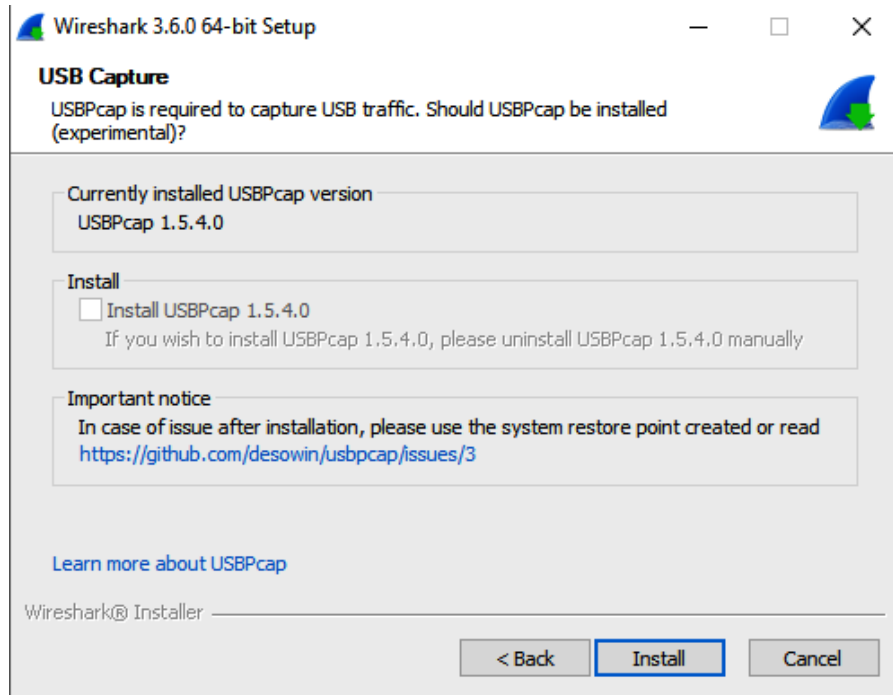


Figure 3.7 — The USB Capture screen

You may find the need to use **USB Capture**, for example, for troubleshooting or monitoring transactions. If you choose not to install **USB Capture**, then you can install it at a later date.

Once you have made all of your selections, Wireshark will present a notification that the process has been completed.

## Completing the Wireshark setup

Once complete, the screen will show the output of the files extracted during the installation. At this time, you can choose to run Wireshark. In addition, you can also select **Show News**, which will bring up the latest Wireshark news and information.

Because of the variety of options available, it may seem overwhelming. There is help. The next section provides an overview of many of the resources found on the Wireshark home page.

## Reviewing available resources

When you first visit <https://www.wireshark.org/>, you are presented with a splash page that offers download options. In addition, across the top, there are several hyperlinks to resources such as news, where to find help, and where to go to meet other Wireshark users.

Let's see where you can find the latest news.

## Viewing news and help topics

The **News** section is where you will find the latest on Wireshark improvements, vulnerabilities, and bug fixes. Once there, you can drill down to specific versions of release notes and find more information.

On the lower part of the page, you will find links to archived news events for past Wireshark releases, as shown in the following screenshot:

### Wireshark 3.6.0 Released

**November 22, 2021**

Wireshark 3.6.0 has been released. Installers for Windows, macOS 10.13 and later, and source code are now available.

#### What's New

Many improvements have been made. See the "New and Updated Features" section below for more details. You might want to pay particular attention to the display filter syntax updates.

#### New and Updated Features

The following features are new (or have been significantly updated) since version 3.6.0rc3:

- The macOS Intel packages now ship with Qt 5.15.3 and require macOS 10.13 or later.

The following features are new (or have been significantly updated) since version 3.6.0rc2:

- Display filter set elements must now be comma-separated. See below for more details.

The following features are new (or have been significantly updated) since version 3.6.0rc1:

- The display filter expression "a != b" now has the same meaning as "!(a == b)".

The following features are new (or have been significantly updated) since version 3.5.0:

- Nothing of note.

Figure 3.8 – The News section at Wireshark.org

The **Get Acquainted** menu choice provides a link to the **About** page, where you will find general information on Wireshark, including features, authors, awards, and accolades. Additionally, you'll find another link to download Wireshark, but you will also find a blog. It's worth visiting the blog because there are personal insights from developers, including Gerald Combs, the original developer.

The **Get Help** menu lists many opportunities to ask questions, using the following links:

- **Ask a Question**
- **FAQs**
- **Documentation**
- **Mailing Lists**
- **Online tools**
- **Wiki**
- **Issue Tracker**

The resources will help you keep up to date with Wireshark. The Wireshark community is very helpful in trying to assist users and novices with issues. Guests are encouraged to visit the **Ask a Question** forum, where you can view some of the questions and registered users can post a response. As shown in the following screenshot, there are several topics to investigate:

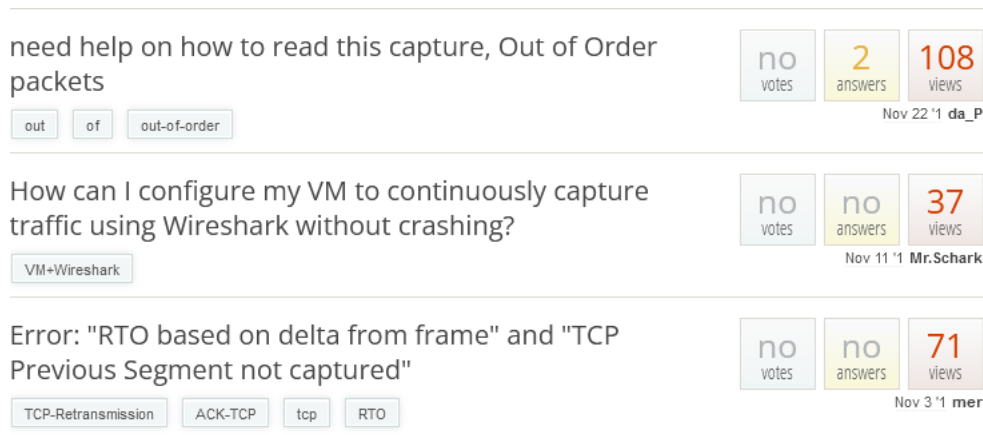


Figure 3.9 — Questions in a Wireshark forum

While most of us are Wireshark *users*, there are hundreds of developers that have worked hard to improve Wireshark over the years. The **Develop** menu choice lists various links – **Get Involved**, **Developers Guide**, **Browse the Code**, and **Latest Builds**.

The **SharkFest** menu choice will point to the Wireshark conference, SharkFest. The conference is where you can undergo training, gain practical experience, and network among the Wireshark community and the developers that make Wireshark possible.

Everyone needs a sponsor. The **Our Sponsor** menu choice takes you to the sponsors that make SharkFest possible.

Because most of the time you visit the Wireshark home page to download Wireshark, the following sections explore options you may find when downloading Wireshark.

## Evaluating download options

Once on the **Download** page at Wireshark.org, you will see options for the type of file you want, along with what release. You can choose from stable, old stable, and documentation. The following screenshot shows the choices for downloads under **Stable Release**:

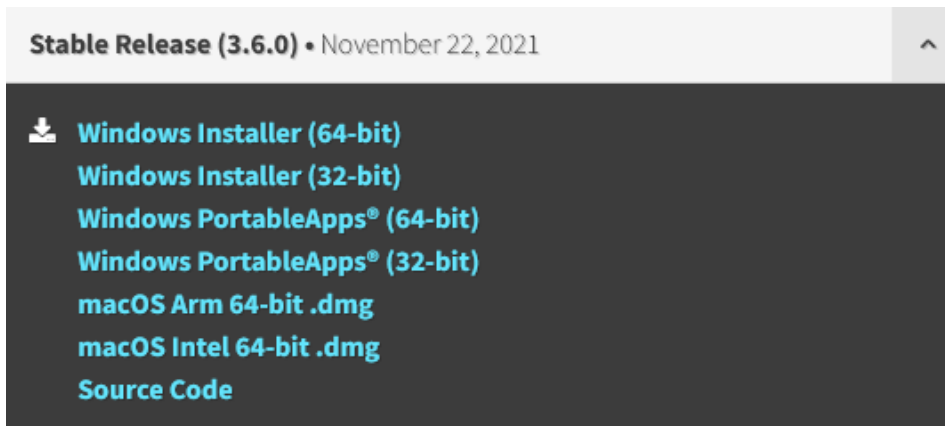


Figure 3. 9 — Choices for downloads under Stable Release

In most cases, you will select an option under **Stable Release**, which is the most recent version of Wireshark. The stable release will have most of the bugs resolved and will function at an optimal level. The following lists the available choices:

- **Windows Installer:** This provides a standard download for either a 64-bit or 32-bit Windows OS.
- **Windows PortableApps (64-bit) or (32-bit):** This is an option that you can run from a flash drive for troubleshooting without having to install on a system.

- **macOS Arm 64-bit .dmg or Intel 64-bit .dmg:** This is an option for macOS users to install, download, and unpack the **Disk Image (DMG)** and then run the install. In some cases, you may have to complete additional configuration options in Wireshark to resolve any errors.
- **Source Code:** This provides an archive of the source code, where you can study the various files. The following is a screenshot of the result when selecting the **Source Code** option on the **Download** page:

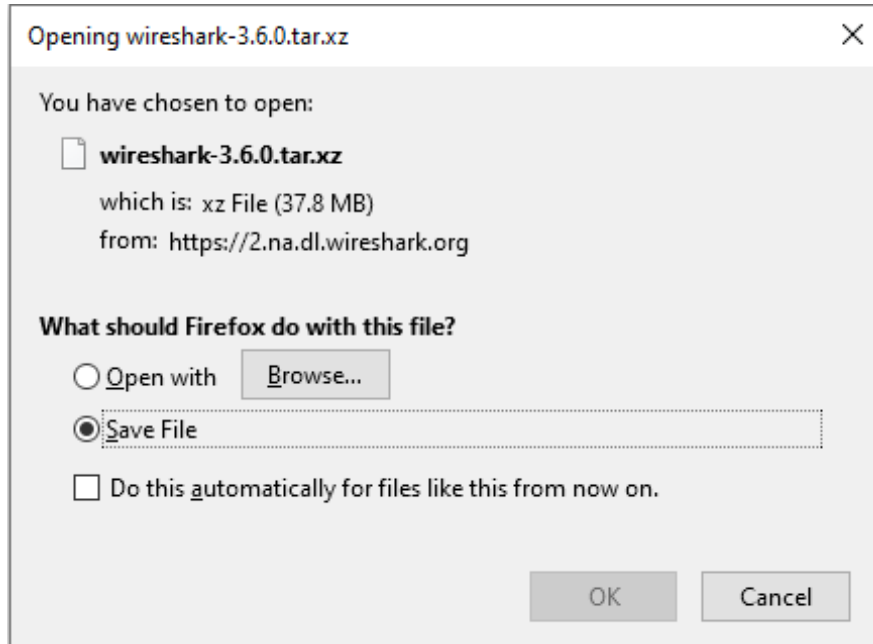


Figure 3.11 – The Source Code option on the download page

#### Important Note

If you are serious about development, then you should obtain and update your code from Wireshark's Git repository. Git will automatically merge changes into your personal source so that you can keep your source updated.

It's worth taking the time to explore all the resources on <https://www.wireshark.org/> to help you learn more about packet analysis and find the latest information on Wireshark.

## Summary

In this chapter, we covered how you can easily download and install Wireshark on a variety of different OSes. By now, you have a better understanding of the different capture engines and how they provide a way to gather the traffic from the network. In addition, you can now appreciate Npcap, the newest capture engine. We learned how Npcap provides enhanced features, such as the ability to capture raw 802.11 packets when using a supported wireless adapter.

When you're ready to install Wireshark on a Windows machine, you'll be more confident as you step through all the prompts, from launching the installer to completing the installation. As with most software installations, the user is given some choices. So that you are more aware of the many choices when downloading Wireshark, we reviewed the various options for the file you want, along with what type of release. And finally, you now understand that if you do run into trouble, there is help available, as evidenced by the many resources at [Wireshark.org](http://Wireshark.org).

Now that you understand what's involved during the installation process, you're ready for the next chapter, where we will explore the Wireshark interface. We'll take a look at all the elements to help you navigate better as you begin capturing and analyzing packets. We will then examine the Wireshark welcome screen and go through the various icons and shortcuts. Finally, we'll explore the three most commonly used access menu choices, **File**, **Edit**, and **View**, all of which will help improve your workflow.

## Questions

Now, it's time to check your knowledge. Fill in the blanks from the multiple choices and then check your answers, which can be found in the Assessments appendix:

1. \_\_\_\_\_ is a capture engine originally developed for Unix-like OSes and is baked into Snort, TCPDUMP, and other packet analyzers to grab packets as they come off the network interface:
  - A. Capinfos
  - B. MATE
  - C. libpcap
  - D. Transum



2. Radiotap headers can be used when troubleshooting Wi-Fi, as they can provide a lot of information associated with each \_\_\_\_\_ frame:
  - A. 802.3
  - B. 802.15
  - C. 802.11
  - D. 802.8
3. \_\_\_\_\_ is a program that will identify and print a packet's geolocation by using IPv4 and IPv6 addresses:
  - A. dftest
  - B. tshark
  - C. mergecap
  - D. mmdbresolve
4. \_\_\_\_\_ is the newest capture engine option for Wireshark, with many benefits and features to enhance your packet capture:
  - A. AirPcap
  - B. Npcap
  - C. WinPcap
  - D. libpcap
5. One of Wireshark's tools, \_\_\_\_\_ allows you to adjust timestamps, delete packets, and convert file formats:
  - A. Editcap
  - B. Capinfos
  - C. dftest
  - D. Reordercap
6. One of the tools available during download is \_\_\_\_\_, which computes response time with a number of different protocols:
  - A. Rawshark
  - B. Transum
  - C. Randpkt dump
  - D. MATE

7. Once at Wireshark.org, the \_\_\_\_\_ section is where you will find the latest on Wireshark improvements, vulnerabilities, and bug fixes:
- A. **Get acquainted**
  - B. **Get help**
  - C. **News**
  - D. **Windows Installer**

## Further reading

Please refer to the following links for more information:

- For more information on signal strength, visit <https://www.securedgenetworks.com/blog/wifi-signal-strength>.
- For more information on radiotap headers, visit <https://www.intuitibits.com/2015/04/06/link-layer-header-types/>.
- To read the user guide for Transum, visit <https://gitlab.com/wireshark/wireshark/-/wikis/TRANSUM-User-Guide>.
- To obtain the latest GeoLite2 database, go to <https://dev.maxmind.com/geoip/geoip2/geolite2/>.
- For more information on using Wireshark on a macOS, visit [https://www.wireshark.org/docs/wsug\\_html\\_chunked/ChBuildInstallOSXInstall.html](https://www.wireshark.org/docs/wsug_html_chunked/ChBuildInstallOSXInstall.html).
- You can compare the features of WinPcap or Npcap by going to <https://nmap.org/npcap/vs-winpcap.html>.



# 4

## Exploring the Wireshark Interface

When you launch Wireshark for the first time, it's sometimes puzzling having to navigate the interface until you are familiar with all of the elements. Once you have a grasp of all the components, toolbars, and menu choices, you can capture and analyze traffic more efficiently. In this chapter, we'll start by exploring the Wireshark interface and reviewing all of the essentials of the welcome screen, including the sparklines, capture filters, and interfaces.

Although Wireshark currently has over 10 menu choices, in most cases, you'll find that there are a few that are more commonly accessed. So that you are more confident when moving about the interface, we'll examine the **File** menu, where you can open a file, save, print, and export a capture. We'll also investigate the **Edit** menu, where you can mark packets, set time references, and add comments. Finally, we'll take a look at the **View** menu so that you can learn how to customize the look and feel of the Wireshark interface.

In this chapter, we will cover the following topics:

- Opening the Wireshark welcome screen
- Exploring the **File** menu
- Discovering the **Edit** menu options
- Exploring the **View** menu

## Opening the Wireshark welcome screen

Once you launch Wireshark, you'll find the menu choices across the top of the Wireshark welcome screen. If you don't have a capture file loaded, you will see that all of the menu choices are available. However, the icons might be dimmed, as shown in the following screenshot:

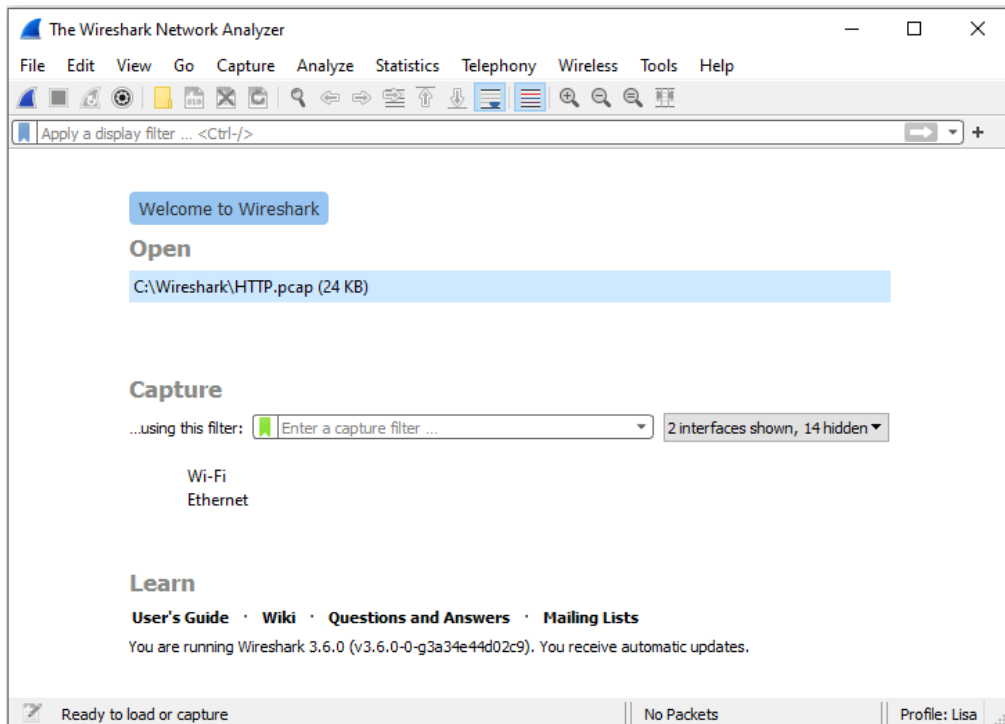


Figure 4.1 – The Wireshark welcome screen

The icons will become active once you have a file open or are actively capturing packets.

Once in the welcome screen, you'll most likely either launch a file or begin capturing traffic. So, let's start with the many options that are available when opening a packet capture.

## Selecting a file

Beneath the icons and the display filter, you'll see a banner that reads **Welcome to Wireshark**. Underneath the banner, you will see the **Open** label, which will identify any previously opened packet captures that are available.

If you right-click on a file, you have the following options: **Show in Folder**, **Copy file path**, or **Remove from list**.

**Note**

In some cases, you might not have any files listed. In that case, the **Show in Folder**, **Copy file path**, or **Remove from list** options won't be available.

For example, while in a Windows **operating system (OS)**, if you select **Show in Folder**, as shown in the following screenshot, this will launch the file explorer:

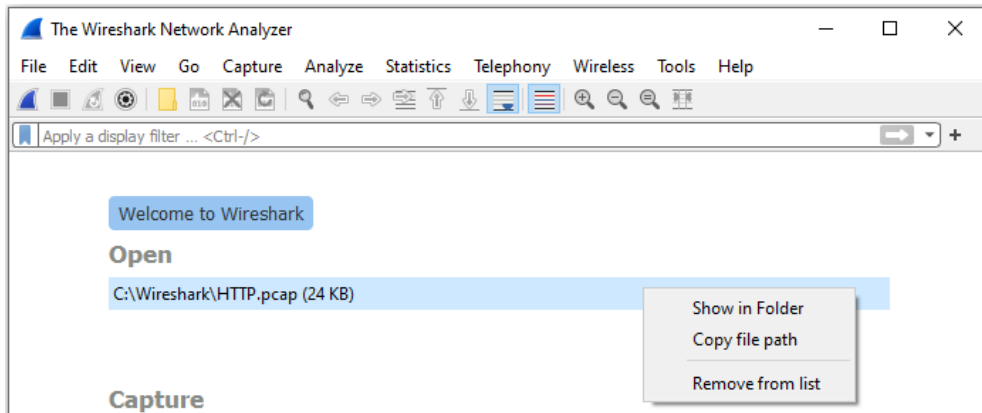


Figure 4.2 – Right-click and Show in Folder

At that point, you can then select a file, drag it onto the Wireshark screen, and the file will open.

Once you begin capturing packets, you might have a dozen or so files in the **Open** file area. Although the files are shortcuts for ease of access, they could be distracting. If you want to remove the files, navigate to the **File | Open Recent | Clear Menu** menu, as shown in the following screenshot:

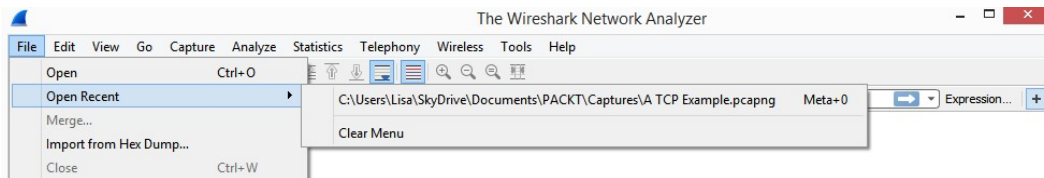


Figure 4.3 – The Clear Menu choice

Next, let's take a look at the options for gathering network traffic.

## Capturing traffic

When you're ready to capture traffic, you'll want to properly set up Wireshark. You can find the **Capture** label in the middle of the screen. Below that, you'll see **...using this filter:**. Once there, you can apply a capture filter in the space provided.

**Note**

A capture filter allows you to filter specific traffic during a capture. If you do use a capture filter, be aware that it will limit what you capture to only what you have filtered on, and you could miss the traffic that can help with your analysis.

On the right-hand side of the capture filter, you will see a drop-down menu that reads **All interfaces shown**. If you want to remove any of the classes of interfaces (such as **Wired**, **Bluetooth**, or **Virtual**), you can select one from the drop-down menu, as shown in the following screenshot:

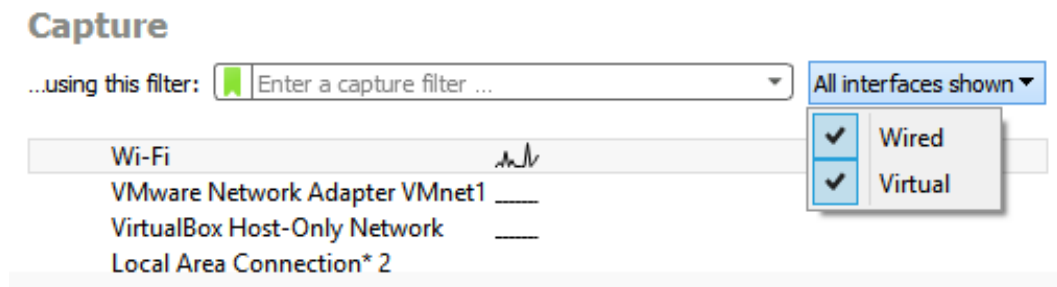


Figure 4.4 – The display interface(s)

Beneath the capture filter area, you'll see a list of available interfaces, and you can quickly begin capturing traffic by selecting an active sparkline.

Once you have either opened a packet capture or run a capture for analysis, you will most likely use one of the many menu choices. The following section covers what is possible in the **File** menu.

## Exploring the File menu

When working with the Wireshark interface, **File** is the go-to menu because it has all of the tasks commonly associated with working with a file, as shown in the following screenshot:

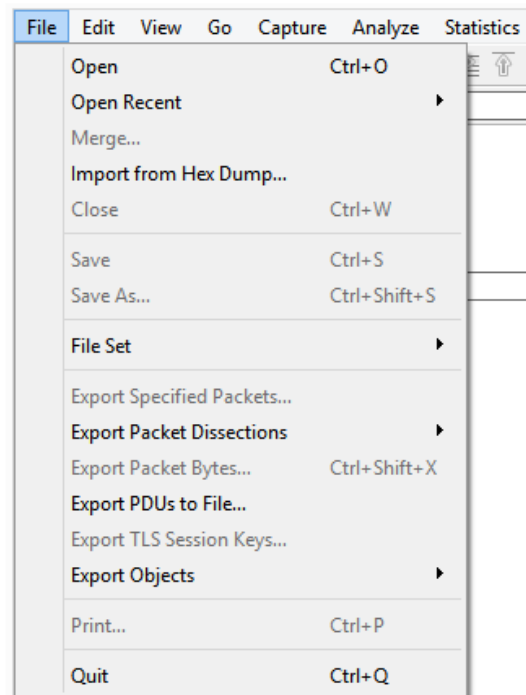


Figure 4.5 – The File menu

In this section, we'll walk through the many options found in the **File** menu. Let's begin with the ways to locate and open a file, save a capture, and compare options that are available when you close a file.

## Opening a file, closing, and saving

The first section in the **File** menu offers many choices for locating and opening files so that you can begin your analysis. While looking at the menu choices, you will see a light gray line that separates grouped objects. The first grouping is related to opening and closing a file, and it includes the following choices:

- **Open** will launch a dialog box that will allow you to select any supported file.
- **Open Recent** will list any recently accessed files.
- **Merge...** will allow you to merge a file with the capture you have open. When merging, it's important that the time values are synchronized, as that is what Wireshark uses to merge the two files.



- **Import from Hex Dump...** is convenient when someone has sent you a hex dump from another device for analysis. The import dialog box will step you through the process of selecting the appropriate choices when importing the file.
- **Close** will close the current capture. If it is a new capture or you have added comments, Wireshark will ask you whether you'd like to save the file.

The next grouping consists of tasks that are related to *saving* a file:

- **Save** allows you to save the current file. This is useful if you have added comments or modified the file and want to preserve the changes.
- **Save As...** allows you to save the file as something other than the default extension, `.pcapng`. Once in the dialog box, you can select from the many different file formats that Wireshark has available.

The **File Set** option offers the ability to work with a set of files. For example, if you're doing a firewall ruleset and you're going through a whole month of files, you can work through the list one by one. When this option is selected, you can right-click and your choices will be **List Files**, **Next File**, and **Previous File**.

The following **File** menu section examines the many ways to export parts of a capture.

## Exporting packets, bytes, and objects

Instead of saving an entire file, you might want to only save a portion of the file or even just the objects found within the file. Within this section, you'll find several export options.

The first option is **Export Specified Packets**. Once the dialog box is open, you can include only displayed packets, a range of packets, and marked packets, as follows:

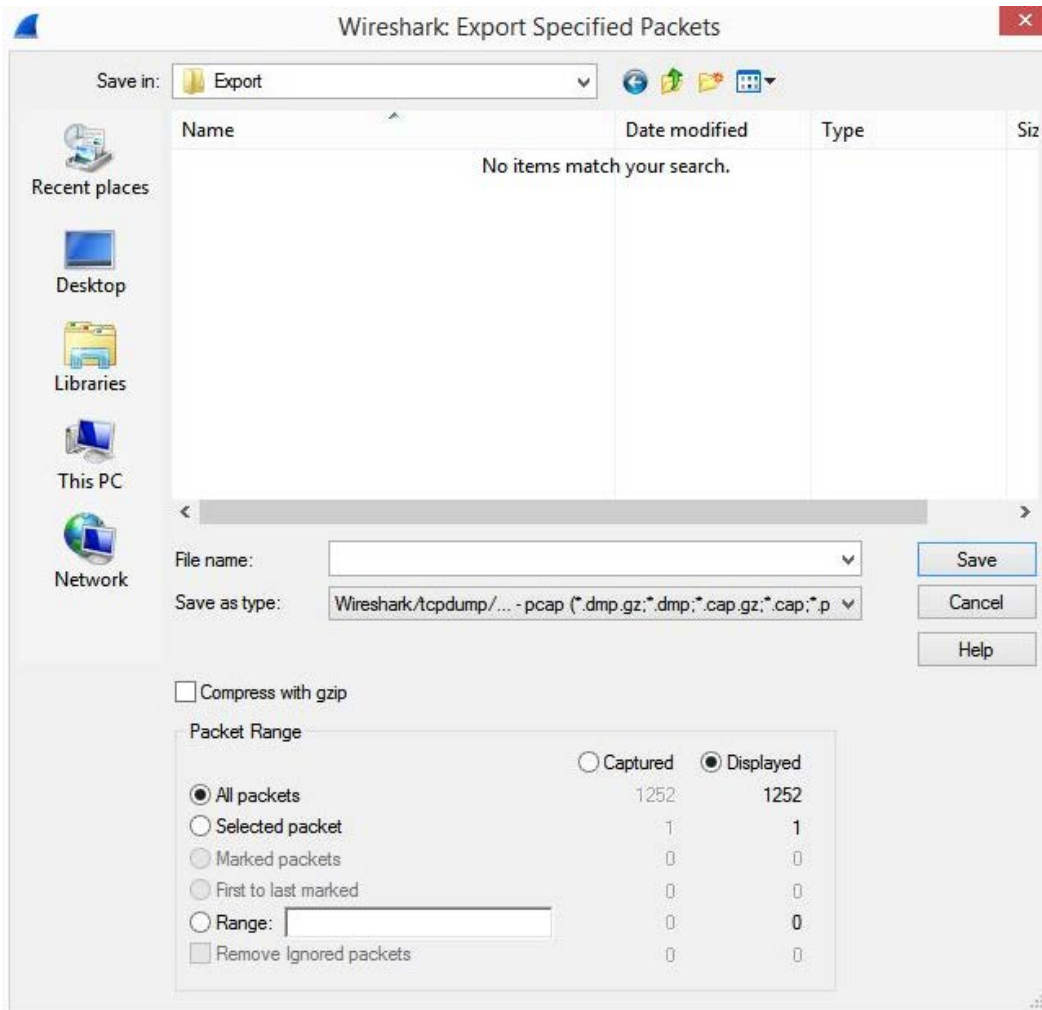


Figure 4.6 – The Export Specified Packets dialog box

Another submenu choice is **Export Packet Dissections**, which offers many choices to export, as shown in the following screenshot, including **Comma-Separated Values (CSV)**, plain text, and **JavaScript Object Notation (JSON)**:

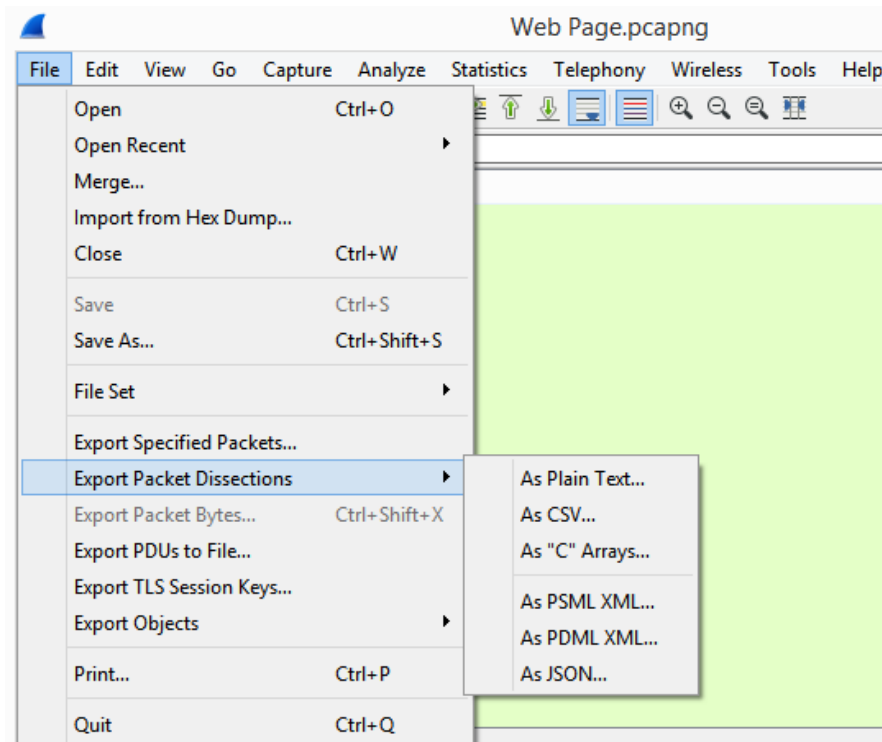


Figure 4.7 – The Export Packet Dissections menu

The next three submenu items offer more ways to export components and include the following:

- **Export Packet Bytes...:** This feature exports the packet bytes into C arrays so that you can import the stream into a C program.
- **Export PDUs to File...:** This menu choice offers many selections to export. However, this feature may not show a usable output and may only work with specific applications.
- **Export TLS Session Keys...:** If there are session keys within the file, select this option to export the keys that can be used to decrypt the data. Wireshark will display a popup if there are no **Transport Layer Security (TLS)** keys to save.

- **Export Objects:** This menu choice allows you to export various objects found within the file, such as **Hypertext Transport Protocol (HTTP)** objects.

Of all the options, **Export Objects** can provide a way in which to visualize the various objects found within the data stream, such as files, images, and executables. Let's explore this option next.

## Exporting objects

The **Export Objects** submenu choice identifies any objects found within the file and allows you to save and examine the objects. Once in the submenu choice, right-click to see the selection, as shown in the following screenshot:

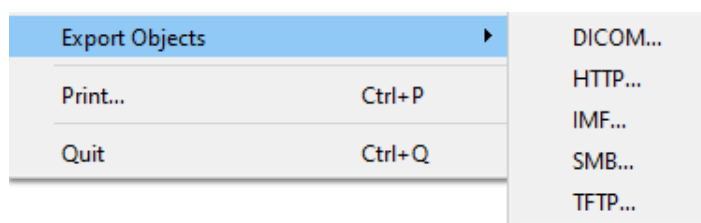


Figure 4.8 – The Export Objects selections

To see an example of what you can export, navigate to <https://www.cloudshark.org/captures/0012f52602a3>.

Once you're on CloudShark, select **Export | Download File** from the menu found on the right-hand side of the screen. Then, open the packet capture file, <http://packetlife.net/captures/HTTP.cap>, in Wireshark. Once it has been downloaded, save it as `HTTP.pcap`.

Once open, select **Export Objects | HTTP**, which will display the list of objects found, as follows:

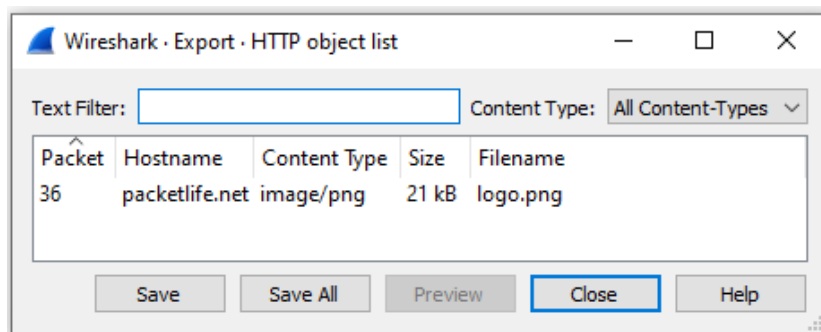


Figure 4.9 – The Export Objects | HTTP dialog box

Within this window, you can select **Save**, **Save All**, or even **Preview**. If there are multiple objects in the capture, you can use the **Text Filter** function to drill down to a particular object.

I selected **Save** and then navigated to a temporary folder, `Export`. For the filename, I selected `logo.png`, as follows:

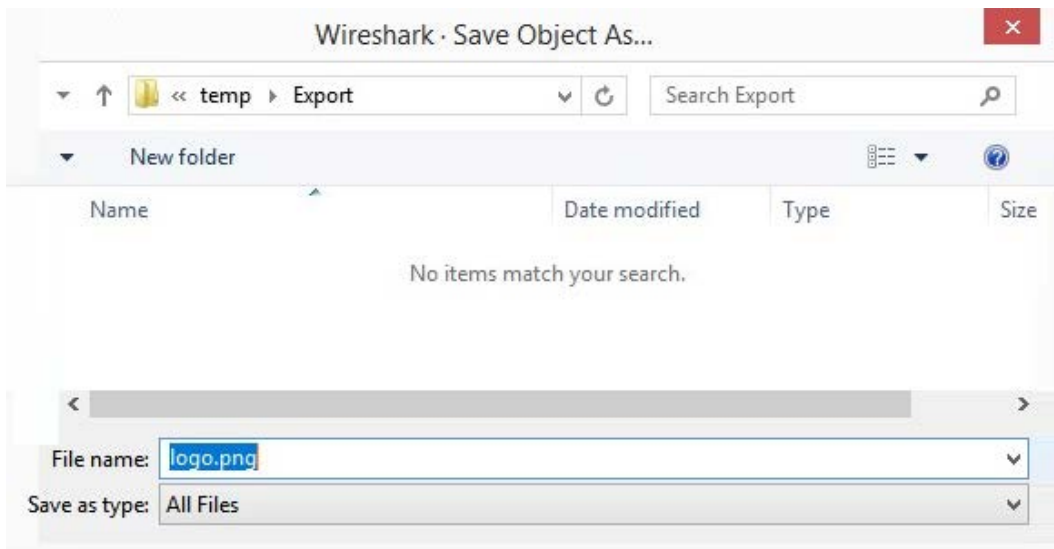


Figure 4.10 – The Export Objects | Save As dialog box

When you are done exporting, navigate to the folder and open the image. You should see the **PacketLife.net** logo.

If there are other objects within the file, you can save them in a similar manner. Alternatively, you can select **Save All**, which will save all objects within the capture.

As you can see, there are many ways to export components in Wireshark. In the lower part of the **File** menu, there are options to **Print** and **Quit**, which we'll evaluate next.

## Printing packets and closing Wireshark

While examining packets, Wireshark offers many ways in which to print different sections of the capture. Using the `HTTP.cap` file, select **Print**, and you will see the following:

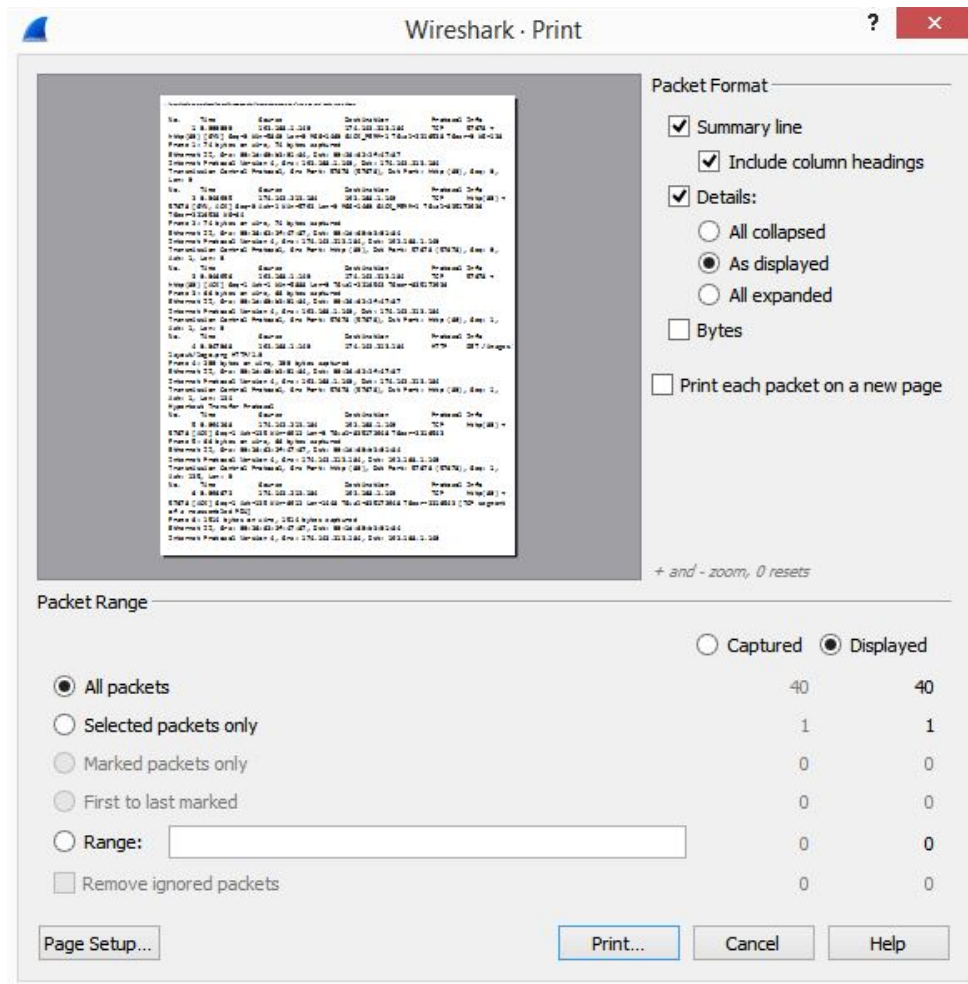


Figure 4.11 – The File | Print option

Once there, you can choose to print all of the packets, selected packets only, or a range of packets to a **Portable Document Format (PDF)**, which you can then include in a report.

After completing your analysis, you'll want to quit the application. If you select **Quit**, and you have a new capture, Wireshark will ask you whether you'd like to save the file.

After running a capture or opening a file, you'll want to begin your analysis. In the following section, we will cover the **Edit** menu, where you can discover the many possibilities that are available when working with a packet capture.

## Discovering the Edit menu

The **Edit** menu allows you to find and mark packets, set a time reference, copy, provide detailed information for creating a configuration profile, or modify your preferences. The following is a screenshot of the **Edit** menu:

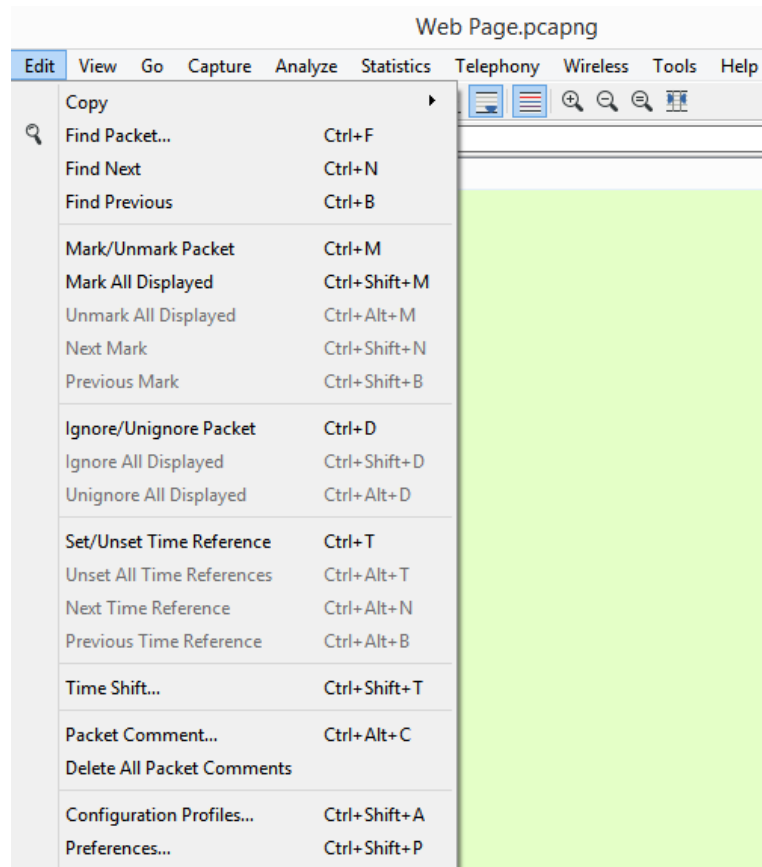


Figure 4.12 – The Edit menu

Within the **Edit** menu, there are numerous options. The following discussion outlines ways in which to copy various items and find packets within Wireshark.

## Copying items and finding packets

In this section, we'll learn how Wireshark makes it easy to copy several objects within the interface. In addition, we'll discover how we can locate a specific packet or a string value within the capture.

Let's begin by covering the many options within the **Copy** submenu.

## Outlining copy options

While analyzing packets, you might see an item or value you would like to copy. The **Edit | Copy** menu choice has many submenus to further define options, as follows:

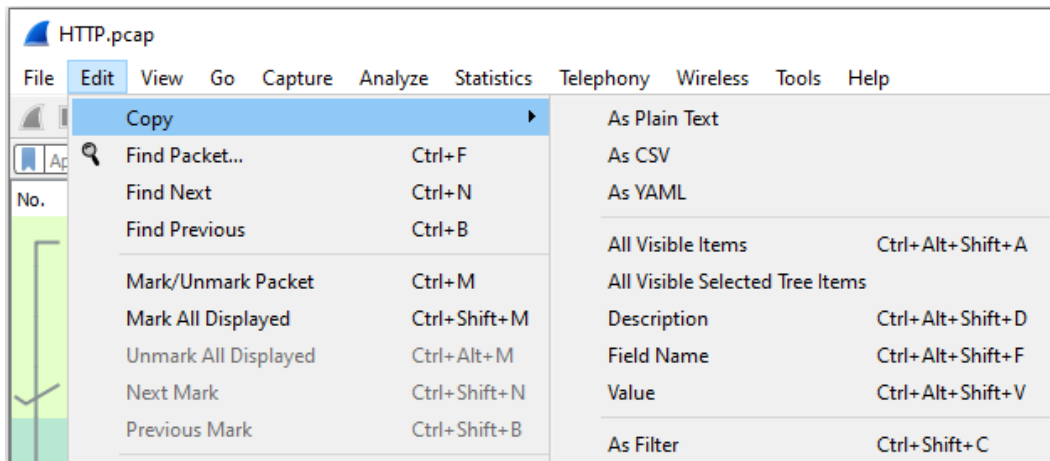


Figure 4.13 – The Copy options

Using the HTTP .cap file, we'll walk through some of the results when selecting the different copy options:

- **As Plain Text:** Selecting this option will copy the details of the frame in plain text. To view this option, go to frame five (5) and then select **As Plain Text**. It will appear as follows:

No.	Time	Source	Destination	Protocol	Info
5	0.094268	174.143.213.184	192.168.1.140	TCP80 →	57678 [ACK] Seq=1 Ack=135 Win=6912 Len=0 Tsval=835172948 TSecr=2216543

- **As CSV:** The CSV format is useful if you need to import the data in a spreadsheet. This format will present the information with each value separated by a comma. To view this option, go to frame five (5) and select **As CSV**. This will copy the details of the frame, as follows:

```
"No.", "Time", "Source", "Destination", "Protocol", "Info"
"5", "0.094268", "174.143.213.184", "192.168.1.140", "
TCP", "80 → 57678 [ACK] Seq=1 Ack=135 Win=6912 Len=0
TSval=835172948 TSecr=2216543"
```



- **As YAML:** The **YAML Ain't Markup Language (YAML)** presents the information as a YAML binary dump. In addition to the details of the packet, you will see the details of the file path.
- **All Visible Items:** This will copy all elements of the selected frame.
- **All Visible Selected Tree Items:** This will copy all elements of the selected tree items. For example, if we go to frame five (5) and select the TCP header, this option will copy any of the tree items shown:

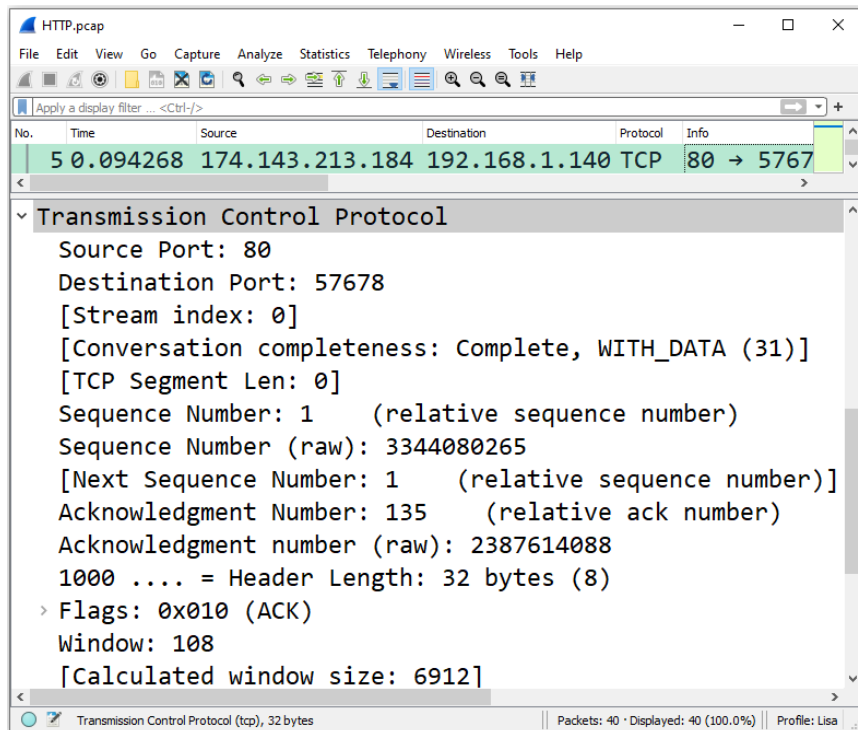


Figure 4.14 – The expanded TCP header

However, when using this option, it will not copy any tree items that are *not* expanded. For example, you will not see the details of the TCP `Flags`, as the tree has not been expanded.

- **Description:** This will copy the selected value. For example, go to frame five (5), expand the IP header, and select the source IP address. Select **Edit | Copy | Description**, which will copy the following: `Source Address : 174.143.213.184.`
- **Field Name:** This copies the selected field name. For example, go to frame five (5), expand the IP header, and copy the **Field Name** option of the source IP address. This will copy the following: `ip.src.`
- **Value:** This will copy the selected field value. Go to frame five (5), expand the IP header, and select the source IP address. This will copy the IPv4 address: `174.143.213.184.`
- **As Filter:** This will create a filter based on the IPv4 address you selected or any other value. Following this, you can paste the filter in the display filter area, press *Enter*, and Wireshark will run the filter.

After the **Copy** submenu choice, the next grouping offers ways to find packets.

## Locating packets

While conducting an analysis, you might need to find specific packets. The following is a list of choices that can help you to navigate a packet capture:

- **Find Packet...:** This is where you can search for specific packets and even find string values within a packet capture.
- **Find Next:** If Wireshark finds what you are looking for, **Find Next** will go to the next instance.
- **Find Previous:** If Wireshark finds what you are looking for, **Find Previous** will go back to the previous packet.

While working with packets, you might find and mark packets that are interesting so that you can return to them at a later date. In addition to this, you might want to ignore specific packets.

## Marking or ignoring packets

Marking packets while in Wireshark is easily achieved. Once you have selected a packet, right-click, and Wireshark will mark the packet with a black background and white text, as follows:

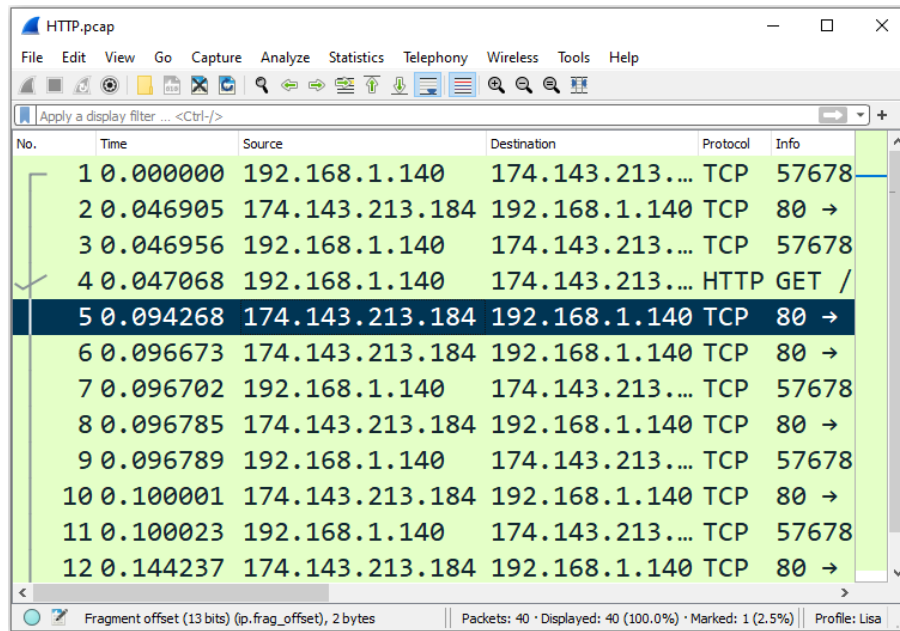


Figure 4.15 – Marking a packet

The following is a list of options that you can use when marking packets:

- **Mark/Unmark Packet:** This allows you to mark a specified packet or packets, which turns the packet(s) black for easy visual reference.
- **Mark All Displayed:** This marks all displayed packets. For example, if you use a display filter, Wireshark will only mark the packets that are displayed.
- **Unmark All Displayed:** If all displayed packets are already marked, then this will unmark all displayed packets.
- **Next Mark:** When the packets are marked, this option allows you to move to the next marked packet.
- **Previous Mark:** When the packets are marked, this option allows you to navigate back to the previously marked packet.

In addition to marking packets to identify items of interest, you might want to ignore specific packets. The following list describes how you can select specific packets to ignore while doing your analysis:

- **Ignore/Unignore Packet:** This allows you to select a packet, and once selected, it will be as if the packet never existed. The packet won't show up in statistics or a flow graph; it's simply ignored. Once selected, the packet line will have a reference that reads **<Ignored>**, as follows:

41	0.26	23.62.105.87	172.16.133.41	TCP	http(80) → 52678
42	0.26				<Ignored>
43	0.32	23.62.105.87	172.16.133.41	TCP	http(80) → 52678

Figure 4.16 – Using the Ignore Packet option

- **Ignore All Displayed:** This ignores all displayed packets, meaning if you use a display filter, Wireshark will only ignore the displayed packets.
- **Unignore All Displayed:** If the displayed packets are ignored, when selected, Wireshark will unignore all displayed packets.

While some packets might be ignored as they hold no value in the analysis, you might want to use other methods to determine delays, as we'll explore next.

## Setting a time reference

In your analysis, you might have a group of packets where you want to see exactly how long the delay was within those packets. In Wireshark, you can set a time reference on the packet where you think the trouble began and watch the time values to see gaps in the transmission. Wireshark provides a variety of ways to set a time reference and then offers ways to navigate through the references. Options include the following:

- **Set/Unset Time Reference:** This allows you to set/unset a time reference.
- **Unset All Time References:** This will unset all time references.
- **Next Time Reference:** Once a reference has been set, this allows you to navigate to the next time reference.
- **Previous Time Reference:** Once a reference has been set, this allows you to navigate to the previous time reference.

If you need to adjust the time reference, you can use the **Time Shift** option.

## Shifting time

If, during your analysis, you need to merge two captures that each used a different file format, you might want to use **Time Shift**. For example, if one file used the **Network Time Protocol (NTP)** and the other file used the **Precision Timing Protocol (PTP)**, this option will help to sync up the files.

Once you select this option, it will launch a dialog box where you can set your values, as shown in the following screenshot:

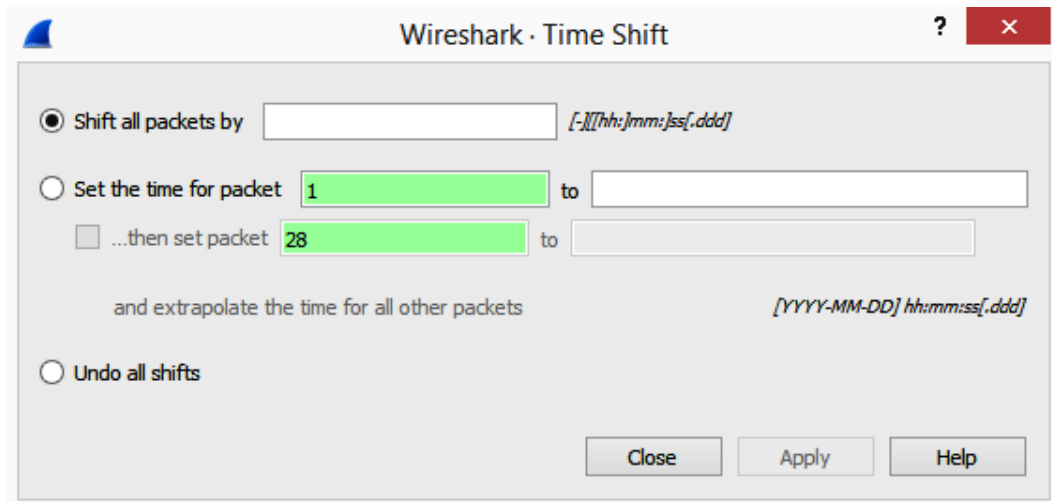


Figure 4.17 – The Time Shift option

The last option gives you the ability to undo all shifts if you get unexpected results.

Now that we understand how we can reference or shift time in Wireshark, let's take a look at ways in which to personalize your work area.

## Personalizing your work area

While working with a capture, you can record your changes by using comments or modifying the look and feel of your workspace.

When working with comments, the following choices are available:

- **Packet Comments:** This allows you to include comments within a single packet.
- **Delete all Packet Comments:** This removes all comments.

In addition, you can fine-tune the interface in the following ways:

- **Configuration Profile:** This allows you to create a customized profile that is specific to your workflow. This is a powerful feature, as you can create several profiles so that they can be used for specific applications or clients.
- **Preferences:** This is where you can adjust the font and color or even the layout. We'll cover this in more detail in *Chapter 6, Personalizing the Interface*.

Although the **Edit** menu is widely used, let's take a look at the **View** menu so that you can see the many ways in which to modify the look and feel of your capture during analysis.

## Exploring the View menu

The **View** menu is where you can alter the appearance of the captured packets, and it includes ways to colorize packets, expand subtrees, or show a packet in a separate window.

Let's start with ways to adjust the toolbars and panels and how to go into full screen mode. If you would like to follow along, use the `HTTP.pcap` file.

## Enhancing the interface

In Wireshark, there are several ways to alter and enhance the interface, including how we view the toolbars and what panels we would like to be visible. We'll start at the top with the toolbars.

The toolbar section represents a grouping where similar items are combined in many menus. Once you are in this section, you will see a list of three toolbars that are currently available, as follows:

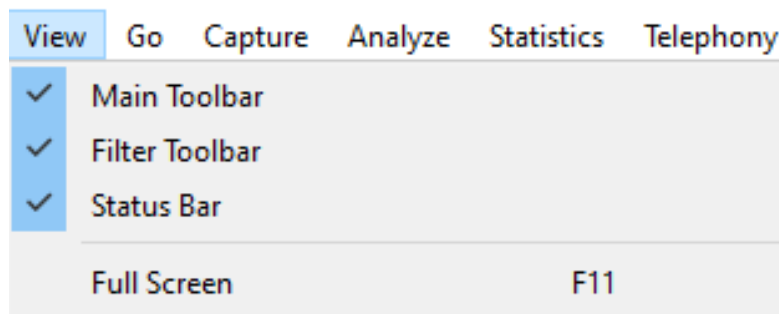


Figure 4.18 – The View menu toolbars and Full Screen options

If you see a checkmark, as shown in the preceding screenshot, that indicates the toolbar is visible. The toolbars are explained as follows:

- **Main Toolbar:** This holds all of the commonly accessed icons, as follows:

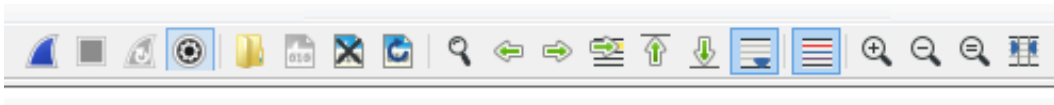


Figure 4.19 – The main toolbar

- **Filter Toolbar:** This is where you will find the display filter.
- **Status Bar:** This toolbar is found at the bottom of the Wireshark screen and lists information specific to the active file. Information includes what filter has been applied, how many packets have been captured and how many are displayed, along with what profile has been applied, as shown in the following screenshot:

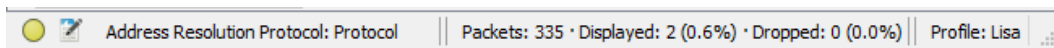


Figure 4.20 – The status bar

- The next menu choice is **Full Screen**, which will cause Wireshark to fill the current window.

Once you get used to the toolbars, you will see that they provide a handy way to help you navigate the interface. Now, let's take a look at the next grouping, which is the panel view, so you can modify what is visible on the screen. A checkmark indicates the panel is visible. If you do not want a panel to be visible, uncheck the panel and it will be hidden from view:

- **Packet List:** This is a list of all of the captured packets, where each line represents a single packet.
- **Packet Details:** This displays the details of a single packet.
- **Packet Bytes:** This is a hexadecimal representation of a single packet.
- **Packet Diagram:** This panel will show each packet with a diagram of each of the headers, as follows:

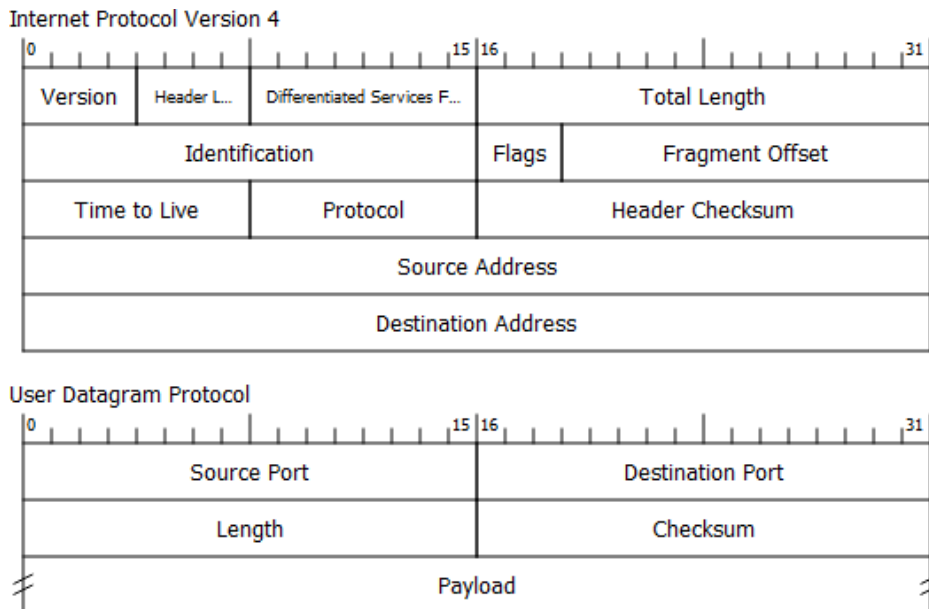


Figure 4.21 – A packet diagram view

The next section outlines the options for displaying time values in Wireshark, along with how to provide name resolution.

## Formatting time and name resolution

The **Time Display Format** and **Name Resolution** menu choices both have several options within their submenus. We'll start with the **Time Display Format** option, which provides several ways to view the time values in Wireshark.

### Displaying time

Once you expand the **Time Display Format** menu choice, you will see several options regarding how you want your time displayed. The options include **Date and Time of Day**; **Year, Day of Year**, and **Time of Day**; and **Time of Day** and **Seconds Since 1970-01-01**.

When carrying out an analysis, most likely, you will use a format that allows you to visualize any gaps in transmission. In that case, the following options are used:

- **Seconds Since Beginning of Capture:** This will show you how many seconds have passed since the capture was started.
- **Seconds Since Previously Captured Packet:** This will show you how many seconds have passed since the previously captured packet.



- **Seconds Since Previously Displayed Packet:** This is used when you apply a display filter, as it will show you how many seconds have passed since the *previously displayed packet*, which will more accurately show gaps in time.
- Other options include the **Coordinated Universal Time (UTC)** formats, as follows:

UTC Date and Time of Day (1970-01-01 01:02:03.123456)	Ctrl+Alt+7
UTC Year, Day of Year, and Time of Day (1970/001 01:02:03.123456)	
UTC Time of Day (01:02:03.123456)	Ctrl+Alt+8

Figure 4.22 – The UTC display options

Time precision is also a consideration. When selecting a format, you have a choice of how many decimal places are displayed. There are several formatting options, as shown:

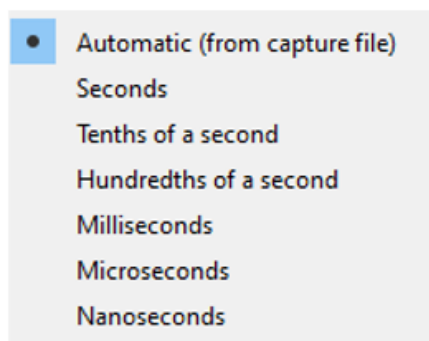


Figure 4.23 – The time precision options

Most of the time, it is best to use **Automatic**, which is the default, and that will be the best precision that the OS can provide.

The last option is **Display Seconds With Hours and Minutes**, which, when set, will appear as follows:

No.	Time	Source	Destination
58	0.000065s	10.0.0.75	52.104.22.55
59	0.153580s	10.0.0.101	10.0.0.255
60	0.204327s	10.0.0.101	255.255.255.255
61	0.000000s	10.0.0.101	224.0.0.1

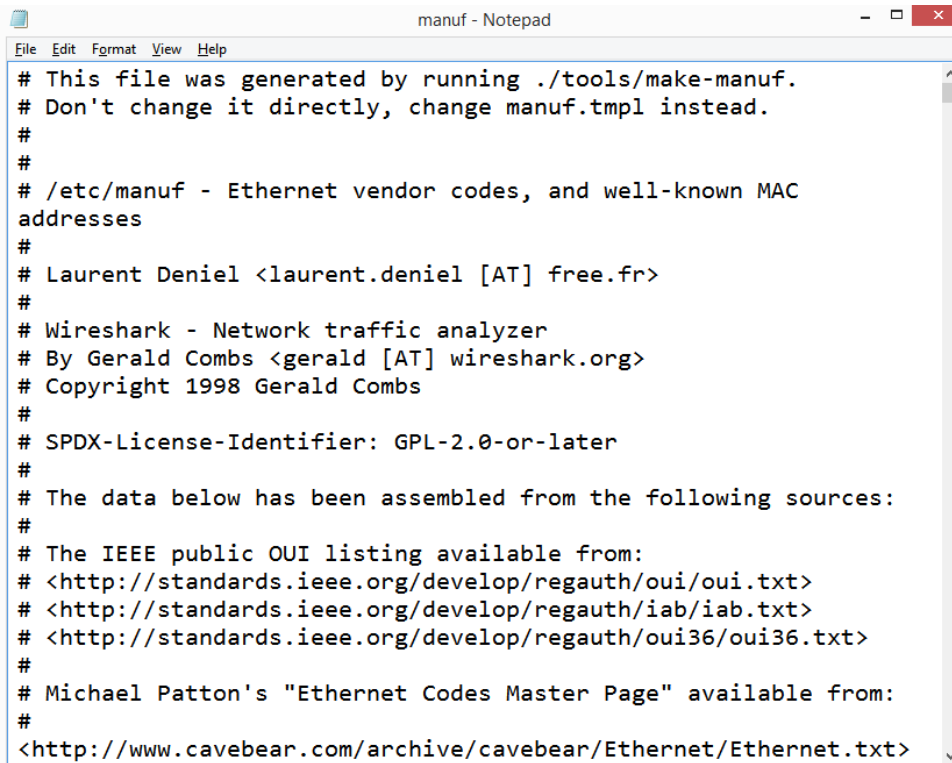
Figure 4.24 – Display Seconds With Hours and Minutes

The whole concept of time is important in packet analysis. Now you understand how you can easily modify the way time is represented. **Name Resolution** is another menu choice that has several selections. The following section will outline the options available to resolve names and the rationale behind why you would select each one.

## Resolving names

Under the **Name Resolution** menu, you can resolve physical, network, and transport addresses. In most cases, Wireshark can resolve physical and transport addresses without any problem, as they both come from a file found in the local Wireshark folder.

To resolve physical addresses, Wireshark looks at the first six digits of a MAC address, which is the **Organizational Unique Identifier (OUI)**. The resolution comes from the `manuf.txt` file, as shown here:



```
manuf - Notepad
File Edit Format View Help
# This file was generated by running ./tools/make-manuf.
# Don't change it directly, change manuf.tmpl instead.
#
#
# /etc/manuf - Ethernet vendor codes, and well-known MAC
addresses
#
# Laurent Deniel <laurent.deniel [AT] free.fr>
#
# Wireshark - Network traffic analyzer
# By Gerald Combs <gerald [AT] wireshark.org>
# Copyright 1998 Gerald Combs
#
# SPDX-License-Identifier: GPL-2.0-or-later
#
# The data below has been assembled from the following sources:
#
# The IEEE public OUI listing available from:
# <http://standards.ieee.org/develop/regauth/oui/oui.txt>
# <http://standards.ieee.org/develop/regauth/iab/iab.txt>
# <http://standards.ieee.org/develop/regauth/oui36/oui36.txt>
#
# Michael Patton's "Ethernet Codes Master Page" available from:
#
# <http://www.cavebear.com/archive/cavebear/Ethernet/Ethernet.txt>
```

Figure 4.25 – The `manuf` file listing NIC card vendors

To resolve the transport address (or port number), Wireshark consults the `services.txt` file, which holds a list of services and the associated port number. For example, the **Simple Mail Transport Protocol (SMTP)** service uses port 25. When Wireshark identifies that port 25 is in use, it will display SMTP as the service, as long as you have requested name resolution.

The file uses the **Internet Assigned Numbers Authority (IANA)** port-number file for consistency and can be found in the Wireshark folder, as shown in the following screenshot:

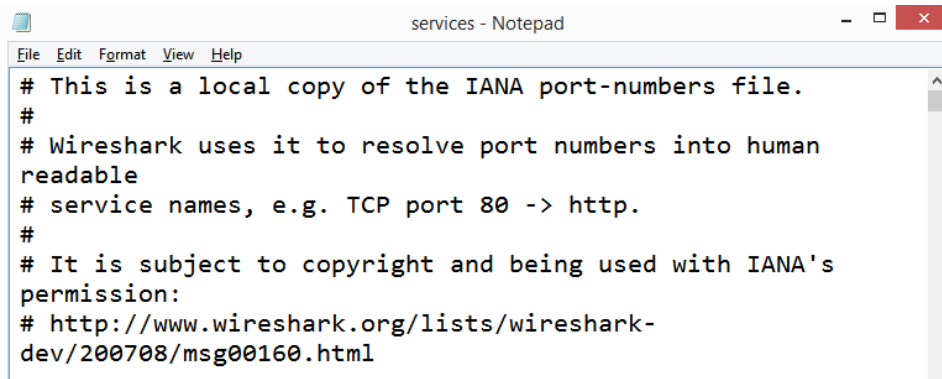


Figure 4.26 – The services file listing ports and associated services

The **Resolve Network Addresses** option will resolve a hostname to an IP address. Normally, this option is not checked because, if it is, Wireshark will ask the OS to contact the DNS server(s) to implement the resolution. This activity will then create a lot of additional network traffic.

If necessary, it is possible to change either the `manuf` or `services` files. In addition, you can also select **Edit Resolved Names**, which will bring up a **Name Resolution Preferences** toolbar where you can edit or add a name.

When working with a capture, there are ways to enhance your view, as we will learn in the next section.

## Modifying the display

To see the details of your capture, there are a few enhancements that include the ability to zoom in, expand the subtrees, and colorize the conversation:

- **Zoom:** This allows you to zoom in, zoom out, or return to normal size.

- **Subtrees:** Within a packet capture, Wireshark will collapse the details of a protocol header. When you expand the subtree, you can see the details of the protocol. As shown in the following screenshot, the expanded UDP subtree provides a detailed view of all of the field values in the UDP header:

```

User Datagram Protocol, Src Port: 57899 (57899), Dst Port: https (443)
  Source Port: 57899 (57899)
  Destination Port: https (443)
  Length: 1358
  Checksum: 0xbb69 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]

```

Figure 4.27 – A UDP header with an expanded subtree

Once done, you can collapse the subtree. In addition, you can also expand and/or collapse *all* subtrees.

To improve visibility and or highlight specific conversations, you can also use color.

## Colorizing packets

Within Wireshark, there are several ways to use color. Coloring formats include the following:

- **Colorize Packet List:** This is a shortcut to turn the coloring rules on or off. Additionally, you can find an icon on the main toolbar underneath the **Telephony** menu, as follows:

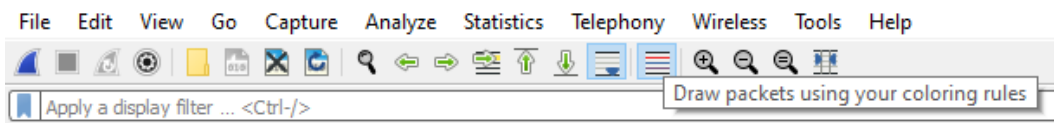


Figure 4.28 – The Colorize Packets icon

- **Coloring Rules:** This opens a dialog box where you can modify the coloring rules or create a new coloring rule.

- **Colorize Conversation:** This will colorize a conversation between two endpoints. You will have a choice as to what you would like to colorize, such as **Ethernet**, **IPv4**, or **UDP**—along with providing a choice of colors from which you can select, as shown in the following screenshot:

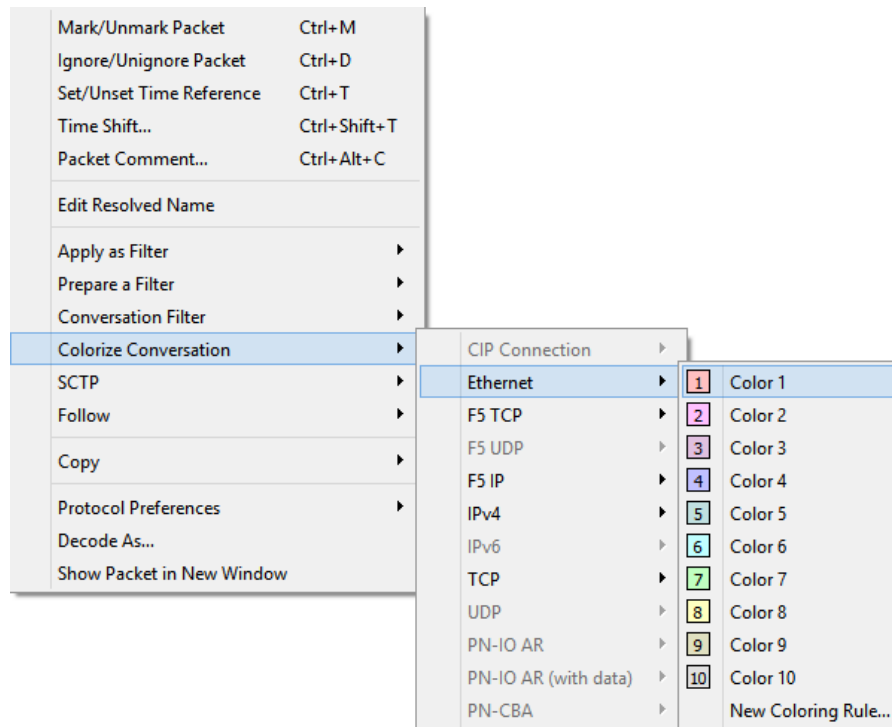


Figure 4.29 – Colorize Conversation

The last grouping of menu choices provides ways to refresh the view to reload, resize, show the packet in a new window, or view the internals.

## Refreshing the view

Wireshark doesn't limit the way you can view the data in the interface. In fact, in this last section, we'll see the many options that allow you to view the captured packets:

- **Resize Layout:** This option, when selected, will resize the visible panels so that they have a uniform appearance.
- **Resize Columns:** In the same way that you can resize the columns in Excel to automatically fit the contents (AutoFit), this option will adjust the columns so that the contents fit. When using IPv4, the columns could adjust nicely, but using IPv6 takes up much more space and might not give you an optimal view.

If you are a developer, the next section outlines what is available behind the scenes to allow Wireshark to dissect and display the various protocols.

## Using developer tools

The **Internals** menu choice provides advanced options that include the following submenu choices:

- **Conversation Hash Tables:** This shows the address and port combinations that identify each conversation. Select a single packet, and then click on this option to bring up the information, as shown in the following screenshot:

### Conversation Hash Tables

conversation\_hashtable\_exact, 2 entries

Address 1	Port 1	Address 2	Port 2
10.0.0.148	55578	204.79.197.213	443
2601:98b:4402:20cd:44ff:2c35:1982:eeae	57899	2607:f8b0:4004:80f::2004	443

conversation\_hashtable\_no\_addr2, 0 entries

conversation\_hashtable\_no\_port2, 0 entries

conversation\_hashtable\_no\_addr2\_or\_port2, 0 entries

Figure 4.30 – The Conversation Hash Tables information

- **Dissector Tables:** This provides tables of subdissectors for each of the supported protocols, as shown in the following example:

http	
Compuserve GIF	GIF image
Distributed Computing Environment / Remote Proce...	DCERPC
eXtensible Markup Language	XML
HyperText Transfer Protocol 2	HTTP2
JPEG File Interchange Format	JFIF (JPEG) image
Portable Network Graphics	PNG
WebSphere MQ	MQ

Figure 4.31 – The Dissector Table showing HTTP subdissectors

Within the table, you will also find the full name as well as the short name of the subdissector. For example, within the list of HTTP subdissectors, the short name for **JPEG File Interchange Format (JPEG file in HTTP)** is **JFIF (JPEG) image**.

- **Supported Protocols:** This will bring up an extensive list of all currently supported protocols and the protocol fields, along with the suggested filter and a brief description.

The last few options will either freshen the view or reload the capture.

## Reloading the packets

When using Wireshark, there is no shortage of ways to present the data. Some additional view options include the following:

- **Show Packet in New window:** Often, you might want a single packet in its own window as a popup. This option might be ideal for training purposes when you want to show the details of a single packet.
- **Reload:** This option will reload the capture, which will freshen the capture file. This is helpful if you have marked packets and have already manipulated the file, and you want a fresh start with the file.

The **Reload as File Format/Capture** option will give you a view *inside the pcap*. Using the HTTP.pcap file, I modified the layout and then selected the option, which displayed this view:

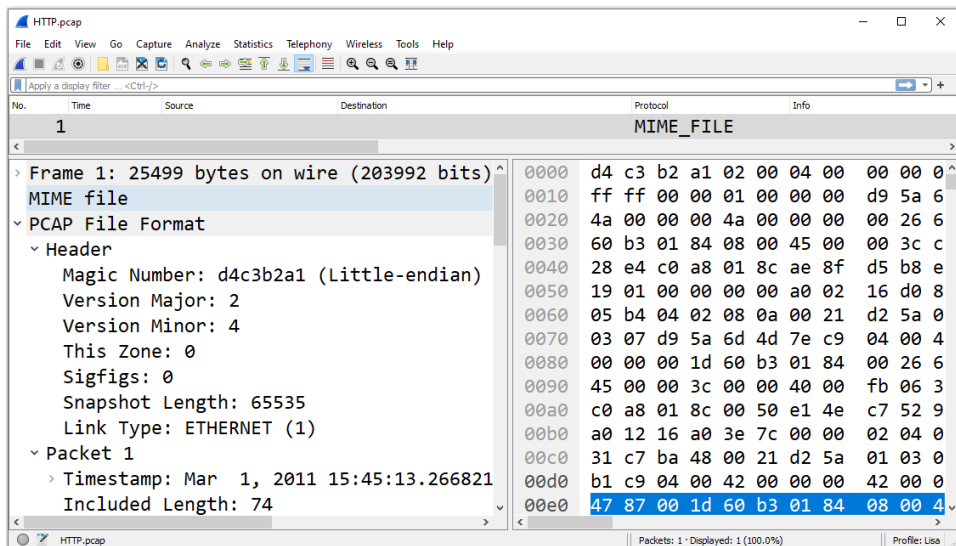


Figure 4.32 – Reloading as File Format/Capture

Whether you are a developer, network administrator, or student, you can appreciate the many flexible ways Wireshark provides to display and format data while working with a packet capture.

## Summary

In this chapter, we explored all of the elements of the Wireshark welcome page, to give you a better understanding of what is available, even before opening a packet capture. We also took a closer look at commonly accessed menu choices to make navigating around Wireshark easier. First, we evaluated the **File** menu, which has all of the tasks commonly associated with working with a file.

Next, we studied the **Edit** menu, which allows you to find and mark packets, set a time reference, or modify your preferences. We concluded with the **View** menu, where you can alter the appearance of the captured packets, including how to colorize them, zoom in, or show a packet in a separate window.

In the next chapter, we will learn where and how to tap into a data stream. Because what you see when capturing data will depend on the type of network you are accessing, we will review the different network architectures. Then, when you are ready to capture, we will discover the various capture options, such as using multiple files and directing output. We'll also compare the difference between conversations and endpoints and finish with stressing the importance of baselining the network to help when conducting a troubleshooting exercise.

## Questions

Now, it's time to check your knowledge. Select the best response and then check your answers, which can be found in the *Assessments* appendix:

1. Once you begin capturing packets, you might have a dozen or so files in the Open file area. If you want to remove the files, go to the File menu. Then, choose Open Recent and \_\_\_\_\_ Menu.
  - A. Clear
  - B. Purge
  - C. Delete
  - D. Freshen



2. Seconds Since \_\_\_\_\_ is used when you apply a display filter, as it will show how many seconds have passed since the previously displayed packet, which will more accurately show gaps in time.
  - A. Recently Created Epoch
  - B. Previously Captured Packet
  - C. Beginning of Capture
  - D. Previously Displayed Packet
3. \_\_\_\_\_ is a shortcut to turn the coloring rules on or off. The shortcut is also available on the main toolbar (under the Telephony menu).
  - A. Colorize Conversation
  - B. Coloring Rules
  - C. Stop Color Filters
  - D. Colorize Packet List
4. The \_\_\_\_\_ menu choice in Wireshark allows you to control the look of the displayed packets, including the ability to zoom in, colorize packets, and show a packet in a separate window.
  - A. File
  - B. Edit
  - C. View
  - D. Go
5. When working with a packet capture, the \_\_\_\_\_ menu choice edit allows you to find and mark packets, set a time reference, copy, provide detailed information for creating a configuration profile, or modify your preferences.
  - A. File
  - B. Edit
  - C. View
  - D. Go

6. The \_\_\_\_\_ is found at the bottom of the Wireshark screen. This toolbar is found at the bottom of the Wireshark screen and lists information specific to the active file. Information includes what filter has been applied, how many packets have been captured, and how many are displayed, along with what profile has been applied.
- A. Capture Toolbar
  - B. Main Toolbar
  - C. Status Bar
  - D. Filter Toolbar
7. At times, you might want a single packet in its own window as a popup. In that case, you should use the \_\_\_\_\_ option.
- A. Load Developer View
  - B. Launch Plain Text
  - C. Reload Layout
  - D. Show Packet in New Window



# Part 2

# Getting Started with Wireshark

In this section, we'll cover the various options when tapping into the data stream. We'll also outline ways to personalize the Wireshark interface to improve your workflow. We'll then compare the use of display and capture filters, and then review the OSI model and data encapsulation process.

The following chapters will be covered under this section:

- *Chapter 5, Tapping into the Data Stream*
- *Chapter 6, Personalizing the Interface*
- *Chapter 7, Using Display and Capture Filters*
- *Chapter 8, Outlining the OSI Model*



# 5

## Tapping into the Data Stream

Wireshark provides the ability to capture and analyze network traffic. It is used by network administrators and security analysts in a wide variety of industries, governments, and non-profit organizations. Prior to analysis, you'll need to tap into the data stream and capture packets from the network. Once captured, you can analyze the packets to understand the traffic flow. In this chapter, we'll review the different network architectures, along with the various types of media that can be found on today's networks. Once outlined, this will help you get a better understanding of the complex nature of today's networked environment.

So that you can confidently begin capturing traffic, we'll look at the various options, including capture, input, and output. We'll then review what happens when you tap into a network so that you can identify what types of traffic you'll see. We'll also compare and contrast conversations and endpoints. Finally, so you can better identify abnormal behavior, this chapter ends with a discussion of the importance of baselining network traffic.

This chapter will address all of this by covering the following topics:

- Reviewing network architectures
- Learning various capture options
- Tapping into the stream
- Realizing the importance of baselining

## Reviewing network architectures

We live in an exciting yet challenging period in history. Today, our internet-based ecosystem demands that business networks are available nearly 100 percent of the time. Enterprise networks must be able to adjust to changing traffic demands and maintain constant response times. In addition, they have to be agile enough to respond to unexpected security incidents.

Effective packet analysis begins with understanding the network architecture. In order to determine where to tap in to identify trouble spots, it's important to recognize the way that different media and devices influence network traffic. In this section, we will compare the different types of networks in use today, along with the various types of media used to transport data.

Let's begin our discussion by outlining the diverse types of networks in use today.

## Comparing different types of networks

Today's networked environments are complex and can include data from mobile phones, cloud computing, virtualization, social media, and the **Internet of Things (IoT)**. The network specialist deals with many different types of networks, which include **Personal Area Networks (PANs)**, **Local Area Networks (LANs)**, **Campus Area Networks (CANs)**, and **Wide Area Networks (WANs)**. All of these different types of networks influence how data is transmitted.

To begin, we will review the smallest network, a PAN, which you may encounter in your analysis.

### Discovering PANs

A PAN is a network that shares data between devices that are close, normally within a range of 30 feet. Devices can connect to the internet or other networks. Because devices in a PAN generally communicate using low-powered wireless technology, a PAN is also referred to as a **Wireless Personal Area Network (WPAN)**.

A WPAN is a short-range network that connects personal devices to exchange information using the IEEE 802.15 standard and includes technologies such as Bluetooth, Zigbee, and ultra-wideband.

Conducting packet analysis on a PAN may be done to troubleshoot or test IoT devices that connect to the internet, enabling them to send and receive data. Using Wireshark, you can study protocols such as **Message Queuing Telemetry Transport (MQTT)**, a lightweight messaging protocol used for machine-to-machine communication.

One of the most common types of networks where you will capture traffic is a LAN. The following section provides an overview of the characteristics of a LAN.

## Checking out LANs

A LAN is a private network in a localized area that an organization or individual owns, controls, and manages. A LAN is generally within a restricted geographic area, such as a corporate office, manufacturing plant, or healthcare facility, and provides the ability for hosts to share resources.

A LAN provides high-speed bandwidth using Ethernet technology on a fixed frequency, connecting network devices and enabling the ability to communicate and exchange data on a common channel.

Within a LAN, there might be a data center, which is a large group of servers that provide storage, processing, and distribution of critical company data for network clients. The data center is at the heart of any enterprise network and is located in a central location, generally in a secure computer or server room.

In today's large, multifaceted companies, there may be a larger network than a LAN that requires remote locations to serve all of the clients. The following section takes a look at the concept of a CAN.

## Exploring CANs

A CAN is a large, private LAN in a common entity, such as a college, hospital, corporate campus, or military base, that has two or more interconnected LANs.

A CAN has a main campus where the central elements of the network reside, such as the data center and telephony, and provides connectivity, data, applications, and services to clients. In addition, a CAN might include remote locations that are away from the main campus.

Because a CAN, at times, is spread across a larger geographic area such as a city, remote locations will need to communicate over a WAN using an internet connection. Let's now discover the qualities of a WAN.



## Navigating WANs

A WAN is a geographically dispersed collection of LANs that span a large distance. The internet is the largest WAN, spanning the globe, and is a network of globally connected networks that bring people, processes, data, and things together.

A WAN is different than a LAN in several ways. In most cases, no one entity owns a WAN; rather, WANs exist with shared or distributed ownership and management. WANs use common technology such as **Multiprotocol Label Switching (MPLS)**, which is a data transport method for high-performance telecommunication networks. WANs can carry a signal using a variety of methods, which include the **Plain Old Telephone Service (POTS)**, fiber-optic cables, wireless transmissions, and satellites.

As you can see, there are many different types of networks. In the next section, we'll explore each of the different types of media used to carry the signals.

## Exploring various types of media

Devices on a network share access to a common medium that provides a channel for traffic to travel. Media can be either of the following forms:

- **Bounded signals** are controlled or confined to a specific path by traveling over a copper or fiber-optic cable.
- **Unbounded signals** travel using a wireless radio wave.

The following is a diagram that represents various types of network media:

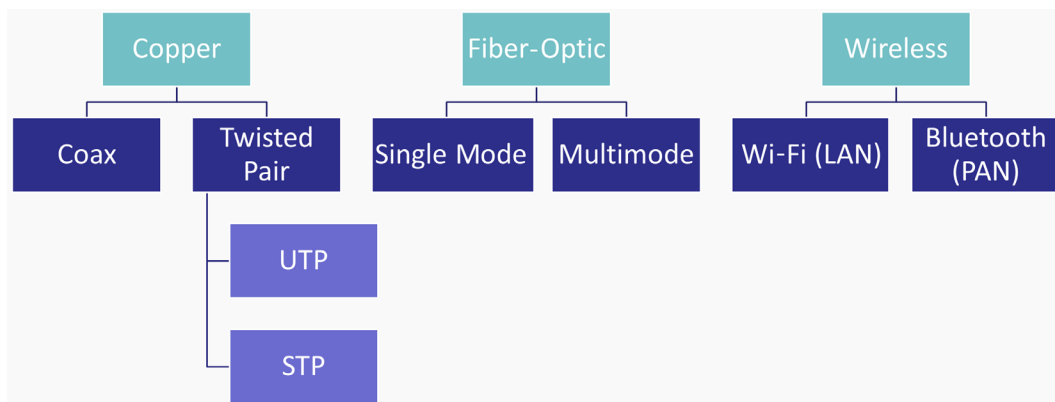


Figure 5.1 — Various types of network media

For enterprise networks, it's common for multiple types of media to make up the networking environment. Each media type will influence the data flow and can include copper and fiber-optic cables, along with wireless transmissions.

Network administrators use Wireshark for troubleshooting network connectivity issues. Because of this, it's important to recognize some of the problems that may occur because of the physical layer (layer one) of the **Open Systems Interconnection (OSI)** model. In this section, we'll take a look at the different types of media that can transmit data. Let's begin by reviewing copper, which is subdivided into two categories, coaxial and twisted pair.

## Understanding copper

Copper is a commonly used media type in today's networks for data communications. The two types of media that use copper are **coaxial** and **twisted pair**.

### Transmitting over coaxial

Coaxial, also called **coax**, consists of a single copper wire encased by a layer of insulation and then by a grounded shield of braided wire. Coax is able to support high bandwidth and was originally the primary way to transmit data on a LAN.

Coax is no longer used by LANs to transmit data. However, you will still see coax, as it is used by cable television companies to transmit signals to clients in homes and businesses.

Although rare, it is possible to troubleshoot the differences in traffic transmitted between the cable modem and router, as Wireshark has a **Data Over Cable Service Interface Specification (DOCSIS)** dissector for that purpose.

### Communicating with twisted pair

This type of cable consists of twisted pairs of copper wire that use pulses of electricity to carry a signal. The twists provide a shielding effect that minimizes crosstalk.

Twisted pair cabling has eight wires with four pairs of twists and comes in two forms:

- **Unshielded Twisted Pair (UTP)**: This is the most commonly used wire.
- **Shielded Twisted Pair (STP)**: This is used when protection from **Electromagnetic Interference (EMI)** is necessary.

Today, LANs use twisted-pair cables to transmit data. Twisted-pair cabling is so popular because it is reasonably priced, easy to install, and in most cases, provides high bandwidth for carrying both data and multimedia traffic.

In addition to copper, many companies employ fiber within their organization to provide a high-speed, high-bandwidth option over copper. The following section outlines the characteristics of fiber, which is subdivided into two categories, multimode and single mode.

## Using fiber optic

A fiber-optic cable uses pulses of light to carry network traffic over longer distances. Fiber has high throughput that is naturally resistant to EMI. The signals are sent via laser or a **Light-Emitting Diode (LED)**, using a core of glass or plastic. Many times, fiber is used as the backbone on a LAN and comes in two forms:

- **Multimode (MMF):** This uses multiple light signals, has a higher bandwidth than UTP, and is used to carry backbone traffic in a LAN. MMF can use either glass or plastic, using either LED or laser signals, over a distance of up to 2 km.
- **Single mode (SMF):** This uses a single light signal. Single-mode fiber has a higher bandwidth than MMF and can carry a signal for many miles. SMF must use a laser to produce a bright, coherent light.

Fiber optic has many benefits, but it is more expensive than twisted pair and requires special equipment to manage. As a result, LANs use fiber primarily for backbone traffic and use twisted pair for work areas.

Today, it is common to see wireless network communication, which uses radio waves to transmit signals. The following section outlines the various ways you may work with Wireshark to analyze a wireless connection.

## Discovering wireless networks

Wireless networks use unbound media, which allows users to roam freely while still being connected to the network. Over time, wireless networks have improved in speed and bandwidth, and as a result, you will most likely capture wireless traffic during a troubleshooting exercise.

Wireless technology can provide connectivity for a LAN using Wi-Fi, or for a PAN using Bluetooth. Here, we will compare the two:

- **Wi-Fi** provides networking on a LAN using the family of IEEE 802.11 standards. Currently, the most widely used standards are 802.11a, 802.11b/g/n, and 802.11ac.
- **Bluetooth** provides networking on a PAN over short distances from fixed and mobile devices. The technology allows devices to communicate with each other to transfer files, control IoT devices, and provide hands-free calling in your car.

As you can see, there are many variables that you may deal with while capturing and analyzing traffic using Wireshark. The type of network and the media will influence how you capture traffic and what you might see once it has been captured. In most cases, however, packet capture using Wireshark is done on a LAN.

In the next section, we will explore how to properly set up a capture and examine each of the capture option tabs – **Input**, **Output**, and **Options**.

## Learning various capture methods

When capturing traffic with Wireshark, most of us are familiar with the main interface, where we go to the lower part of the screen to see what interfaces are active by viewing the sparklines. Once here, you can select an active interface and begin capturing traffic. In addition, you can put in a capture filter and begin capturing traffic.

In addition to the welcome screen, you can go to the **Capture** drop-down menu and then select **Options** to do advanced configuration before capturing traffic. Across the top, you will see three tabs, **Input**, **Output**, and **Options**, as shown in the following screenshot:

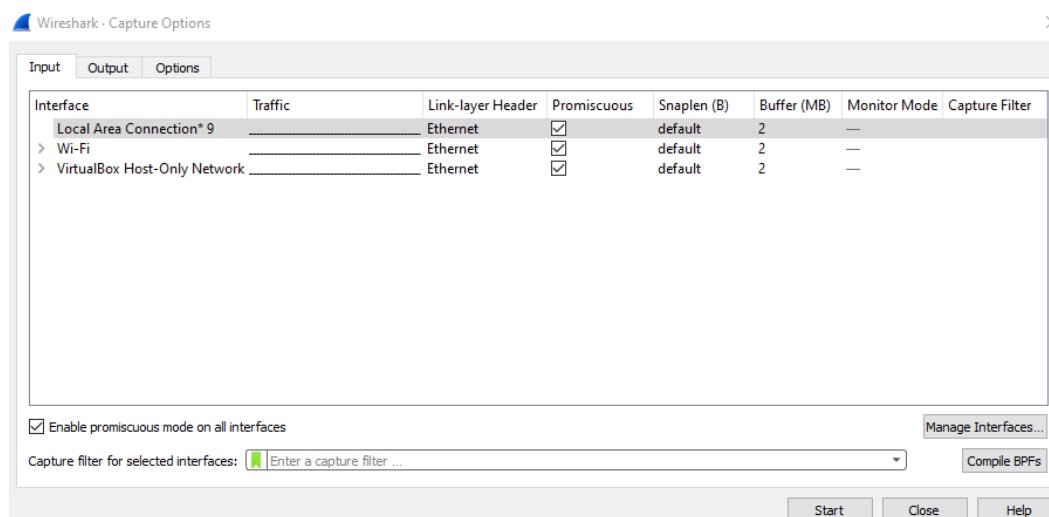


Figure 5.2 — Capture options

Let's start with a discussion on how to set up a capture by selecting an input interface.

## Providing input

In the **Capture Options** dialog box, the **Input** tab will show a list of available interfaces on your device. Across the top, you will see various column headers, which include **Interfaces**, **Traffic**, **Link-layer Header**, and **Capture Filter**.

In the lower-left corner, there is a checkbox called **Enable promiscuous mode on all interfaces**. If you uncheck the box, it will take off promiscuous mode on all interfaces. You can then select the interface you want to be in promiscuous mode by checking the box to the right of it.

Across the bottom, you can create a capture filter for the selected interface.

In the lower right, you can select **Manage Interfaces...**, which will allow you to hide interfaces that you do not want to be visible on the **Input** tab. For example, we can see five unchecked USBPcap interfaces in the following screenshot:

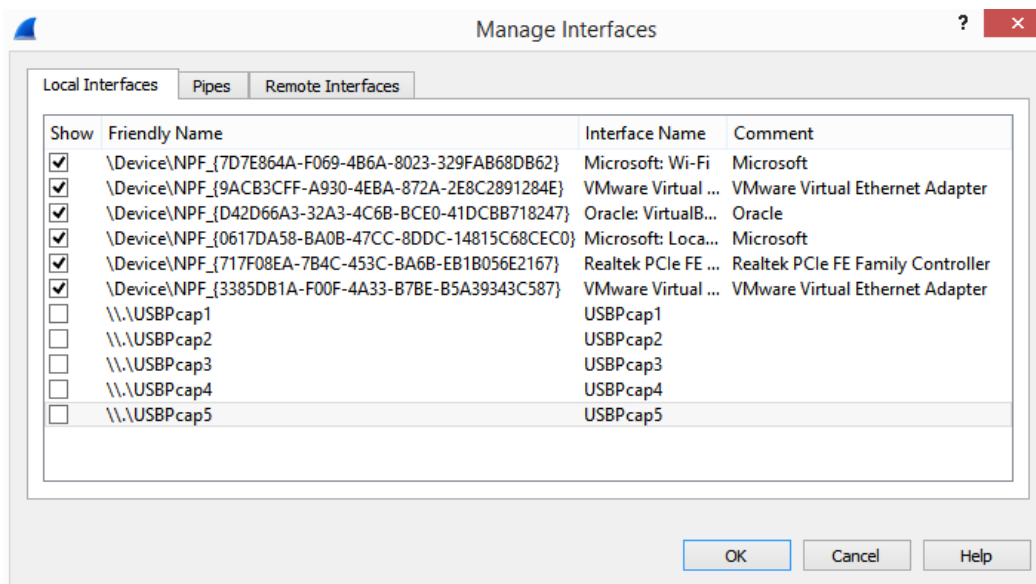


Figure 5.3 – Capture options – Managing Interfaces

Once you have selected what you would like for input, you may want to save your file in a specific way. The next section outlines the **Output** tab.

## Directing output

The **Output** tab directs where and how you want to save your file. Within this tab, there are several options.

The first option is **Capture to a permanent file**. In most cases, this box is left blank. When you begin capturing traffic, Wireshark will save the capture to a temporary file until you save it as something else.

**Output format** defaults as saving the file as **pcapng** (short for **PCAP Next Generation**); however, you can force Wireshark to save the file as **pcap**. Most of the time, **pcapng** is the best choice, as it allows you to add comments.

Whatever option you select, the next selection is **Create a new file automatically....** The options include the following:

- After  $n$  packets
- After  $n$  file size
- After  $n$  seconds, minutes, or hours
- When there is a multiple of  $n$  seconds, minutes, or hours

The following screenshot shows the **Output** tab of the **Capture Interfaces** dialog:

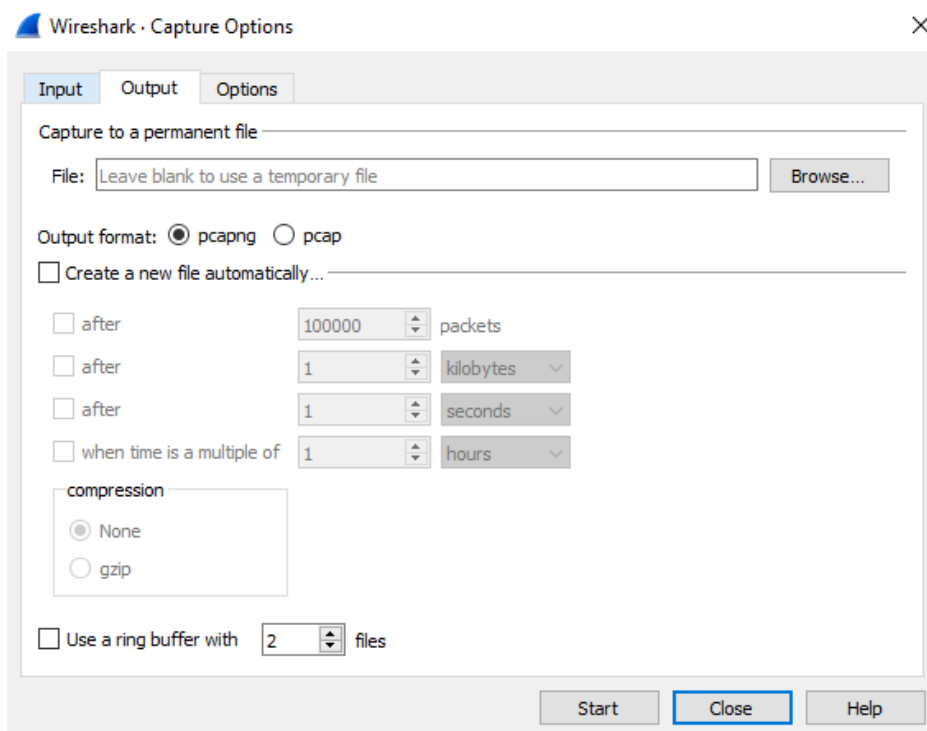


Figure 5.4 – Capture options – Output options

Wireshark also provides the ability to compress files using **GNU zip (gzip)**.

Although you may be tempted to launch Wireshark and let it run while monitoring traffic for a long period of time, that isn't the best option. This is mainly because Wireshark will consume all of your memory if you leave a capture running, as it holds the capture in a temporary file until you stop the capture and save it to a permanent file.

The next selection outlines how you can use a ring buffer to monitor traffic.

## Using a ring buffer

A ring buffer is handy if you want to run a capture to watch for a specific protocol or signature on your network. To use a ring buffer, you create multiple files and set a parameter to create a file automatically after either a specific file size is reached, such as after 1 megabyte, or after a period of time has passed, such as 10 seconds.

If you do want to create multiple files, you must specify a filename and location for the files; otherwise, you will throw an error, as shown here:

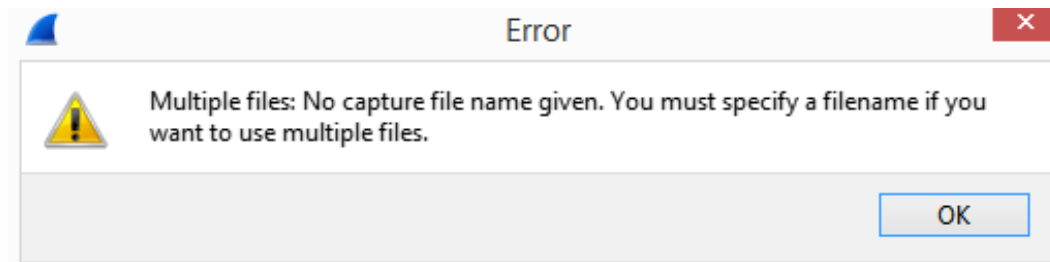


Figure 5.5 – An error message in the capture options

When you are ready, select **Use a ring buffer with** at the bottom of the dialog box and enter how many files you want to overwrite.

In addition to providing ways to select input and output options, Wireshark provides some custom options that you can modify. Let's take a look.

## Selecting options

When looking at the **Options** tab, you will see across the top **Display Options**, which can be set as follows:

- **Update list of packets in real-time**
- **Automatically scroll during live capture**
- **Show capture information during live capture**

In addition, you'll also find the **Name Resolution** choices, which include the following:

- **Resolve MAC addresses**
- **Resolve network names**
- **Resolve transport names**

In most cases, it's okay to resolve MAC addresses and transport names, as these are changed into a human-readable format using static text files found in the local Wireshark folder. The files include the following:

- `manuf.txt` is a list of Ethernet vendor codes and well-known MAC addresses.
- `services.txt` holds a local copy of the IANA port numbers file.

However, if you select **Resolve network names**, this will ask the host **Operating System (OS)** to contact the **Domain Name System (DNS)** server multiple times while resolving the IP addresses. This activity will most likely impact system performance and cause additional traffic on the network.

The last selection on the **Options** tab is **Stop capture automatically after...**, whatever option you select. There are four choices:

- Packets
- Files
- The size of files
- After a specified time period

This last option can be used when baselining, and you can specify to stop capturing after 1,000 packets and then start your capture; Wireshark will capture 1,000 packets and then automatically stop the capture after a specified time period.



All options are shown in the following screenshot:

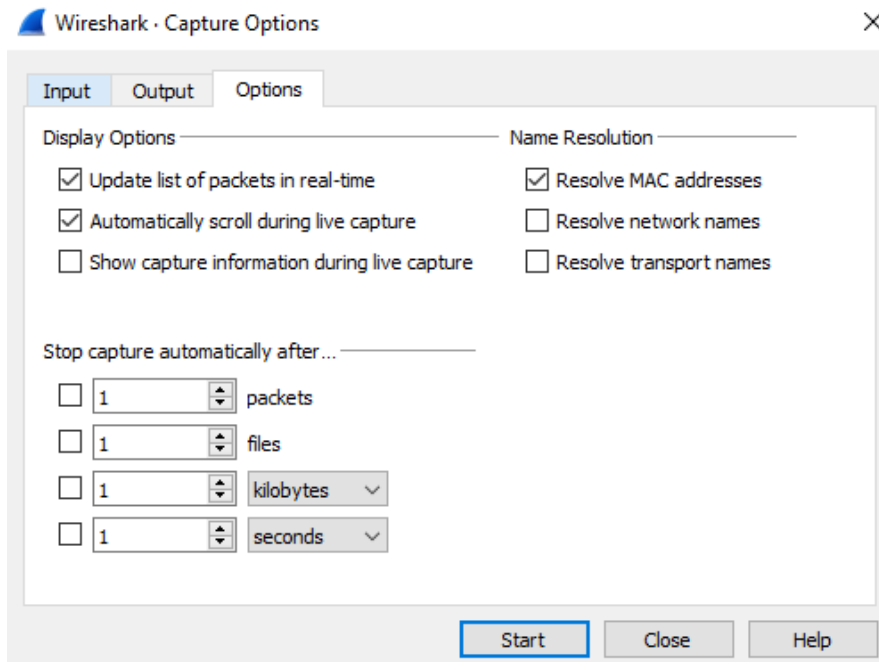


Figure 5.6 – Capture options – the Options tab

After you understand the network architecture and topology and have selected your capture options, you're ready to tap into the network. This next section will review the different types of packets you will see, along with how to look at the conversations and endpoints that are gathered while capturing traffic.

## Tapping into the stream

While tapping into a LAN with the **Network Interface Card (NIC)** in promiscuous mode, the adapter captures the traffic and sends the packets up through the **Enhanced Packet Analyzer (EPAN)** for dissection and decoding, and then on to the Wireshark interface.

You'll then see the packets filling the screen. If you are on an end device and communicating with another host, you will most likely see three types of packets – namely, broadcast, multicast, and unicast:

- **Broadcast:** Packets are sent from one host to everyone on a network – for example, an ARP broadcast.
- **Multicast:** Packets are sent from one host to many hosts – for example, using **Enhanced Interior Gateway Routing Protocol (EIGRP)** multicasts.
- **Unicast:** This sends packets from one host to another host– for example, from your computer to a web server.

In a normal conversation with another host, once you have a connection, the OS will create a socket, which consists of an IP address and a port. During a capture, Wireshark will keep track of all of the connections or streams, which you can examine.

This next section explains how you can take a look at the conversations and endpoints in a capture.

## Comparing conversations and endpoints

Whenever you are actively connecting with other hosts on the network, the OS keeps track of all the connections. To see all of your active connections on a Windows machine, open a command line and run `netstat` with the `-an` parameters, as shown in the following screenshot:

TCP	10.0.0.148:49559	17.249.124.141:5223	ESTABLISHED
TCP	10.0.0.148:49768	34.212.110.138:443	ESTABLISHED
TCP	10.0.0.148:62310	13.89.217.116:443	ESTABLISHED
TCP	10.0.0.148:62789	23.55.20.137:443	CLOSE_WAIT
TCP	10.0.0.148:62790	204.13.192.141:80	CLOSE_WAIT

Figure 5.7 – The `netstat` command showing Transmission Control Protocol (TCP) connections

In Wireshark, a conversation consists of two endpoints that are in a connection together. An endpoint is one side of the conversation. To view all of the conversations in a capture, go to **Statistics** and then **Conversations**. Once the window opens, there are tabs along the top that allow you to view a specific type of conversation.

Each tab provides details of the type of conversation you selected, and each row represents one conversation. For example, the **Ethernet** tab shows conversations listing the MAC addresses of the endpoints, as shown in the following screenshot:

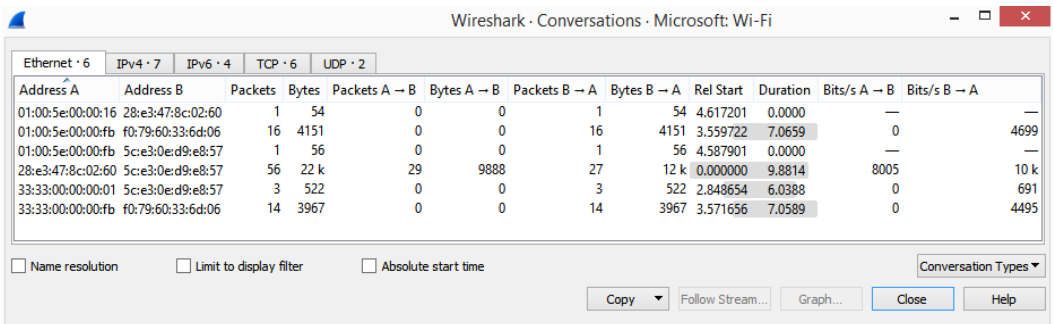


Figure 5.8 – Viewing conversations

You can always add or remove tabs by selecting **Conversation Types** in the lower right-hand corner, as shown here:

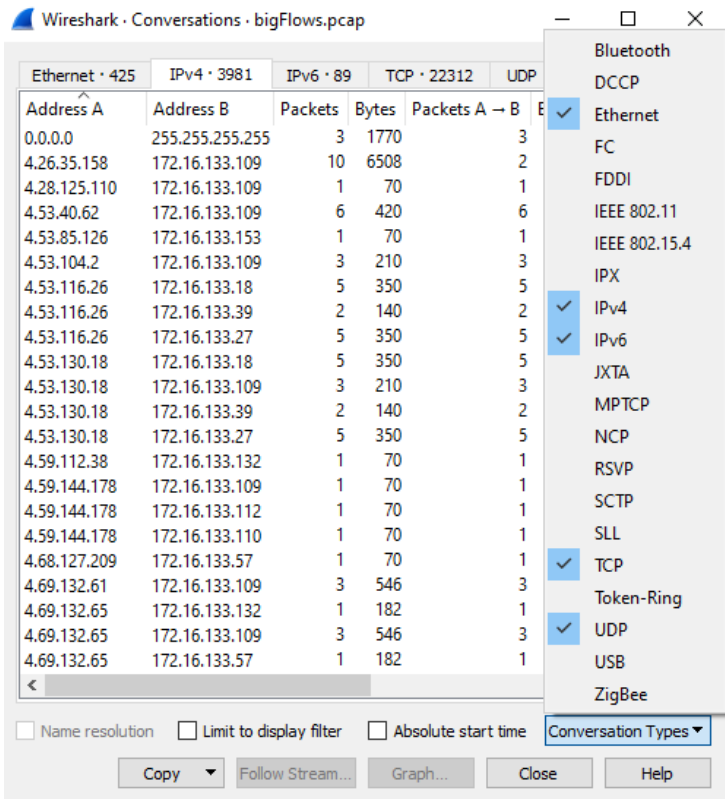


Figure 5.9 – Selecting conversation types

Wireshark has advanced options within this window. Right-click any of the conversations and you will see the following options:

- **Apply as a filter:** This will select the highlighted conversation and run the filter.
- **Prepare as a filter:** This will select the highlighted conversation and prepare the filter, which can then be modified. When you are done editing, you must press *Enter* to run the filter.
- **Find:** This will select the highlighted conversation and place the variables in the search toolbar.
- **Colorize:** This will select the highlighted conversation and allow you to create a custom coloring rule.

The following screenshot shows the search toolbar that is launched when you select **Find**:

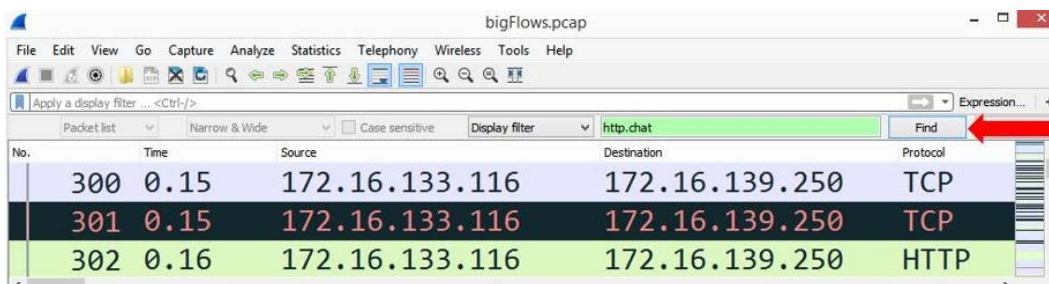


Figure 5.10 – The Find toolbar

All of these options allow you to further refine your selection. Right-click and select one of several options that include **A to B**, **B to A**, and **A to Any**.

At the bottom of the window, there are additional choices with which you can refine and customize your view:

- **Name resolution:** Wireshark will resolve the physical, network, and transport addresses for the specific conversation type. For example, if the **TCP** tab is selected, the transport address will be resolved.
- **Limit to display filter:** This will show only conversations included in the current display filter.
- **Absolute start time:** This will change the start time column to the absolute start time, which is in the **Time of Day** display format. If you uncheck this, the time will revert to the relative start time, which is in the **Seconds since Beginning of Capture** time display format.

- **Copy:** This will copy the list to the clipboard in either **Comma Separated Values (CSV)** or **Yet Another Markup Language (YAML)** format. You can then paste it into a notepad file or a spreadsheet.
- **Follow Stream:** This allows you to see the details of a single TCP or **User Datagram Protocol (UDP)** conversation.
- **Graph:** This will launch and display a TCP stream graph on the selected TCP conversation, as shown here:

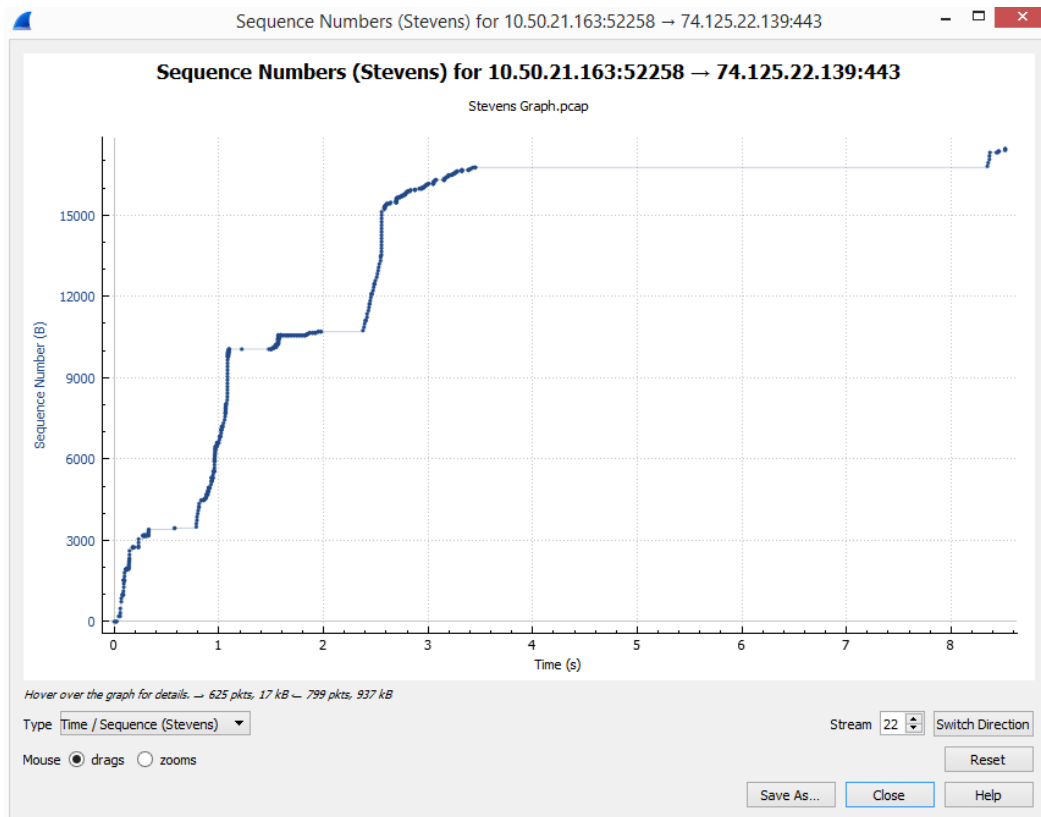


Figure 5.11 – A TCP stream graph

As you become more experienced with using Wireshark, you will be able to navigate around the interface with ease. Until then, experiment with some of the menu choices and options.

In order to troubleshoot a network more effectively, it is important to start with a packet capture so that you can compare possible changes. One way to achieve this is by creating a baseline, which we will cover in this next section.

## Realizing the importance of baselining

Every network is like a snowflake in that no two are alike. Each network has its own signature that includes characteristics such as utilization, network protocols, and latency issues.

A baseline is a packet capture on a subnetwork that is obtained using Wireshark or `tshark` during normal working conditions. If the network is experiencing problems, the network administrator can then use the baseline to identify any changes. Once you learn what normal network behavior is, you can better identify abnormal network behavior.

In addition to troubleshooting, a network baseline can be used for optimizing, forecasting, planning, and tuning a network. The baseline process goes through several stages – plan, capture, analyze, and save.

We will begin with planning, which provides steps for the best way to go through the process.

### Planning the baseline

To plan the baseline, create a network map and list all of the subnetworks and **Virtual Local Area Networks (VLANs)**. You should have a strategy on how you are going to go about the process. Some things to consider include the following:

- The time of day to run the capture
- Whether you are going to capture wireless or wired traffic
- Details about the location
- What applications may be in use on a particular subnetwork

Once planning is complete, we can then move on to the next step, which is where we actually capture network traffic.

### Capturing traffic

When it's time to capture the traffic, limit the packet capture to a consistent size so that you have a consistent capture size for the baselines. Determine what an appropriate capture would be for your network so that you can get a snapshot of the network at a slice in time.

The process should be documented – for example, document where the capture was obtained, the time of day, and what equipment was used during the capture.

The key is to be as consistent as possible with your captures so that you compare apples to apples. If you select **Statistics** and then **Capture File Properties**, you can add a comment to provide additional information about the capture, as shown in the following screenshot:

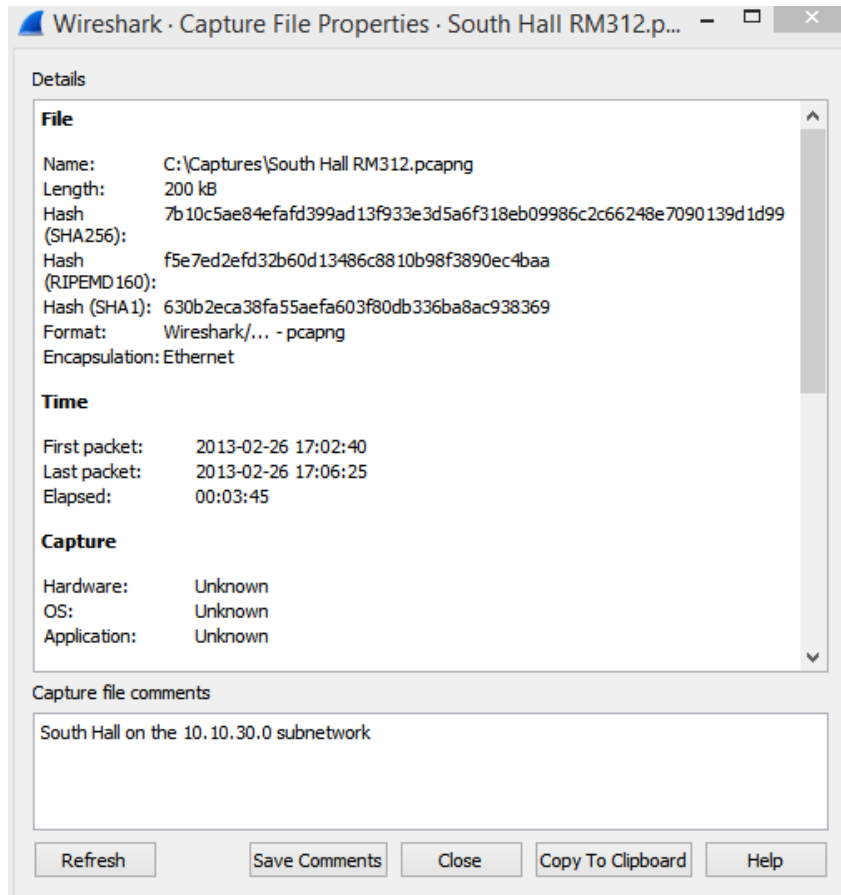


Figure 5.12 – The Capture File Properties window

After we complete the capture phase, we then move to the analyzing phase, where we will take a closer look at the capture.

## Analyzing the captured traffic

Once you have obtained the packet capture, take a moment, and review the capture to see whether anything stands out as unusual or suspicious. Within Wireshark, you can go to **Statistics** and then **Protocol Hierarchy** to spot-check what protocols appear on the subnetwork, as shown here:

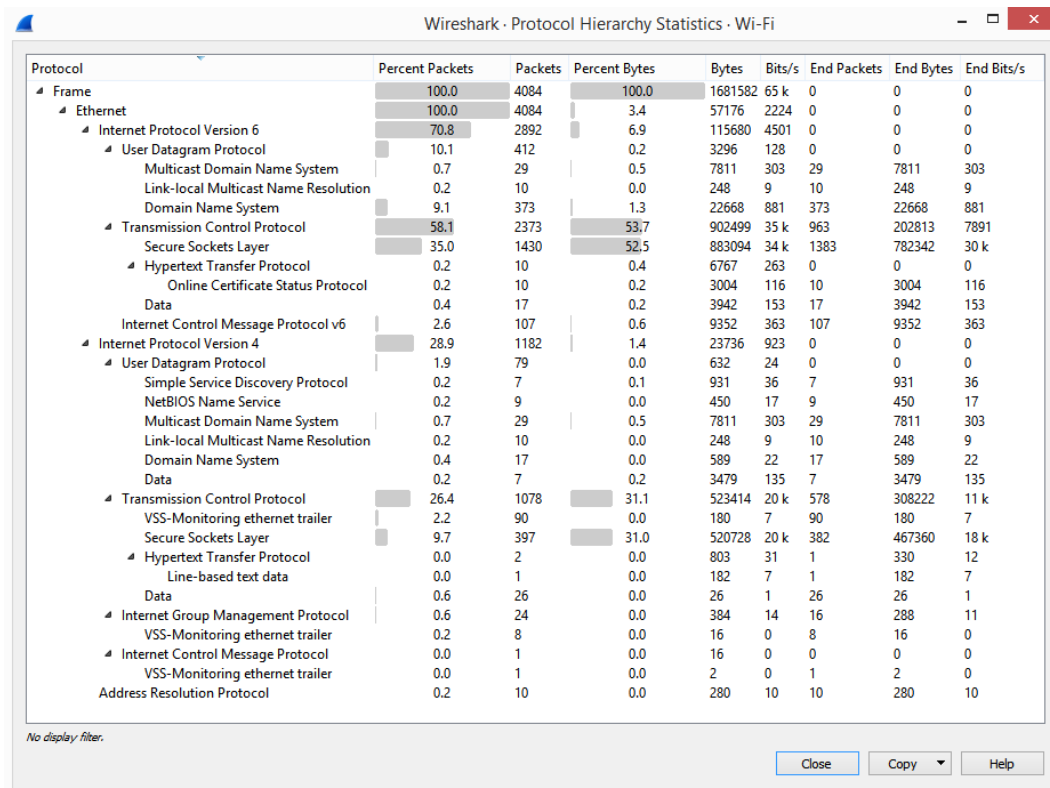


Figure 5.13 – The Protocol Hierarchy Statistics window

For example, after examining the capture, you will see a large amount of Spotify (a streaming music service) traffic, which is prohibited on your network. At that point, you might choose to investigate the source of the traffic.

In addition, you can go to **Statistics** and then **Conversations** to identify what ports are in use. After all of the captures are complete, we will move on to the final phase where we save the captures for later comparison.

## Saving the baselines

Once you have completed and analyzed the capture, and made any appropriate comments, it's time to preserve the baseline. Whether you work on your own or within a team, you should have a standard format and procedure to document the findings.



The suggested guidelines for documentation include the following:

- List where the capture was taken, and include the name of the building and/or subnetwork.
- List the name of the technician who performed the capture. If someone has questions about the capture, they can contact the individual.
- List the date and time of the capture.
- Outline or summarize the overall findings, such as "*normal traffic flow with no unusual or unauthorized protocols in use*".
- List the name of the file and where the baseline is stored.

Although much of the information in the preceding list can be recorded within the capture in the form of comments, it's best to document to preserve the information.

When naming the capture files, have your team agree on a standard format. This is so you can easily search through the captures later. One format or standard might be to use the building name and room or even the subnetwork IP address.

For example, you might use this format – *building-room-subnet* (or *BLD-RM-SN*). Then, if you have a capture from *aviation building – room 78 – subnetwork 192.168.10.112*, you can save the file as `AV-78-10.112.pcap`.

The format for saving the information can be a shared spreadsheet that the team can update and use to record their findings.

## Summary

By now, you understand the many different types of networks that can influence how data travels. In addition to the various network types, we saw how we must also contend with the media that transmits the data. So that you can effectively capture traffic, we took a closer look at various capture selections that include display options, using multiple files, and name resolution.

We then moved into a discussion of the different types of traffic you will see when tapping into a switched network and compared the difference between conversations and endpoints. We then looked at the many options for analysis within the **Conversations** window. Finally, we summarized the importance of baselining the network and provided some steps on how to move through this process.

In the next chapter, we will discover the many ways to personalize the Wireshark interface. You will learn ways to adjust the appearance and basic layout. I'll show you ways to add, modify, and personalize the configuration profiles. Then, we will evaluate how to add comments to a single packet or an entire capture. Finally, we'll take a look at creating a complex filter expression and a button for your toolbar to simplify your analysis.

## Questions

Now, it's time to check your knowledge. Select the best response, and then check your answers with those found in the Assessments appendix:

1. A \_\_\_\_\_ is a private network in a localized area that an organization or individual owns, controls, and manages:
  - A. LAN
  - B. WAN
  - C. CAN
  - D. PAN
2. When using name resolution, it is okay to select **Resolve MAC Addresses** and **Resolve transport names**, as these come from static text files. \_\_\_\_\_ is a list of Ethernet vendor codes and well-known MAC addresses:
  - A. `vendor.txt`
  - B. `services.txt`
  - C. `manuf.txt`
  - D. `network.txt`
3. In the capture options, the \_\_\_\_ tab allows you to specify where and how you want to save your file:
  - A. **Input**
  - B. **Output**
  - C. **Options**
  - D. **Baseline**

4. When using a fiber-optic cable, \_\_\_\_ carries a single light beam that can carry a signal for many miles and must use a laser:
  - A. multimode
  - B. coaxial
  - C. UTP
  - D. single mode
5. In the **Output** tab of the capture options, **Output format** saves the file by default as \_\_\_\_; however, you can force Wireshark to save the file as .pcap:
  - A. .pcapng
  - B. dmp.gz
  - C. cap.gz
  - D. .txt
6. This type of cable consists of twisted pairs of copper wire that use pulses of electricity to carry a signal:
  - A. Multimode
  - B. Coaxial
  - C. UTP
  - D. Single mode
7. In Wireshark, a \_\_\_\_\_ consists of two endpoints that are in a connection together:
  - A. coax
  - B. named pair
  - C. conversation
  - D. statistic
8. If you select \_\_\_\_\_ and then **Capture File Properties**, you can add a comment to provide additional information about a capture:
  - A. **Capture**
  - B. **Edit**
  - C. **View**
  - D. **Statistics**

# 6

# Personalizing the Interface

Everyone likes to arrange their desks in their own way. Working with Wireshark is no different, as you can personalize the settings to suit your needs. In this chapter, we'll dive into the Wireshark interface and look at ways to enhance the appearance and layout, as well as create custom configuration profiles. You'll gain a better appreciation of how you can design the interface to meet your specifications and evaluate ways to manipulate columns, change the font, and fine-tune color choices.

So that you get a better understanding of how to document capture information in Wireshark, we'll review tips on how to add comments. You'll discover how to comment on a single packet or an entire capture. Finally, we'll step through the process of building and modifying a complex filter during packet analysis. We'll finish by learning how to create a button on the toolbar as a shortcut for commonly used filters in Wireshark.

This chapter will cover the following:

- Personalizing the layout
- Creating a tailored configuration profile
- Adjusting columns, font, and color
- Adding comments

## Personalizing the layout

Although Wireshark is functional in default mode, it's easy to modify appearance and layout to optimize your workflow. In addition to personalizing layout and general appearance, you can modify many other components. For example, you can change the language, customize the number of icons, recent filters, and folders, and define what you want to appear in the status bar. Let's start with how you can customize the appearance.

## Altering the appearance

In Wireshark, you can tailor the general appearance in the following ways:

- Identify the default location to open files.
- List how many display filters to show.
- Define how you want the main toolbar to appear.

To view all the choices where you can make and modify your selections, go to **Edit** and then **Preferences**, which will bring up a dialog box. Once there, expand **Appearance**, as shown in the following screenshot:

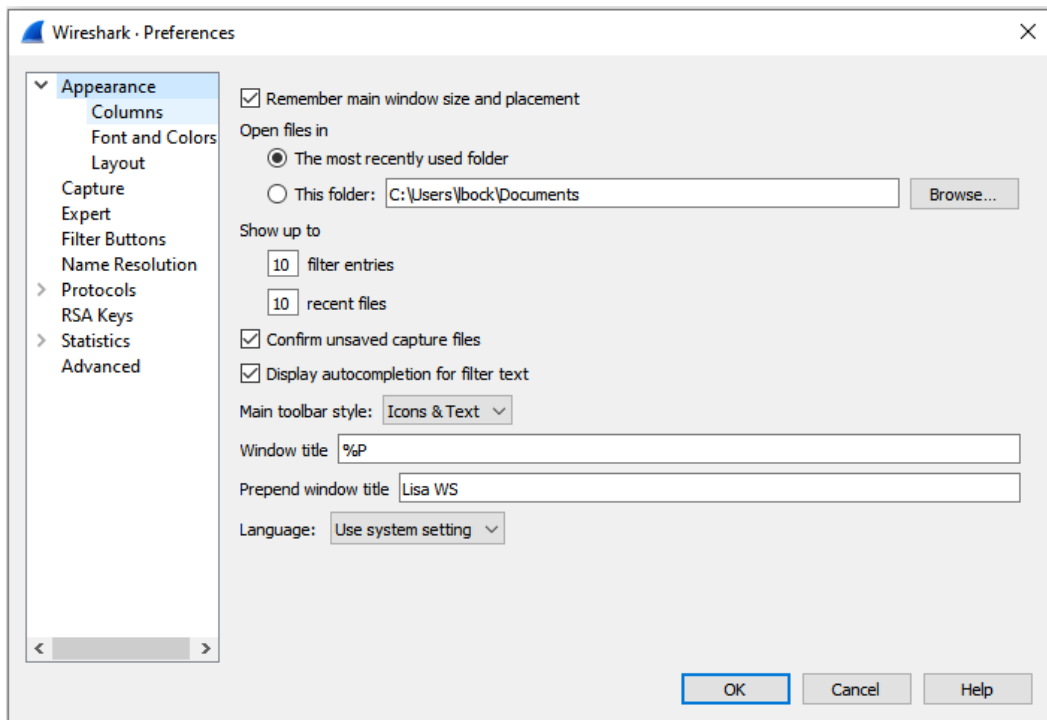


Figure 6.1 – The Wireshark Preferences dialog box – Appearance

At the top of the dialog box, we can see the **Remember main window size and placement** option. If selected, this will retain the window size and placement after you shut down Wireshark. To see how this works, do the following:

1. Reduce the main window to one-quarter size, position the app on the top left-hand side of your screen, and then select **File | Quit**.
2. Launch Wireshark, and the window will appear in the same location and size.

The next selection, **Open files in**, allows you to point to a location when accessing files. Here, you will find two choices:

- **The most recently used folder** is the default. This option will save files in the last folder you used.
- **This folder** will allow you to choose another folder when saving files. To modify, select **Browse...** and then drill down to the location of a specific folder.

**Show up to** allows you to modify how many files or filters are preserved. The choices are outlined as follows:

- **filter entries** will indicate how many *capture filter* entries to display.
- **recent files** will indicate how many *recent files* to keep visible when you go to **File | Open Recent**.

When working with captures, Wireshark can help remind you to save your file by selecting **Confirm unsaved capture files**. If checked, Wireshark will display a prompt before closing the app, starting a new capture, or opening another file, as shown here:

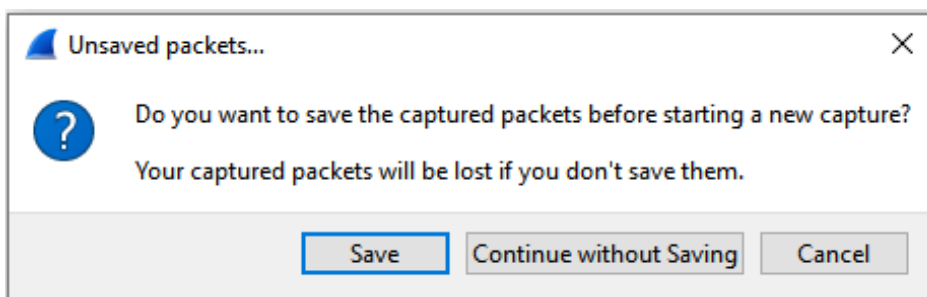


Figure 6.2 – A prompt before closing

The next option, **Main toolbar style**, will allow you to alter the appearance of the main toolbar, which is across the top, underneath the menu choices. You can choose to display as **Icons only**, **Text only**, as well as **Icons & Text**, as shown here:

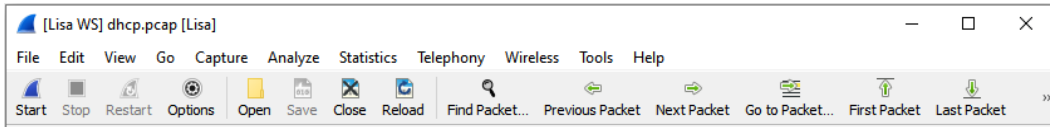


Figure 6.3 – Main toolbar – Icons and text

The next selections will enable you to modify how the title across the top of the screen appears. The choices include the following:

- **Window title:** This option will allow you to append a label at the end of the title and includes options such as %F (file path) and %P (profile).
- **Prepend window title:** This option will allow you to insert a label at the beginning of the title.

I have included both options, so the title appears as [Lisa WS] dhcp.pcap [Lisa], as shown in *Figure 6.3*.

**Note**

For **Window title**, I used the %P (profile) selection.

The last option is **Language**. When using Wireshark, the default is **Use system setting** when displaying capture information. However, the developers have added a powerful feature – the ability to select from a variety of languages, including **Chinese**, **English**, **French**, **German**, **Italian**, **Japanese**, and **Polish**.

Now that we have set up our workspace, let's evaluate ways to modify the layout.

## Changing the layout

As covered in the *Exploring the View menu* section in *Chapter 4, Exploring the Wireshark Interface*, you can select which of the panels you want to appear, as follows:

- **Packet List**
- **Packet Details**
- **Packet Bytes**
- **Packet Diagram**

Once you launch Wireshark, the default setting is the three panes – **Packet List**, **Packet Details**, and **Packet Bytes** stacked one above the other.

However, it's easy to rearrange the layout in one of six other layouts. To make changes to the layout, go to **Edit | Preferences**, which will bring up a dialog box. Once there, expand **Appearance**, and then select **Layout**, as shown here:

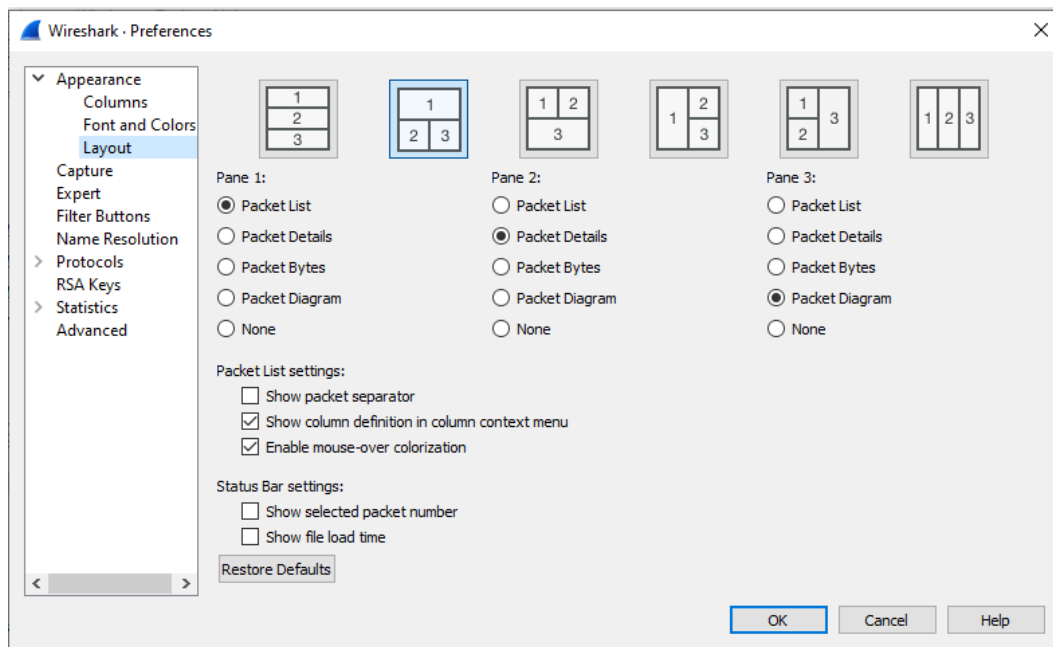


Figure 6.4 – The Wireshark Preferences dialog box – Layout



For example, if you have selected the settings shown in the preceding screenshot, your layout will appear as follows:

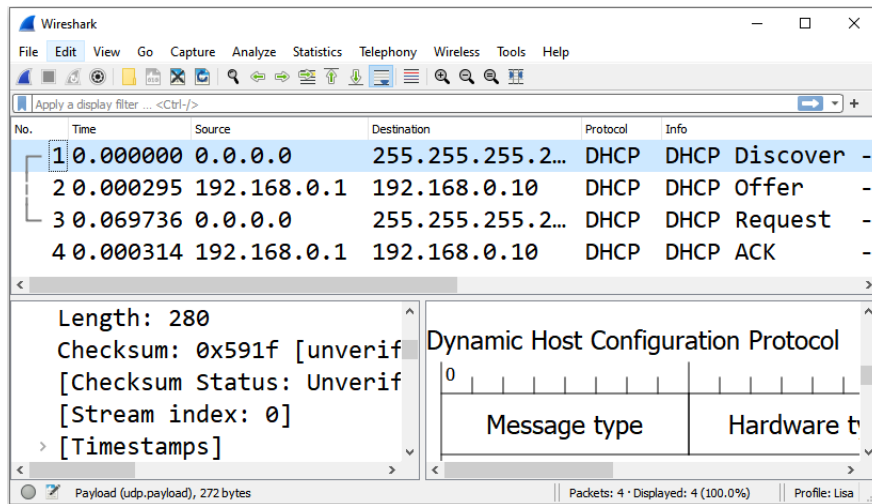


Figure 6.5 – Wireshark with a modified layout

In addition, you have choices to modify additional selections, for either the **packet list** or **status bar** settings. Let's start with how we can enhance the view of the **packet list** settings.

## Adjusting the packet list settings

Within this section, Wireshark provides additional options to visualize the packet list.

The **Show packet separator** option will insert a fine white line in between each frame in the packet list.

If you select **Show column definition in column context menu**, when you right-click on any of the column headers, Wireshark will describe each of the column values, as shown here:

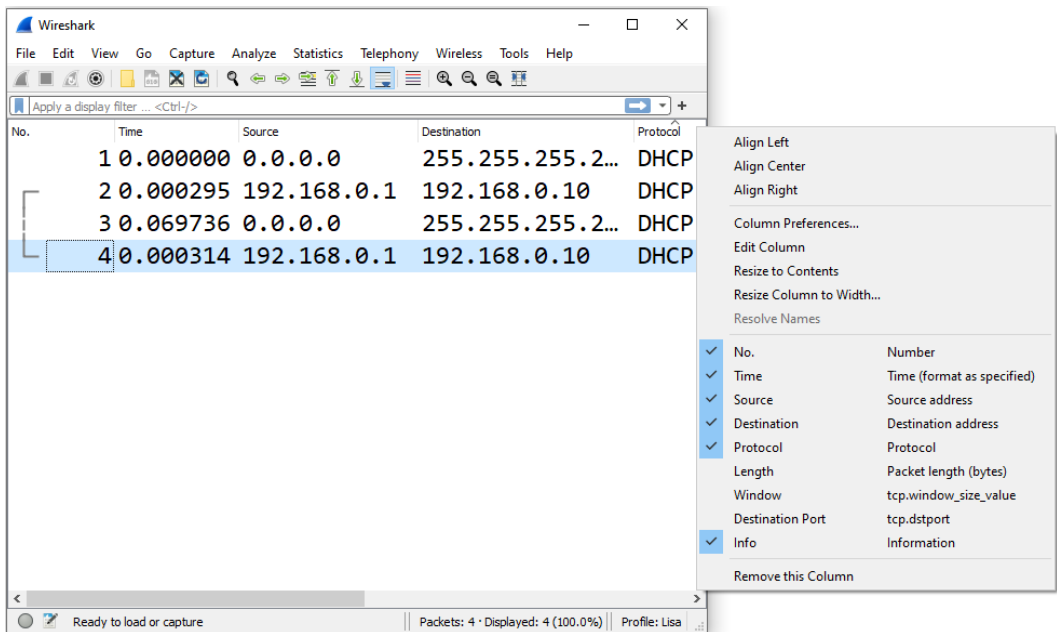


Figure 6.6 – Show column definition in column context menu

When you select **Enable mouse-over colorization**, this will highlight the selected packet in a light blue color, as shown in frame 4 in the preceding screenshot.

In addition, we can modify the status bar, which is found at the bottom of the Wireshark interface.

## Adding status bar settings

Here, you can choose to select **Show selected packet number** and **Show file load time**.

Once selected, this information will be displayed in the lower-right-hand corner of the status bar, as shown here:

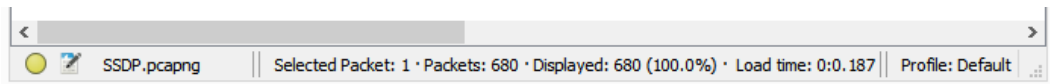


Figure 6.7 – The status bar

The last option, **Restore Defaults**, will reset any options that you have selected to the default state.

Next, we'll examine ways to generate a custom configuration profile to provide a unique set of settings that are specific to our needs.

## Creating a tailored configuration profile

A configuration profile is a set of preferences and configurations. Once you launch Wireshark, at the lower right-hand corner of the interface, you will see **Profile: Default**, as shown in *Figure 6.7*.

In Wireshark, users can create their own custom configuration profiles, which can include personalized preferences, coloring rules, font styles, and buttons.

To create a custom profile, go to **Edit | Configuration Profiles**. Once the dialog box is open, you will see that Wireshark has a default configuration called **Default**. In addition, you will see three **Global** configurations, **Bluetooth**, **Classic**, and **No Reassembly**, as shown in italics in the following screenshot:

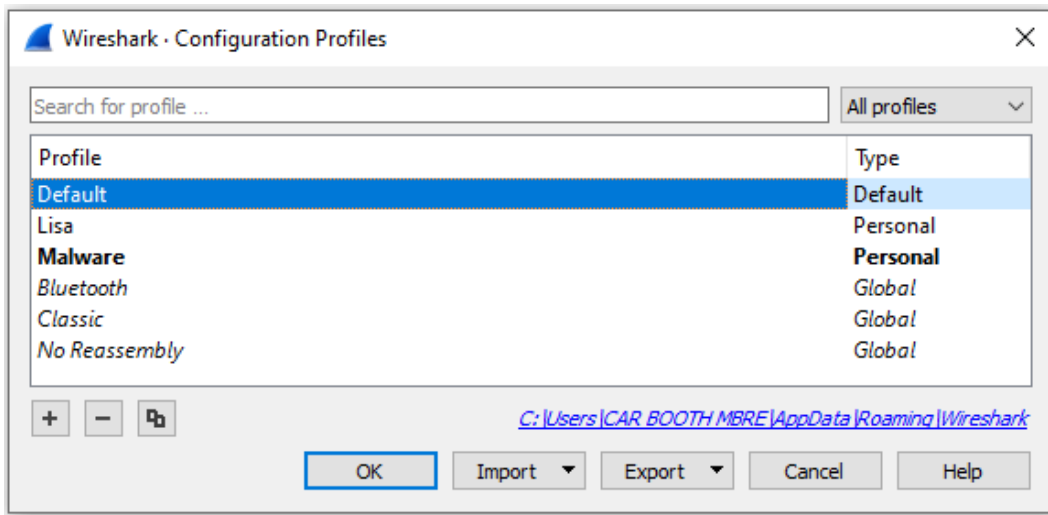


Figure 6.8 – The Configuration Profiles dialog box

In addition, I have created two personal profiles, *Lisa* and *Malware*. In the next section, we'll see how easy it is to create a new profile.

## Customizing a profile

Once you have determined that it will be beneficial to create a custom profile, you can begin the process by going to **Edit | Configuration Profiles**. Once there, select the + sign and assign the profile a name. For example, I created a profile named *Malware*, as shown in *Figure 6.8*. Once you add the profile, close the dialog box, and then you can modify the profile.

You can make several changes to suit your needs, such as the following:

- Modify the layout by going to **View** and unchecking **Packet Bytes**.
- Go to **Edit | Preferences** and make changes such as font color and size, or even disable or change some of the column headers.

Once done, Wireshark will save any changes in the custom profile.

In my **Malware** profile, I wanted to modify the settings so that I can hunt for an Ettercap signature. **Ettercap** is a tool that is used to launch man-in-the-middle attacks on a LAN. I want to be able to quickly identify the `e77e` Ettercap signature, which identifies Ettercap as it searches for other poisoners on a LAN.

#### Note

The Ettercap signature `e77e` translates to **ette** (short for **Ettercap**) in leetspeak. You can check this by visiting <https://www.dcode.fr/leet-speak-1337>.

To customize my profile, I adjusted the columns and removed the **Packet Bytes** lower panel. Finally, I added an **ette** button (which we'll learn how to create in the *Crafting buttons* section). Once selected, the button will apply and run the `icmp.ident == 0xe77e` display filter.

The results are shown in the top right-hand corner of the following screenshot:

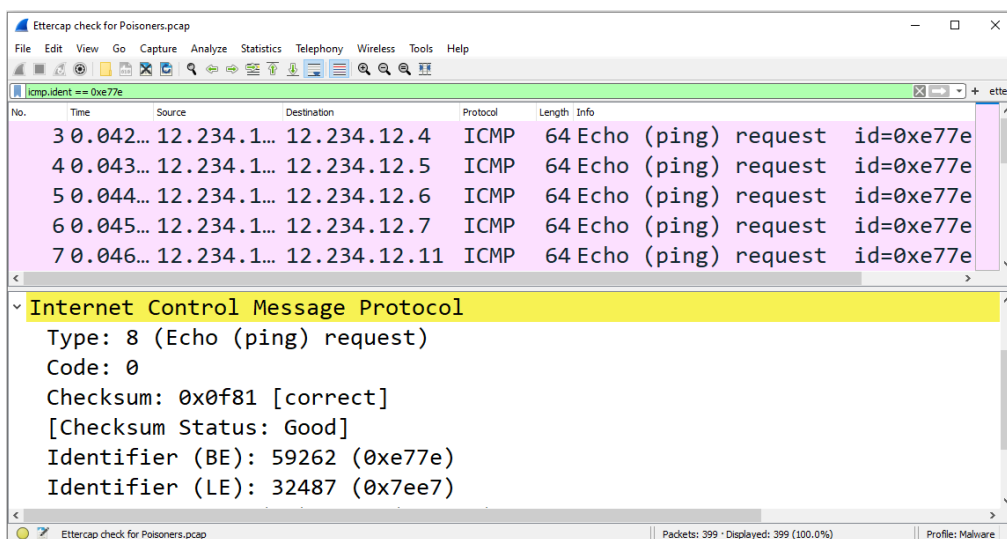


Figure 6.9 – The Malware profile

Using my Malware profile, I can easily check for Ettercap poisoners by hitting my button, which will then show any packets in the capture that have that signature.

If you want to *change* the profile, *click* on the lower right-hand corner and select the profile you want to use, as shown in the following screenshot:

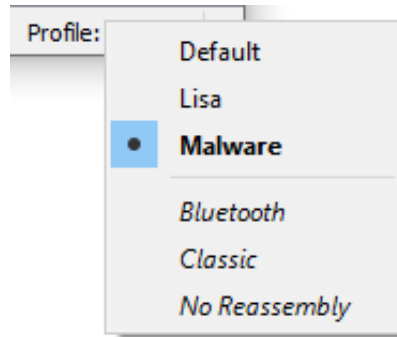


Figure 6.10 – Modifying the profile

In addition, if you want to *manage* the profiles, *right-click* on the **Profile** label. Wireshark will bring up a dialog box, where you can then select **Manage Profiles...**, as shown in the following screenshot:

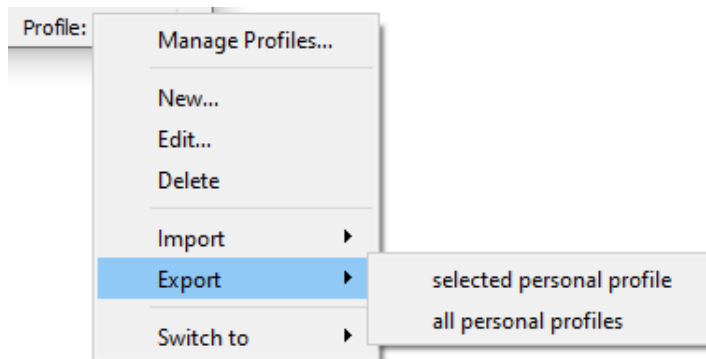


Figure 6.11 – Manage Profiles

Along with being able to manage and modify the profiles, Wireshark allows you to import and export profiles, so you can easily share them with your team. Simply right-click on either choice to make your selection.

You can import profile(s) from the following:

- **from zip file**
- **from directory**

You can export the following profile(s):

- **selected personal profile**
- **all personal profiles**

Many times while conducting packet analysis, there is a need to refine our view by filtering traffic. If you have created a filter that you find you will need to run often, then you can create a handy toolbar button, as outlined in the next section.

## Crafting buttons

A display filter allows you to show only specific traffic. Many times, we'll use a simple filter, such as a `dns` (Domain Name System) or `http` (Hypertext Transfer Protocol). However, there are times where you may need to create a more complex filter, such as when comparing a specific protocol field against a value using logical operators.

While working with Wireshark, you may have created a complex filter by using shortcuts.

### Note

We'll learn more about shortcuts in the *Discovering shortcuts and handy filters* section in *Chapter 7, Using Display and Capture Filters*.

For example, you created a display filter to show only the **Transmission Control Protocol Synchronization** (TCP SYN) flag and *not* the **Acknowledgement** (ACK) flag:

```
(tcp.flags.syn == 1) && !(tcp.flags.ack == 1)
```

Once you have created an expression and it is visible in the display filter, you can effortlessly create a button. In the right-hand corner, right after the **Display Filter**, hit the + (plus) sign, and Wireshark will display a drop-down dialog box, as shown in the following screenshot:

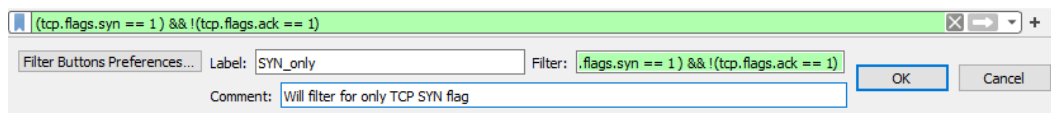


Figure 6.12 – Creating a filter button

Once the dialog box is open, you can enter an appropriate label and add a comment for the filter button. Wireshark will automatically enter the display filter for the button to run when clicked. When done, select **OK**, and the new button will appear on the toolbar.

After working with Wireshark and adding buttons, you may want to remove some to clean up the toolbar. The following list shows the steps to edit filter buttons:

1. Go to **Edit** and then **Preferences**, where you will see **Filter Buttons**, as shown in the following screenshot:

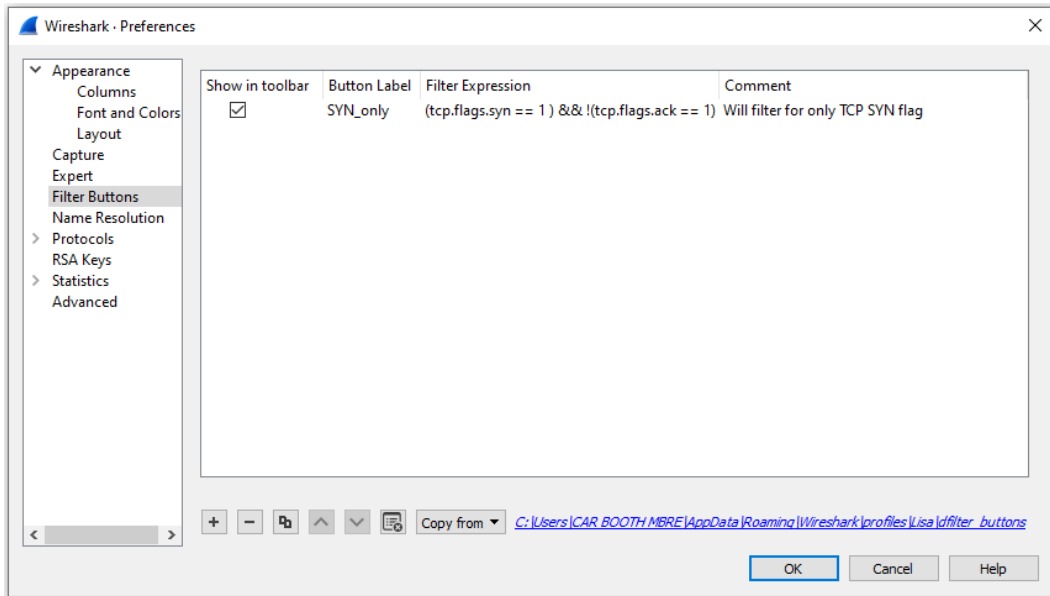


Figure 6.13 – The Wireshark Preferences dialog box – Filter Buttons

When selected, you will see whatever buttons are currently on your toolbar.

2. To add a filter button, select the + (plus) sign in the lower left-hand corner. Add what you want for **Button Label**, which is not case-sensitive.
3. Next, add a **Filter Expression** option, which must adhere to the expression and display filter rules.

If you want to remove any filter buttons, highlight the button you want to remove and select the – (minus) sign in the lower left-hand corner.

You can now see the power of creating a customized profile to personalize your workspace. Now, let's investigate how you can add or remove columns, and adjust font and color to suit your needs.

## Adjusting columns, font, and colors

While working with a packet capture, most users are comfortable with the default settings used in the interface. However, you can adjust font styles and size to personalize the look and feel of your workspace. In addition, you can also modify the colors that Wireshark uses for the various packet identifiers and display filters.

Once you are in the interface, you will see the column headers that are along the top of the screen. While you are working on a capture, you might not ever manipulate the columns. However, you can add, delete, align, and customize the columns at any time.

Wireshark makes it easy to add and modify columns, as we'll see in the next section.

## Adding, editing, and deleting columns

In Wireshark, you can do more than simply expand or shrink the column headers while in the interface.

To improve how you visualize columns, go to **Edit | Preferences**, and then **Columns**, as shown in the following screenshot:

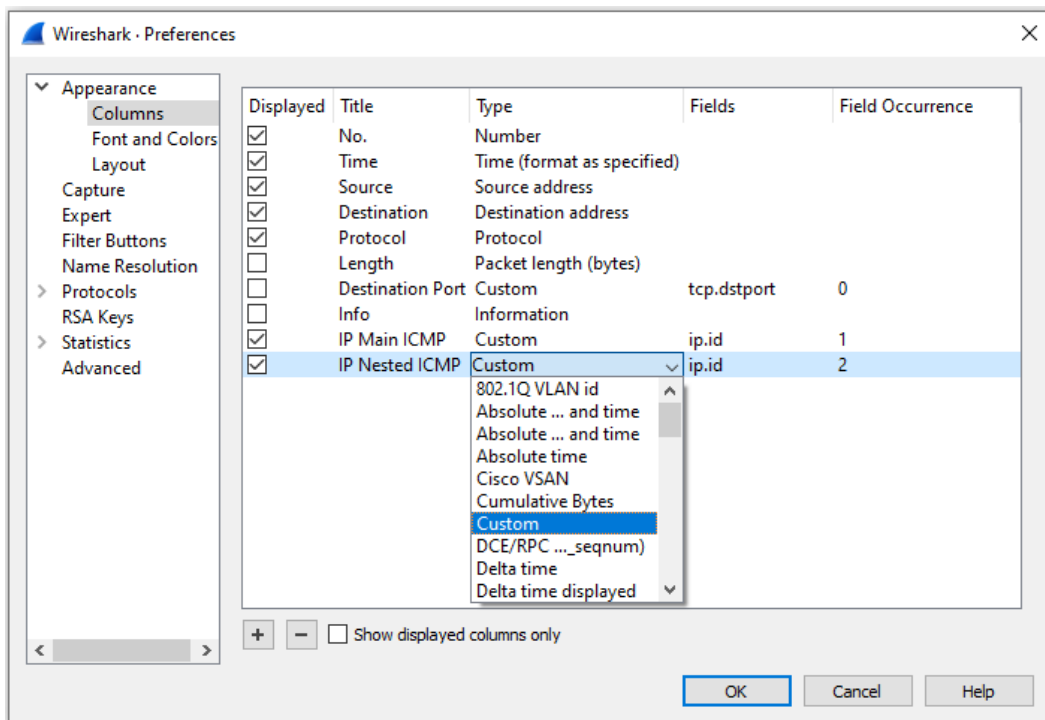


Figure 6.14 – The Wireshark Preferences dialog box – Columns



Once selected, you will see a list of columns. Some are present by default, and others are ones you may have added. You can select the checkbox to make the column visible, as well as add or remove columns.

Along the top of the dialog box, you will see the following selections:

- **Displayed:** When checked, the selected column will be displayed on the interface.
- **Title:** This is the name of the column header. Wireshark will automatically create a name if you right-click and add a column. However, you can change the title name to personalize the header.
- **Type:** This lists the type of value that is in the column. Within the drop-down menu, there are many pre-loaded choices, as shown in *Figure 6.14*, where I have dropped down the **Type** selection for the **IP Nested ICMP** header.
- **Fields:** This identifies the field where the column value originated from. In the preceding screenshot, there is a column header called **IP Nested ICMP**. In the **Fields** column, we can see **ip.id (Internet Protocol Identification)**. That is because the column header was generated by right-clicking on the **Identification (ID )** field in an IP header and selecting **Apply as Column**. Wireshark populated the value as `ip.id` and created a custom column type.
- **Field Occurrence:** This is only used on a custom column definition. In *Figure 6.14*, you can see that there are values of **1** and **2** in the **Field Occurrence** column. When selected, the column headers will appear in this order:
  - A. **IP Main ICMP** will appear first.
  - B. **IP Nested ICMP** will appear second.

To add a column, select the + (plus) sign. Identify the title by typing in an appropriate label where it says **New Column**, and then identify the type by using the drop-down menu and selecting a type. You can also remove any unwanted columns by highlighting the column and hitting the - (minus) sign.

In addition, once in the interface, you can align the columns by right-clicking and selecting the way you want your columns to align, either left, center, or right, as shown here:



Figure 6.15 – Aligning columns

Most other column headers are fairly straightforward in how they are used. However, the one you may not be familiar with or use very often is **Field Occurrence**. Let's explore this concept next.

## Using a field occurrence

When an **Internet Control Message Protocol (ICMP)** error message is sent, the ICMP packet will include the following:

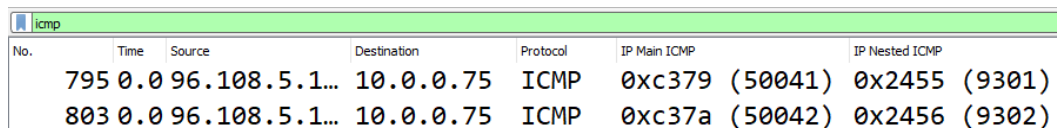
- The IP header for the error message.
- The first 8 bytes (64 bits) of the original datagram that caused the error. This is sometimes referred to as a nested ICMP packet.

To see an example of an error, go to CloudShark to obtain a copy of `ICMPv4_Destination_unreachable`, found at <https://www.cloudshark.org/captures/155db9732c91>, and then open the capture in Wireshark.

### Note

We'll learn more about ICMP in *Chapter 12, Discovering ICMP*.

When using **Field Occurrence**, we can see the ID field of the first (main) IP header along with the ID field of the nested IP header. This will allow us to easily see the difference between the two, as shown in the following screenshot:



No.	Time	Source	Destination	Protocol	IP Main ICMP	IP Nested ICMP
795	0.0	96.108.5.1...	10.0.0.75	ICMP	0xc379 (50041)	0x2455 (9301)
803	0.0	96.108.5.1...	10.0.0.75	ICMP	0xc37a (50042)	0x2456 (9302)

Figure 6.16 — Using the field occurrence option

To create the custom column headers, let's start by creating a new profile.

Go to **Edit | Configuration Profiles**:

1. Once there, select the + sign and name the new profile `ICMP`.
2. Close the dialog box.
3. Make any desired profile modifications, such as changing the font or modifying the panel layout.

Now that we have a new profile, we will create the field occurrence option.

Go to **Edit | Preferences**, and then **Columns**. Uncheck **Length** and **Info**, as shown in the following screenshot:

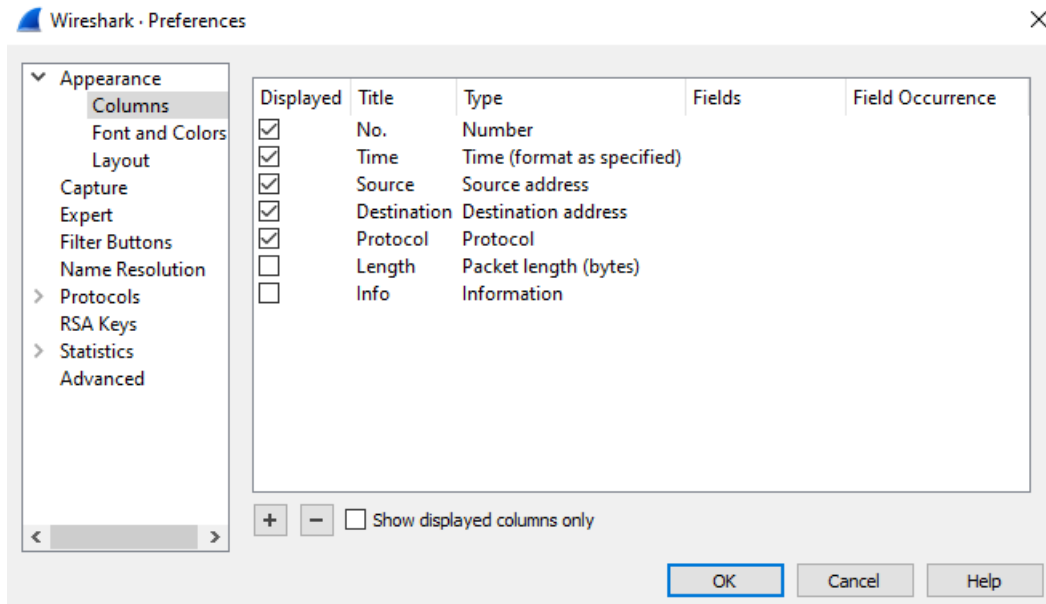


Figure 6.17 — Viewing the Columns dialog box in Preferences

Next, we will add two new column headers.

Select the + (plus) sign and then modify the settings for the first newly created column header as follows:

- **Displayed:** Checked
- **Title:** IP Main ICMP
- **Type:** Custom
- **Fields:** ip.id
- **Field Occurrence:** 1

Select the + (plus) sign and then modify the settings for the second newly created column header as follows:

- **Displayed:** Checked
- **Title:** IP Nested ICMP
- **Type:** Custom

- **Fields:** `ip.id`
- **Field Occurrence:** 2

The result is shown in *Figure 6.16*, where, right after the **Protocol** column header, you will see **IP Main ICMP**, followed by **IP Nested ICMP**.

As we can see, Wireshark is very versatile in modifying columns to adjust your view. Next, let's take a look at an overlooked feature in Wireshark, which is the ability to adjust the font and change the default colors.

## Refining the font and colors

In the main window, you may feel the text is too small, as the **Default** profile uses the Consolas 10 style as the font and size. It's easy to change the font and colors, make the text bold or italic, and change the font size and style.

Go to **Edit | Preferences** and select **Font and Colors**, as shown in the following screenshot:

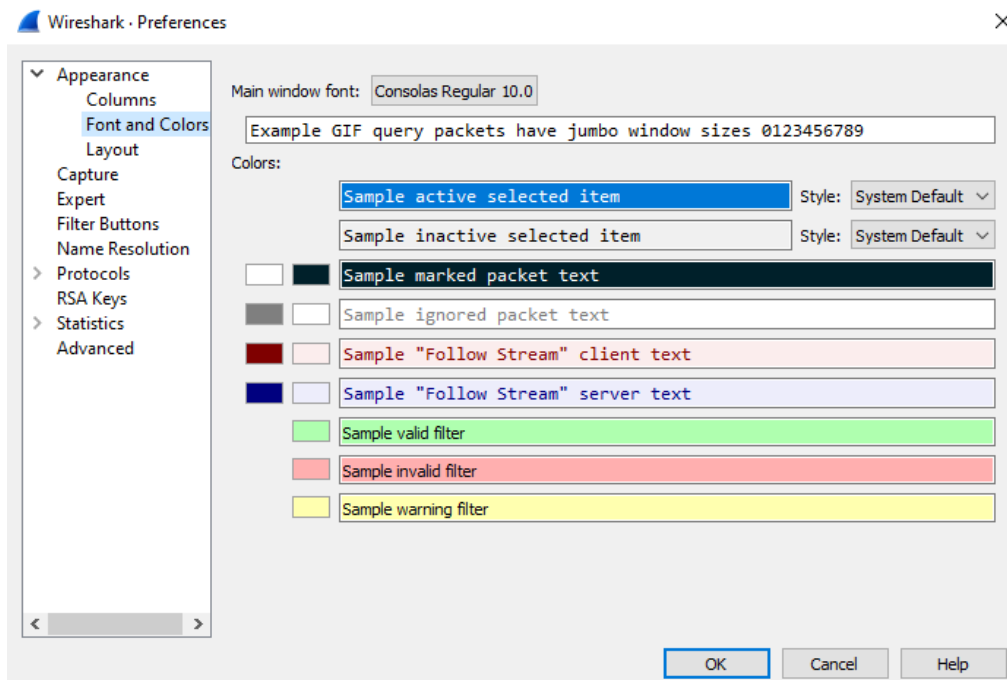


Figure 6.18 – The Wireshark Preferences dialog box – Font and Colors

Along the top, if you select **Main window font**, Wireshark will display a dialog box that will allow you to make changes to the font, as shown here:

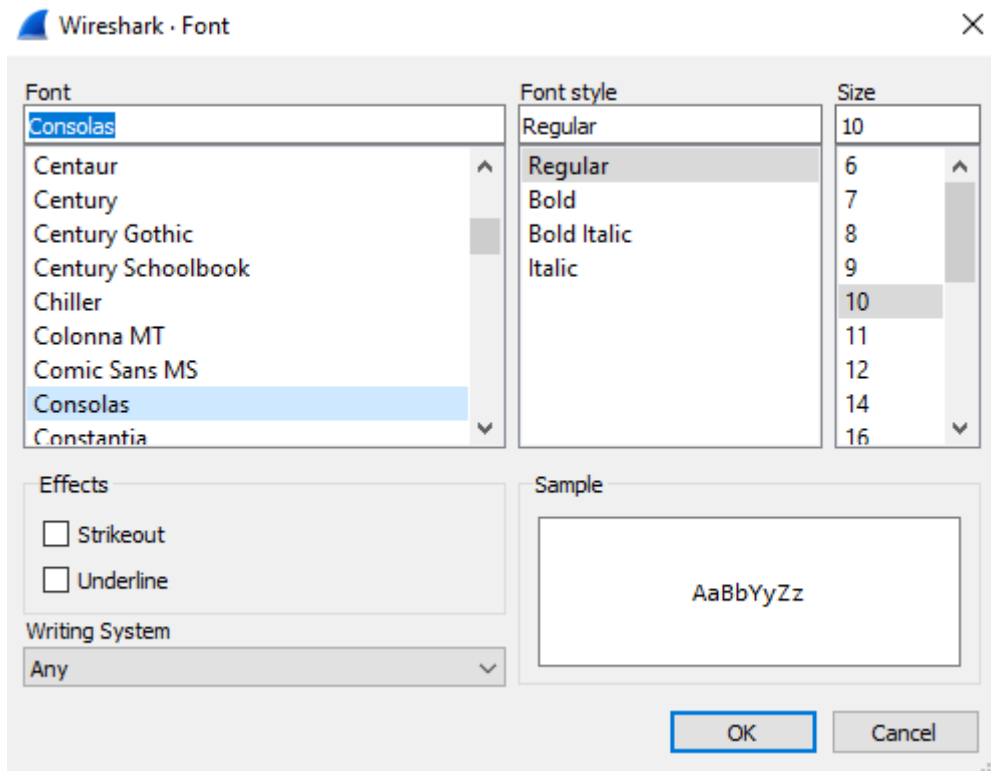


Figure 6.19 – Main window font

Below **Main window font**, you'll see defaults for the way Wireshark colorizes the text and background for active and inactive items, along with marked and ignored packets. After that, you will see the defaults for the client and server text when you right-click on a packet and select **Follow the TCP Stream**, as shown here:

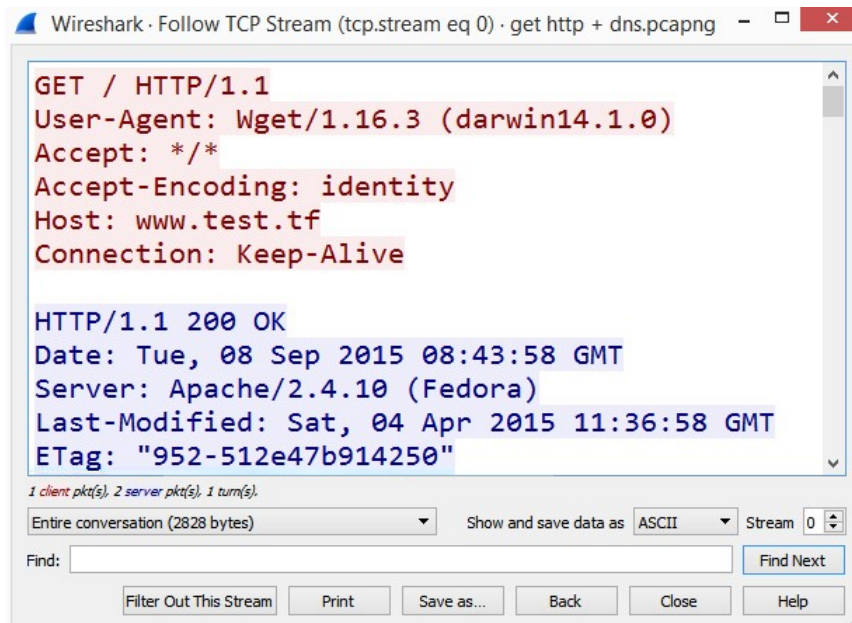


Figure 6.20 – Follow the TCP Stream

The colors help identify whether the client or the server is talking. When you follow the TCP stream, the default value for the client text is red, but this value can be changed.

The last three samples listed refer to the colors when creating a filter. The syntax checker within Wireshark checks the filter as you are entering text. When creating a filter, the following are default values:

- A valid filter turns the background green.
- An invalid filter turns the background red.
- A warning filter turns the background yellow.

As with all the other choices, you can change these values as well.

In Wireshark, we can indicate information about a particular packet or capture a file in the form of comments, as we'll see in the next section.

## Adding comments

While conducting packet analysis, there may be issues that you will want to highlight and identify so that you can reference them at a later date. For example, you might want to make a note on a single packet or an entire capture for future reference.

All of this is possible in Wireshark, as you can write a note in the capture outlining the key issues that were found. Once documented, you or your team can reference the comments at a later date.

### Note

While commenting is optional, it is always good practice to document to help preserve the details of your findings.

Let's start with how we can add file comments.

## Attaching comments to files

Adding a comment to a packet capture is a very handy tool. When adding a comment, you can view it later to refresh your memory on key issues related to that packet capture. For example, you may have identified possible illegal or malicious activity, such as cryptocurrency mining, and you can list the details right in the capture file.

There are a few ways to add a comment to the file. Some examples are as follows:

- Go to the status bar and select the icon that looks like a pencil and paper, which is found to the immediate right of the expert system icon, as shown on the left-hand side of *Figure 6.7*.
- Go to **Statistics** and then **Capture File Properties**, where you can add comments in the lower pane of the dialog box. Once done, you should select **Save Comment**.

Adding a comment to the file can help remind you or your team of what was significant about the capture. However, it's also possible to add a comment to a single packet, as discussed in the next section.

## Entering packet comments

To add a comment to a single packet, select the packet and then go to **Edit | Packet Comments | Add New Comment**. A dialog box will be displayed, where you can enter your comment. Once you have entered your comment and saved the file, Wireshark will append a note with bright green coloring on the top of the frame, in the **Packet Details** panel, as shown in the following screenshot:

```

- Packet comments
  - NTP Version 3
  - Frame 1: 90 bytes on wire (720 bits),

```

Figure 6.21 – Packet comments in the Packet Details panel

Once you have added comments to either the entire capture or a single packet, you'll want to save them and then, at some point, go back in and reference the notes. The following section reviews how we can take a look at the comments and how to preserve our remarks.

## Viewing and saving comments

Comments can be a powerful tool, as you can use them in many ways to preserve what you felt was significant in the capture. Even after several years have passed, I still find the comments valuable, as they help me to remember my train of thought when I analyzed a packet capture. However, in order for the comments to be of value, you must save them. Let's discuss how this is achieved.

Once you have created a comment, you will see an asterisk by the title across the top of the Wireshark interface, as shown here:

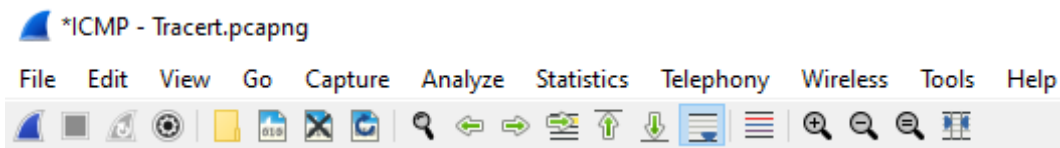


Figure 6.22 — An asterisk indicating that there are file comments

The asterisk will remain until you save the file. Keep in mind that to preserve the comments, you must save the file in PCAP Next Generation ( .pcapng) format.

If you or your team has taken the effort to make a comment, then it's well worth your time to read them. You can view either packet or capture file comments in a few ways:

- To see all packets that have comments, use the `pkt_comment` display filter.
- To see all the comments for the entire capture, go to **Statistics | Capture File Properties** and view the comments found in the lower pane.



You can also view packet comments by going to **Expert Information**, which is found by selecting the colored circle in the lower left-hand corner of the interface. Once the dialog box opens, expand the **Comment** section to see any comments, as shown in the following screenshot:

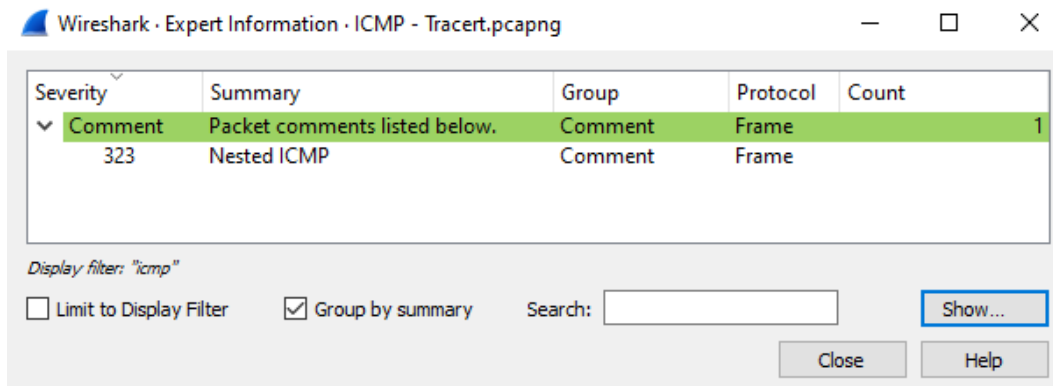


Figure 6.23 – A packet comment in the Expert Information panel

By now, you can appreciate the many ways in which you can personalize the interface and document your work while analyzing packet captures.

## Summary

Whether you are a power user or a casual analyst, it only takes a few minor tweaks to personalize Wireshark so that you can complete tasks more efficiently. In this chapter, we examined the many ways to customize the Wireshark interface to fit your workflow. We covered how to modify choices such as recent filters and folders, along with the layout and general appearance. We discovered how easy it is to create tailored profiles to include preferences, coloring rules, and font styles. We then learned how to create a filter button on the toolbar for commonly used filters in Wireshark.

Furthermore, we discovered how to modify, add, or remove columns and column headers. We examined ways to fine-tune the font to make packets easier to read. We also reviewed how we can change the default colors for the various identifiers, such as the text color for marked packets. Finally, we illustrated the ability to add comments to a single packet or the entire capture to communicate issues to team members.

In the next chapter, we will take a closer look at using display and capture filters, as well as learn about some tricks and specific rules for using filters. We will then learn about using capture filters, including default capture filters, and how you can build your own. Finally, we will discover how to use shortcuts to create filters and review some commonly used filters to better manage your workflow.

## Questions

Now, it's time to check your knowledge. Select the best response and then check your answers, which can be found in the *Assessment* appendix:

1. Normally, when you open Wireshark, the configuration profile will be set as the \_\_\_\_\_ profile:
  - A. Marquee
  - B. Bluetooth
  - C. Classic
  - D. Default
2. You can set Wireshark to open files from a specific location. Go to \_\_\_\_\_, then **Preferences**, and then select the file location under \_\_\_\_\_:
  - A. **Tools | Folder**
  - B. **Edit | Appearance**
  - C. **View | Appearance**
  - D. **View | Folder**
3. The default value for the client text for **Follow the TCP Stream** is \_\_\_\_\_, but this value can be changed:
  - A. black
  - B. blue
  - C. red
  - D. cyan

4. When working with Wireshark, you can easily create and add a button to automatically run a custom filter when selected. To remove a button, go to \_\_\_\_\_, then \_\_\_\_\_, and select **Filter Buttons**, and remove any buttons that you no longer need:
  - A. **Edit | Preferences**
  - B. **View | Appearance**
  - C. **View | Buttons**
  - D. **Tools | Buttons**
5. When you want to add a comment to a packet, select the packet, go to \_\_\_\_, and then go to **Packet Comments**:
  - A. **Tools**
  - B. **Edit**
  - C. **View**
  - D. **Analyze**
6. A configuration \_\_\_\_\_ is a set of preferences and configurations specific to your needs:
  - A. button
  - B. profile
  - C. tool
  - D. analyze
7. When creating a filter, the default value for an invalid filter is to turn the filter background \_\_\_\_\_:
  - A. green
  - B. blue
  - C. red
  - D. yellow

# 7

## Using Display and Capture Filters

Whether analyzing data in real time while capturing traffic or investigating a pre-captured file, you're generally faced with a huge amount of data. How do you make sense of all this information? Most likely, you will benefit from filtering the traffic to narrow the scope, so that Wireshark only displays the traffic that you want to see. This chapter reviews the many methods you can use in Wireshark to filter traffic.

When working with data, it appears as if the capture and display filters are the same. However, while either one can filter traffic, each has its own syntax that must be used when creating a filter. To help you better understand the different ways to refine your view, we'll cover when to filter traffic and outline the difference between display and capture filters. So that you can refine your skills when zeroing in on specific traffic, we'll review ways to drill down to a specific field value by building an expression. In addition, we'll learn how to create more complex filters using logical operators. Finally, because filters are so helpful, we'll cover some tricks, shortcuts, and common filters that will help you achieve a more effective analysis.

This chapter will cover the following:

- Filtering network traffic
- Comprehending display filters
- Creating capture filters
- Understanding the expression builder
- Discovering shortcuts and handy filters

## Filtering network traffic

Wireshark, along with many other packet analysis tools, can take a large capture, filter specific types of traffic, and refine your view to help with analysis. Within Wireshark there are several options you can use to filter traffic:

- **Display filters:** Used during an active capture or on a pre-captured file
- **Capture filters:** Applied prior to capture to only display a certain type of traffic
- **Expressions:** Creates complex filters using logical operators
- **Shortcuts:** Builds a filter on the fly while analyzing packets

Wireshark has capture and display filters that can be used to refine your view. Each filter is applied during a specific time when analyzing traffic. In the next section, let's explore when the best time is to apply a filter.

## Analyzing traffic

While examining traffic, there are four main phases of packet analysis, as discussed in Chapter 2, *Using Wireshark*. The phases are **Gather**, **Decode**, **Analyze**, and **Display**, as shown in the following diagram:

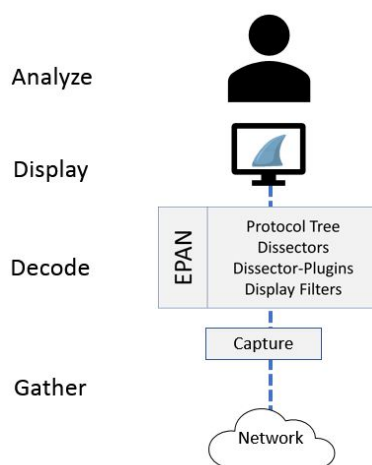


Figure 7.1 – Phases of packet analysis

Refining your view can begin as you are gathering traffic by using a capture filter. The filter will remove unwanted traffic before the packets are presented to the user.

## Capturing traffic

While gathering network traffic, the packets pass through the appropriate capture engine, such as Npcap or WinPcap. If you want to see only specific traffic, you will need to use a capture filter, and when used, Wireshark will drop any packets that are not in the filter. For example, when capturing **Wireless Local Area Network (WLAN)** traffic, and you want to filter out any beacon frames, you will apply this capture filter: `wlan[0] != 0x80`.

Capture filters will remove any unnecessary traffic. However, to filter out unwanted packets while preserving all the traffic, you should use a display filter.

## Viewing traffic

Once the packets go through Wireshark's **Enhanced Packet Analyzer (EPAN)**, it then passes the dissected traffic through the **Graphical User Interface (GUI)** to be presented to the user and then analyzed. While displaying packets in the Wireshark interface, you can apply a display filter using the appropriate syntax, and apply the filter before, during, or after capture.

Similar to our previous example, when capturing wireless traffic, if you want to filter out any beacon frames while preserving all packets you will apply this display filter: `!(wlan.fc.type_subtype == 8)`.

While building either a capture or a display filter, Wireshark will check the syntax to see whether the string is valid. Let's see how this works.

## Checking syntax

When creating filters while analyzing traffic, we inherently know what to build, and in most cases, we will generate a valid filter. Wireshark helps us create an appropriate filter by checking the syntax prior to running a capture. The syntax checker works as follows:

- A valid display filter will turn the background green and the filter *will* run.
- An invalid or incomplete string will turn the background red and the filter *will not* run.
- An unknown display filter or string will turn the background yellow and the filter *might* run.

While it is common to see a green or red background, in rare cases, you may see a yellow background. For example, while filtering for **Dynamic Host Configuration Protocol (DHCP)**, if you enter `bootp` instead of `dhcp` the background will turn yellow, as shown here:

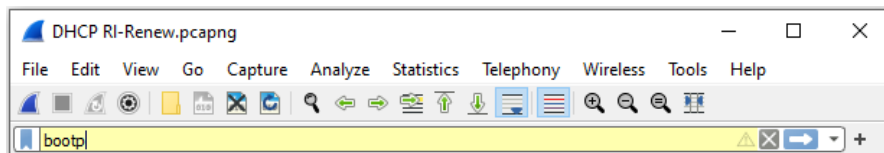


Figure 7.2 – Syntax checker with a yellow background

While the filter might run, you may get unexpected results.

In the next section, we'll outline the purpose of two files, `dfilters.txt` and `cfilters.txt`, in the Wireshark folder, which are used to hold a list of filters.

## Comparing the filters' files

Within the Wireshark folder, there are two text files used for Wireshark's built-in and any user-saved filters, and these are as follows:

- `dfilters.txt` holds a list of display filters.
- `cfilters.txt` holds a list of capture filters.

The files relate to the **bookmark** icon on the hard left of either the display or capture filter bar. As shown here, we see the blue icon on the display filter bar:

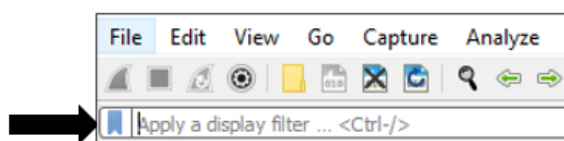


Figure 7.3 – Viewing the bookmark icon

Working with either word filter file, once modified they can be copied and then shared with a coworker. In addition, you can customize either file when using a specific profile, so that you can access the customized filters by simply switching profiles.

First, we'll take a look at the display filters file.

## Investigating display filters

You'll find a list that holds the display filters in the `dfilters.txt` file. Once you locate the file, you can open it in Notepad, as shown here:

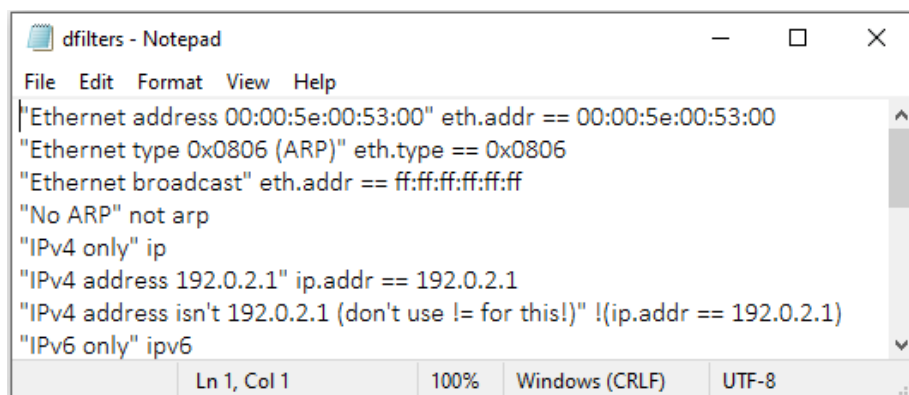


Figure 7.4 – dfilters.txt

The filters are represented by a **Name | Filter** pairing. For example, when examining the filter that excludes **Address Resolution Protocol (ARP)** traffic, you will see the following:

- **Name:** The name of the filter, surrounded by quotation marks. For example, "No ARP".
- **Filter:** The syntax for the filter. For example, `not arp`.

Next, we'll view how the capture filters are stored.



## Examining capture filters

The `cfilters.txt` file holds a list of the capture filters, as shown in the following screenshot:

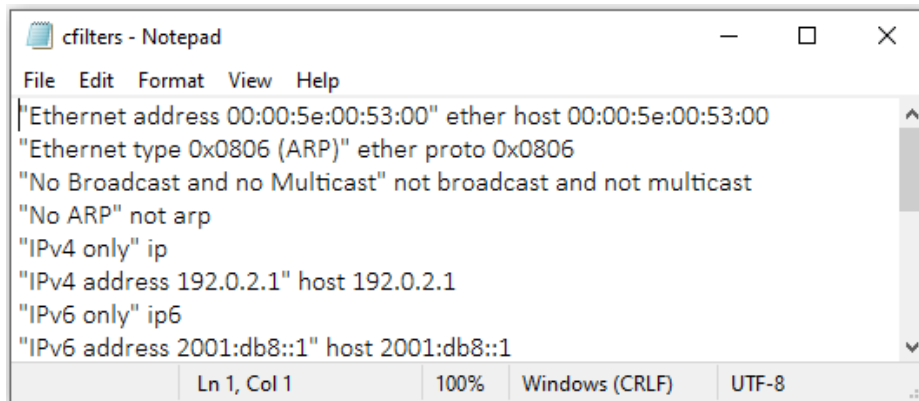


Figure 7.5 – cfilters.txt

Similar to the display filters, the capture filters are represented by a **Name | Filter** pairing as well. For example, when examining the filter that includes only **Transmission Control Protocol (TCP)** traffic, you will see the following:

- **Name:** The name of the filter, surrounded by quotation marks. For example, "TCP only".
- **Filter:** The syntax for the filter. For example, `tcp`.

### Note

While it is possible to open and modify either file in Notepad, as shown, it's safer and easier to use the Wireshark interface to make changes to the filters. For example, by going to the **Analyze | Display Filters...** menu option and then editing in the dialog box, you preserve the correct syntax.

Now that you have seen the main differences, let's take a closer look at the display filters.

# Comprehending display filters

While capturing traffic, or analyzing a pre-captured file, display filters help to narrow the scope and home in on specific types of traffic. It's not uncommon to have a capture with over 3,000 packets containing many different types of traffic.

When you launch Wireshark, you will see the startup screen. Across the top, below the icons, is the filter toolbar. Within the toolbar is the text **Apply a display filter...**, where you can easily apply and edit display filters, as shown here:

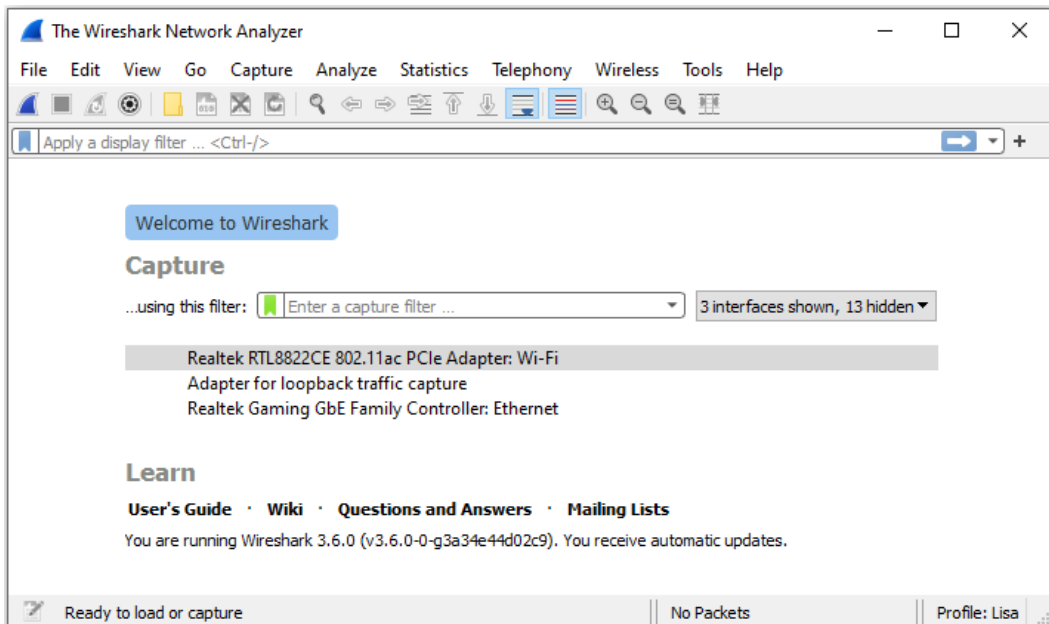


Figure 7.6 – Wireshark startup screen

You can create a simple filter on any of the protocols Wireshark supports by using a single protocol or adding a logical operator. For example, if you want to see TCP or ARP traffic, then you would use the `tcp || arp` display filter.

Wireshark's display filters can easily be modified. The following section illustrates how you can edit the display filters to customize your workflow.

## Editing display filters

After working with the display filters, you may need to change an IP address, port number, or make some other change. To edit the display filter, go to the **Analyze** menu, and then select **Display Filters...**, which will bring up the following dialog box:

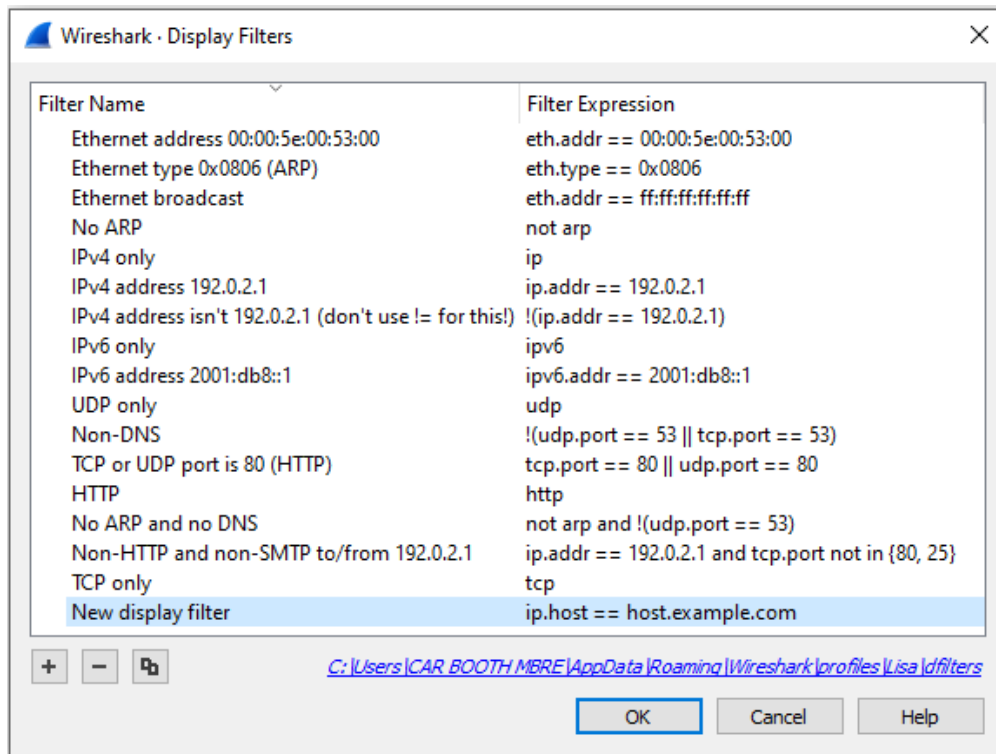


Figure 7.7 – Display Filters dialog box

Once there, you can select one of the three icons as shown in the lower left-hand corner of the **Display Filters** dialog box:

- A plus icon will *add* a new display filter. When selected, Wireshark will create a space where you enter a name on the left and the actual filter on the right, as shown in *Figure 7.7*.
- A minus icon will *delete* a display filter. Select (highlight) the filter you want to remove and hit the minus sign to remove the filter from the dialog box and update the `dfilters.txt` file.

- A copy icon will *copy* a display filter. Once copied, you can modify the filter without changing the original. Wireshark will then add the new filter to the `dfilters.txt` file.

In the next section, we'll see how, when you do get a display filter that works and you would like to reuse it, you can save it to a bookmark.

## Using bookmarks

On the right-hand side of the display filter, there is a blue toolbar icon called **bookmarks**. This is where Wireshark's built-in filters and any user-saved filters reside. If you click on the icon you will see options, along with several pre-loaded filters that you can use, as shown here:

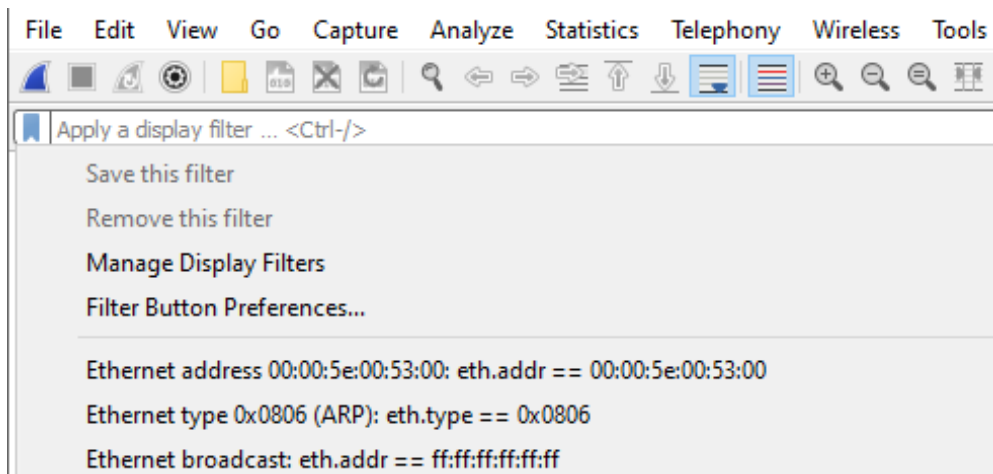


Figure 7.8 – Display filter bookmark dropdown

When working with bookmarks, you will see configuration options that include the following:

- **Save this filter:** After you create a filter, you can save the filter to the bookmark by selecting this option.
- **Remove this filter:** This will delete the filter currently in use that is stored in the `dfilters.txt` file.
- **Manage Display Filters:** This will open the **Display Filters** dialog box.
- **Filter Button Preferences...:** This will open the **Preferences** dialog box with the **Filter Buttons** option highlighted.

Below the selections, you will see a list of filters. Even if you have never saved a filter, you will see the list, as Wireshark will show the list of filters found in the `dfilters.txt` file.

Once you create your own filter or select one from the drop-down list, you can press *Enter* or click the blue arrow on the right-hand side of the display filter to run the filter.

On the far-right side of the display filter is a drop-down arrow (caret). When selected, you can see previously used filters, as shown in the following screenshot:

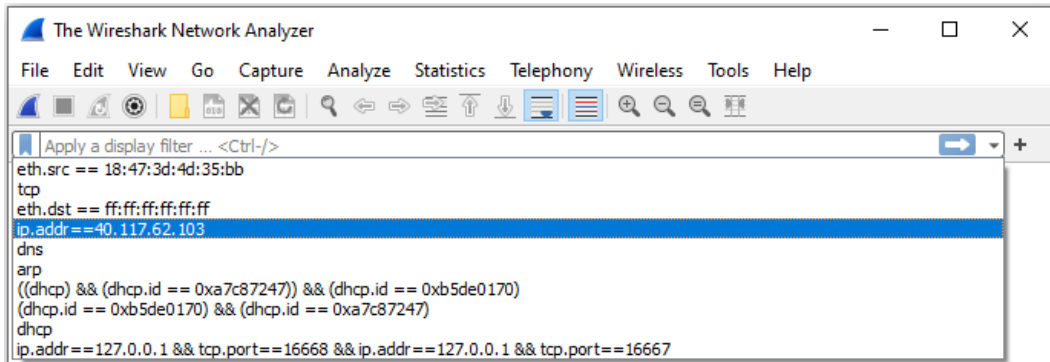


Figure 7.9 – Previously used display filters

A display filter can be applied before, during, or after packet capture. When you are ready to clear the filter, select the **X** on the right-hand side of the filter, as shown here:

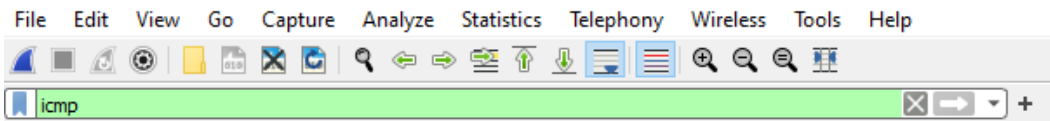


Figure 7.10 – The display filter toolbar

As we can see, display filters can be very helpful in providing a more targeted view of the capture. However, there may be times that you only want to gather a certain type of traffic. In that case, you would use a capture filter, which we'll discuss in the next section.

## Creating capture filters

Understanding how to use display filters is important. However, capture filters help target only the traffic you want to see. In addition, filtering can remove much of the noise, such as management traffic and ARP broadcasts.

You can create a capture filter in a couple of ways as follows:

- Go to the center of the startup screen and enter the filter following **...using this filter:** as shown in *Figure 7.6*.
- Go to the **Capture** menu and then select **Options...**, as shown here:

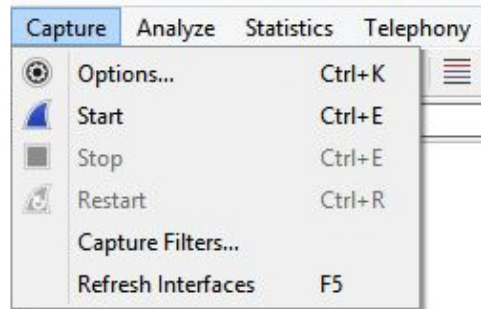


Figure 7.11 – Capture menu dropdown

Once there, you will see capture options, and at the bottom of the dialog box you will see the capture filter area, as shown in the following screenshot:

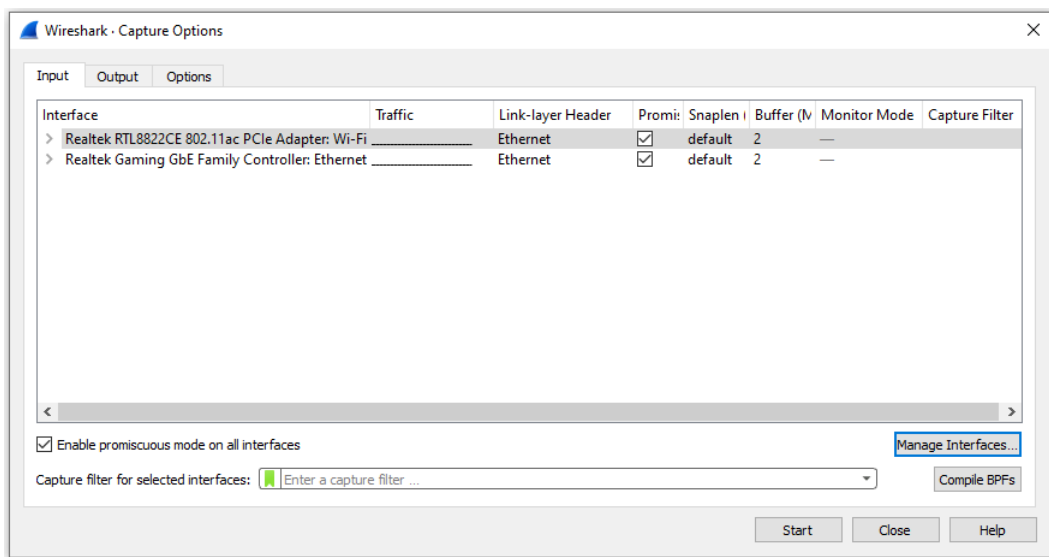


Figure 7.12 – Location to add a capture filter

Although both options will give you the ability to create a capture filter, I generally go to the **Capture** menu choice and select **Options...**, as this will allow me to see all the interfaces.

You can create a capture filter on any of the protocols Wireshark supports by using a single protocol or multiple ones and adding a logical operator. To clear the filter, select the **X** on the right-hand side of the capture filter.

To see your previously used capture filters, go to the right-hand side of the capture filter and select the down arrow to display the list. This is similar to the display filter drop-down list, as shown in *Figure 7.8*.

In addition to being able to modify the display filters, you can customize capture filters as well, as outlined in the following section.

## Modifying capture filters

To edit the capture filters, go to the **Capture** menu choice and then select **Capture Filters**, which will display a list of prebuilt filters, as shown in the following screenshot:

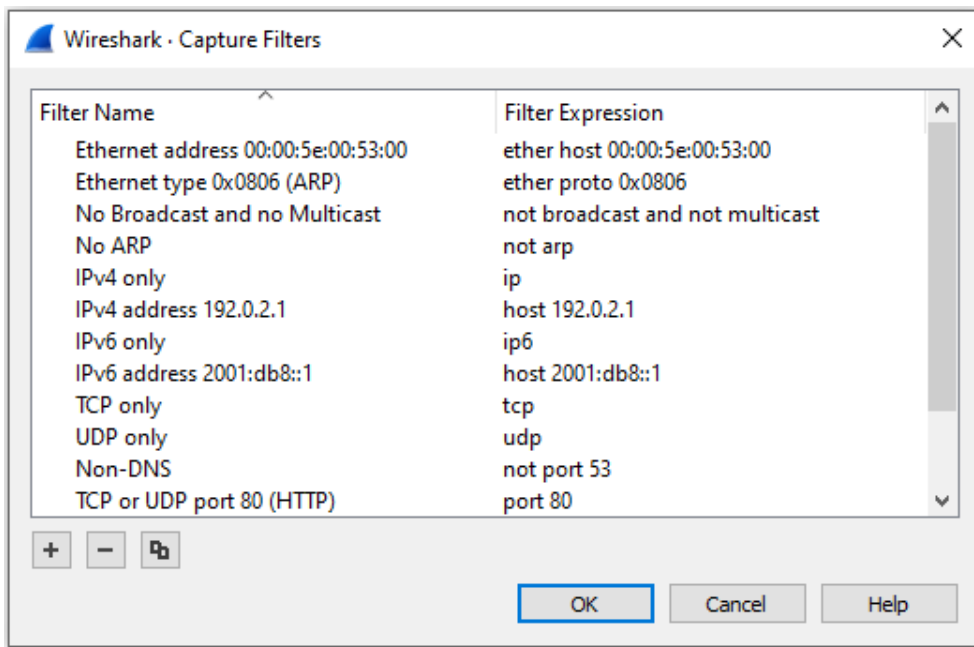


Figure 7.13 – Capture Filters dialog box

At the bottom of the dialog box, there are three icons:

- The plus icon will *add* a new capture filter. When selected, Wireshark will create a space where you enter a name on the left and the actual filter on the right.
- The minus icon will *delete* a capture filter. Select (highlight) the filter you want to remove and hit the minus sign to delete the filter from the dialog box and update the `cfilters.txt` file.
- The copy icon will *copy* a capture filter. Once copied, you can modify the filter without changing the original. Wireshark will then add the new filter to the `cfilters.txt` file.

Although the capture filters interface may look like the display filter toolbar, the syntax is different. As a result, you'll need to be careful when crafting a capture filter as it uses the **Berkeley Packet Filter (BPF)** syntax. In the next section, we'll walk through an example of creating a capture file using the appropriate syntax.

## Recognizing BPF syntax

Capture filters can reduce the number of packets obtained by gathering *only* the type of traffic you want to see. For example, if I need a filter to capture **File Transfer Protocol (FTP)** traffic only, then I might enter `ftp` in the capture filter, as I would in the display filter. However, you will see the syntax checker turn red, which means this is an invalid filter, as shown here:



Figure 7.14 – Invalid capture filter syntax

Although this filter would work as a display filter, you must write a capture filter that uses the correct syntax.

When creating a new capture filter, try using one of the prebuilt filters as a guide to properly build your filter. Let's see how this works.



## Crafting a capture filter

When you need to create a capture filter, find a capture filter *similar* to the one you need, select the filter, and click the copy icon. Wireshark will copy the filter and place it at the end, where you can edit it.

Let's go through an example of creating a capture filter for FTP by copying an existing filter:

1. Go to the **Capture** menu and select **Capture Filters**.
2. Select the `HTTP TCP port (80)` capture filter and then click the copy icon. Wireshark will place the copied filter at the end of the list, as shown in the following screenshot:

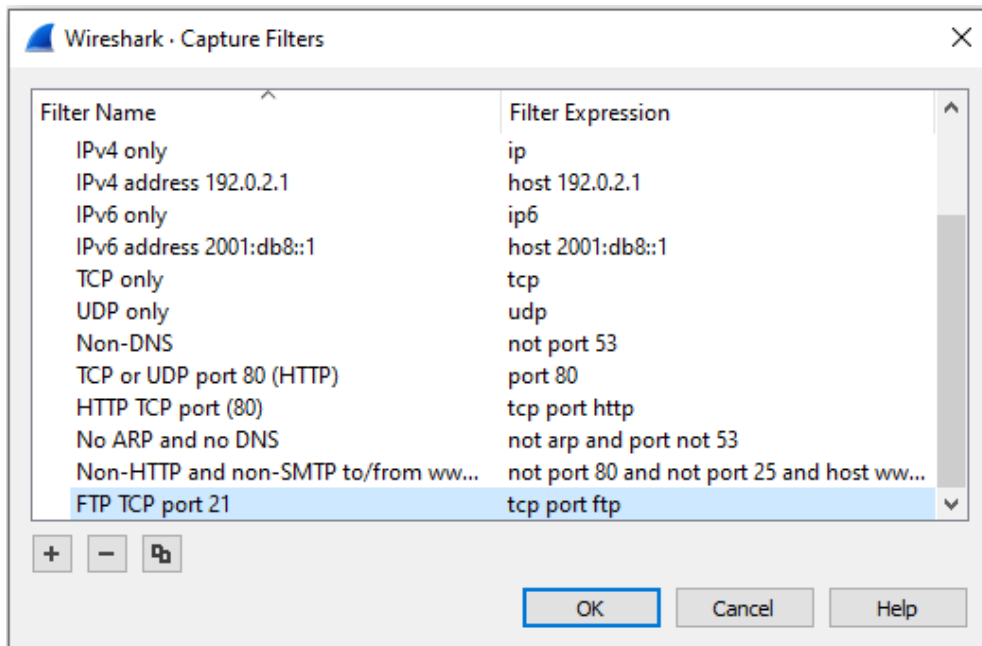


Figure 7.15 – Create new capture filter

3. To edit the filter, change the name to `FTP` and change the filter to `tcp port ftp` or `tcp port 21`.
4. Close the **Capture Filters** dialog box.
5. Restart Wireshark so that the newly created capture filter is recognized.

To use the newly created filter, follow these steps:

1. Go to the **Capture** menu choice, and then select **Options....**
2. Click the interface that will be used to capture traffic. For example, in the following screenshot, the **Microsoft: Wi-Fi** interface is selected.
3. In the capture filter area, drop down the green bookmark and select the filter you just created. The bookmark will turn yellow, as shown here:

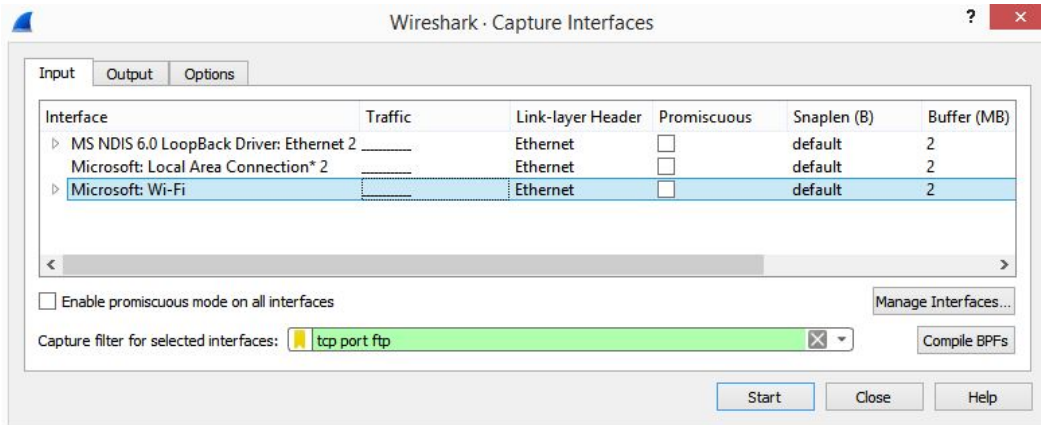


Figure 7.16 – FTP capture filter

4. Once you click **Start** to begin your capture, you will only capture FTP traffic.

#### Note

When you are done using a capture filter, make sure you remove any trace of the filter by going into the **Capture** menu and then **Options....** Once open, delete the capture filter so that you can capture all traffic again.

You might have used a capture filter that works well, and you would like to preserve the filter for future use. Next, let's take a look at how you can save your capture filter to a bookmark.

## Bookmarking a filter

On the right-hand side of the capture filter is a green toolbar icon called **bookmarks**, where the built-in capture filters are stored. Any time you create and save a filter, Wireshark will store the filter in the bookmark.

If you click on the green toolbar icon, you will see a list of capture filters, as shown in the following screenshot:

### Capture

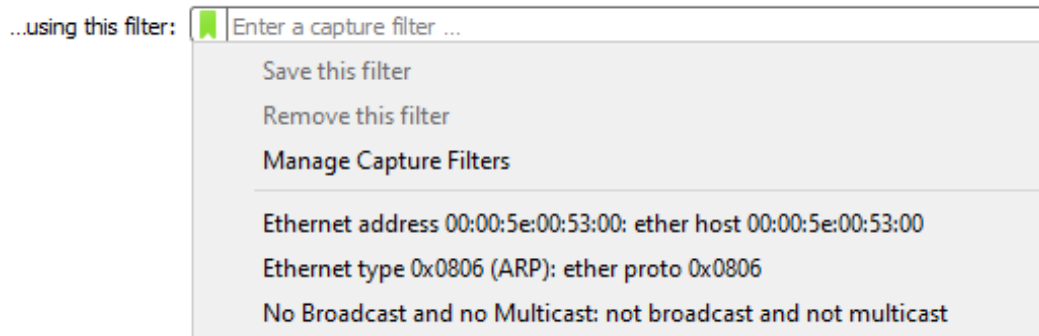


Figure 7.17 – Capture filter bookmark drop-down

Once there, you can select one of the filters, and Wireshark will populate the capture filter field.

If you create a filter and want to save it, drop down the bookmark icon and select **Save this filter**, and Wireshark will store the filter in the bookmark.

Capture filters can be useful; however, keep in mind that while using a capture filter, you might miss important traffic that can help during troubleshooting or searching for evidence of intrusion.

During analysis, you may need to create an expression using specific field values. The following section outlines how to use the expression builder.

## Understanding the expression builder

In addition to building simple display filters, Wireshark has the ability to create an expression that zeroes in on specific field values. To build an expression, go to **Analyze** and then select **Display Filter Expression**, as shown here:

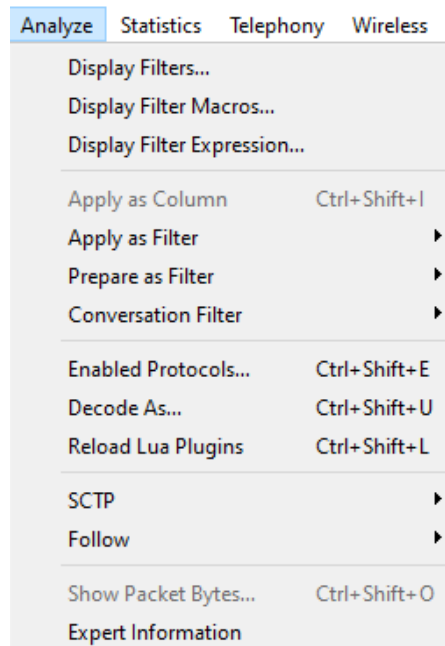


Figure 7.18– The Display Filter Expression menu choice

Click the link to launch the expression builder. On the left-hand side, you will see a list of all of Wireshark's supported protocols, as shown in the following screenshot:

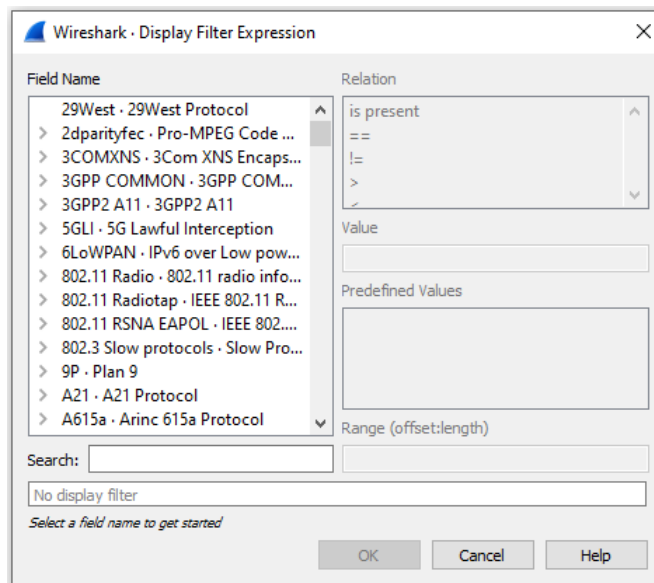


Figure 7.19 – Display Filter Expression

Wireshark is capable of dissecting hundreds of protocols with more added all the time, so the list will be long.

To further refine the filter, you can select from the four variables listed on the right-hand side:

- **Relation:** This is a list of comparison operators to compare a field value against another value using logical operators:
  - **is present:** Indicates the selected field exists in the capture
  - **==:** Equal to
  - **!=:** Not equal to
  - **>:** Greater than
  - **<:** Less than
  - **>=:** Greater than or equal to
  - **<=:** Less than or equal to
- **Value:** Indicates the appropriate value required. Wireshark populates this with the appropriate type of value, that is, Boolean or string.
- **Predefined Values:** Wireshark populates this with the appropriate values for a given field.
- **Range (offset:length):** This allows you to enter a range of integers such as 4 - 8 or 12 - 20 if they are an appropriate selection for this field or filter.

Now that you have a good understanding of what the expression builder can do, let's go through a simple example of building a custom filter.

## Building an expression

In this example, I want the filter to show me all the packets that have the **TCP SYN** flag present. On the left-hand side of the **Display Filter Expression** dialog box, I have drilled down and selected the `tcp.flag.syn` value. Wireshark will populate the following on the right-hand side of the dialog box:

- **Relation:** `==`
- **Value:** `1`
- **Predefined Values:** `Set`

**Range** is grayed out as this is not an appropriate selection for this field value.

Once all the values are selected, Wireshark populates the display filter area along the bottom with the generated expression. In this case, the filter is `tcp.flags.syn == 1`, as shown here:

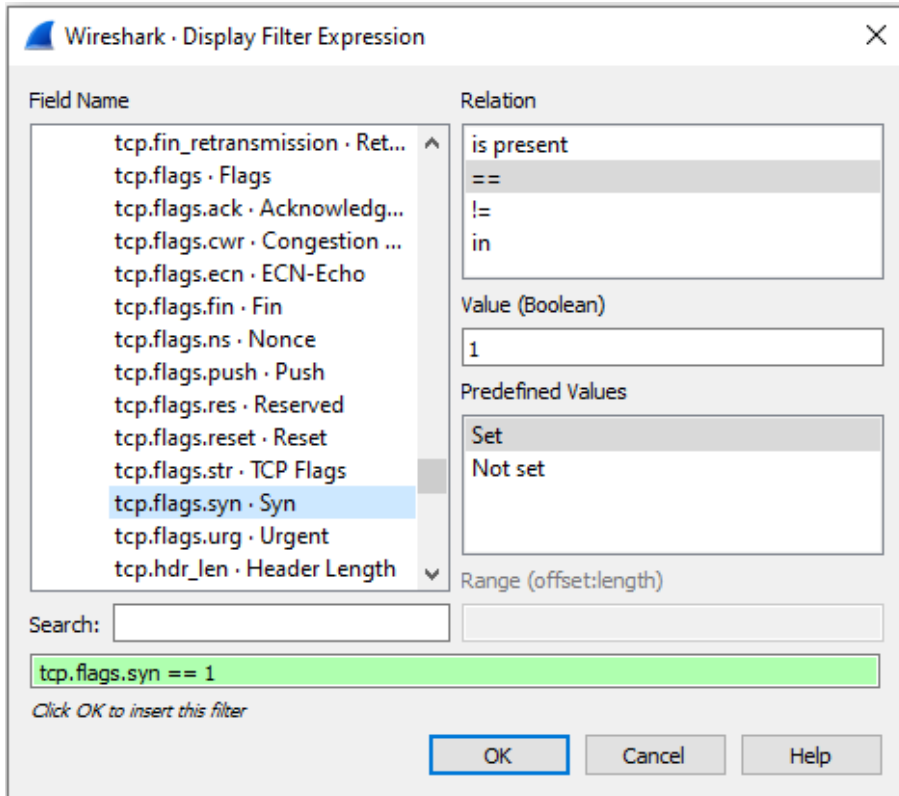


Figure 7.20 – TCP SYN flag filter

At this point, we can select either of the following:

- **Cancel:** This will exit the expression builder and will not create a filter.
- **OK:** This will place the filter in the display filter toolbar.

Once the filter is in the display filter toolbar, you can make any necessary modifications, such as changing `tcp.flags.syn == 1` to a modified filter such as `tcp.flags.syn == 0`. Once you are satisfied with the filter, you can run the filter by pressing *Enter* or clicking the blue arrow on the right-hand side of the display filter.

As you can see, the expression builder is an easy way to go through the process of building a custom filter.

In the next section, we'll see that, while working with a packet capture, Wireshark has many shortcuts that allow you to quickly create a filter to streamline your workflow.

## Discovering shortcuts and handy filters

Over the years, Wireshark has evolved. Now more than ever, it's very easy to create a display filter on the fly while analyzing by simply right-clicking and choosing to apply or prepare as filter.

In this section, let's take a look at the many ways in which we can apply a filter, without going through the complicated exercise of launching the expression builder. In addition, we'll see some handy filters that you can use to get right down to the issue. We'll start with an overview of the many shortcuts to use when filtering traffic.

## Embracing filter shortcuts

While working with Wireshark in the **Packet Details** panel, you might want to filter on a specific IP address or a particular port number. Once you identify the item of interest, you can right-click on it and you will see several filter shortcuts, as shown here:

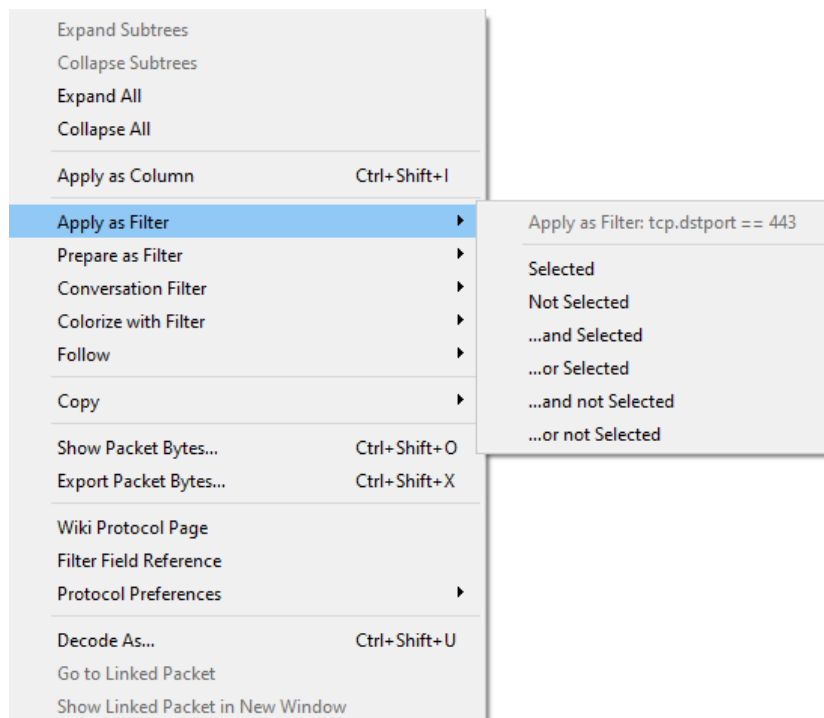


Figure 7.21 – Right-click to view filter shortcuts

Although there are many options when you right-click, in the center you can see the shortcuts that deal with filters. The following options are available:

- **Apply as Filter:** When selected, it will create and run a selected field value.
- **Prepare as Filter:** When selected, it will create and place the selected field value in the display filter area, giving you a chance to make any modifications or add to the filter.
- **Conversation Filter:** When selected, it allows you to follow the conversations according to protocols, such as **Ethernet**, **IPv6**, and **TCP**, as shown in the following screenshot:

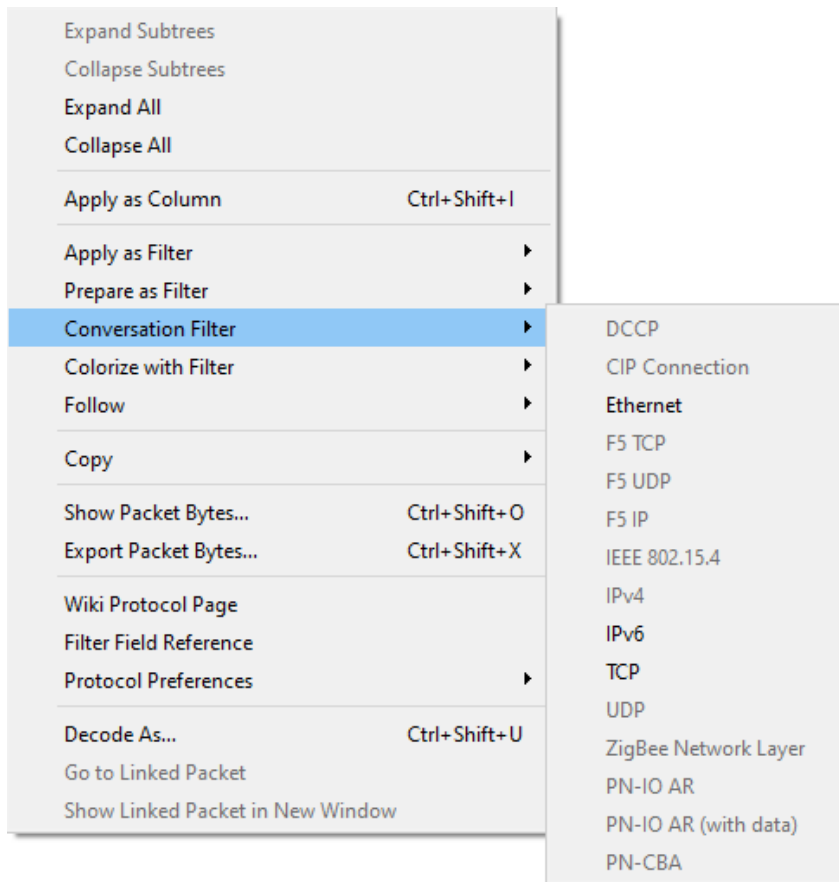


Figure 7.22 – Conversation Filter selections



- **Colorize with Filter:** This allows you to colorize a specific conversation. As you can see, you can select from the many available colors, or you can create your own coloring rule, as shown in the following screenshot:

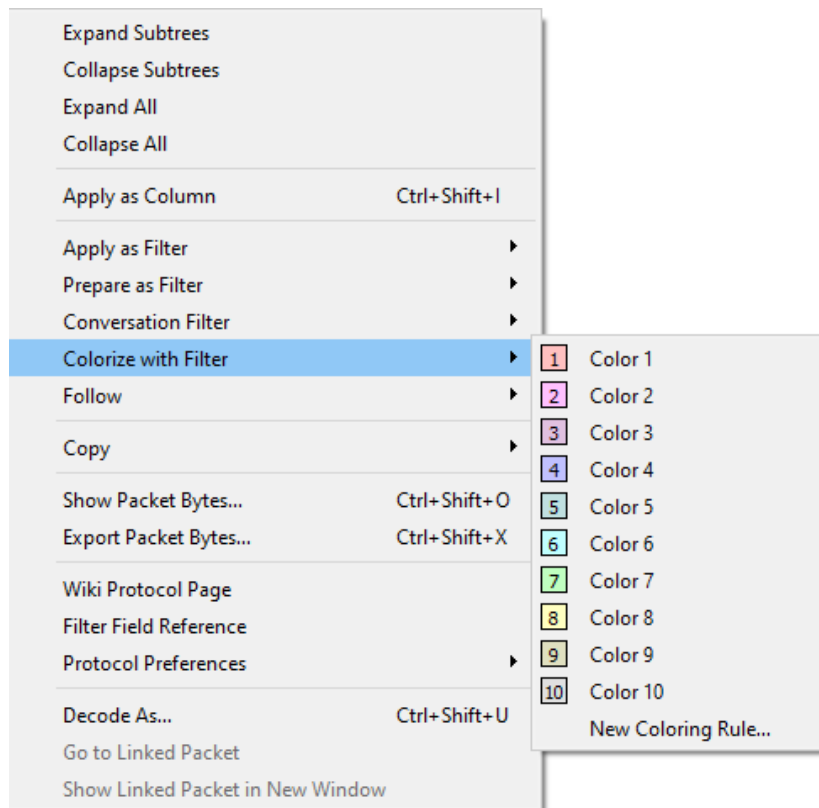


Figure 7.23 – Colorize with Filter

**Note**

If you have applied a filter to colorize a specific stream and then want to undo your changes, go to **View | Colorize Conversation** and then select **Reset Colorization**.

When you right-click on a field value and select either **Apply as Filter** or **Prepare a Filter**, you will see additional choices when building a filter. Let's investigate how this works.

## Viewing additional choices

Even within the shortcuts, you will find additional methods to filter traffic when using either **Apply as Filter** or **Prepare as Filter**. As shown in *Figure 7.21*, you can right-click to view filter shortcuts. The following list shows how you can select simple filters or add logical operators:

- **Selected:** Selects the current field value.
- **Not Selected:** Creates a filter that removes the selected field. For example, if I right-click on destination port 443 and select **Not Selected**, then Wireshark will generate `!(tcp.dstport == 443)` and place it in the display filter.
- **...and Selected:** Adds a field value to the filter.
- **...or Selected:** Creates an OR filter.
- **...and not Selected:** Adds a filter that removes the selected field.
- **...or not Selected:** Creates an OR filter with a filter that removes the selected field.

Using filters is one of the tools that helps you home in on a problem. The next section provides some suggestions on useful filters that can help you when searching for specific types of traffic.

## Applying useful filters

Wireshark is a common tool used by developers, network administrators, students, and security analysts. Network administrators use Wireshark to investigate the many issues that can surface and cause the network to degrade or spread malware.

Once you have worked with Wireshark for any length of time, you'll learn tips and techniques to streamline your workflow. One common task is to apply various display filters.

### Using display filters

While you can always build a filter on the fly, it's common to have some handy display filters that include the following:

- `http.request:` This searches the capture file for any HTTP GET or POST requests.
- `tcp.port==xxx:` Use this filter if you are monitoring TCP traffic by using a specific port.

- `tcp.stream eq X`: This filter will follow a specific stream, where X is the stream index.
- `!(arp or icmp or dns)`: This filter will eliminate arp, dns, and icmp traffic.
- `vlan.id ==X`: This shows a specific vlan.

Wireshark also provides a handy drag-and-drop feature, where you can simply drag a field from the packet tree and drop it into the display filter.

In addition to display filters, there are times when capture filters are used to collect specific traffic.

## Using capture filters

As we know, capture filters narrow the scope when capturing traffic. Some practical examples include the following:

- `port ftp || port ftp-data`: This will capture FTP traffic.
- `ip host x.x.x.x`: This will capture traffic from a specific host.
- `ip multicast`: This will capture multicast traffic.

The Wireshark wiki also lists several capture filters that are used to detect malware. For example, `dst port 135 and tcp port 135 and ip[2:2]==48` will display evidence of the Blaster worm virus.

### Note

Read more about the Blaster worm virus by visiting <https://docs.microsoft.com/en-us/troubleshoot/windows-server/security-and-malware/blaster-worm-virus-alert>.

There are many ways to filter traffic to only display the traffic you want to see, which helps remove unnecessary traffic and improve your analysis skills. Depending on the type of network you work with on a daily basis, you will most likely build your own arsenal of filters.

## Summary

Wireshark is a powerful tool that allows us to capture and analyze traffic. In this chapter, we reviewed how to study a capture more effectively by using the built-in filter functions. We learned how to use a display filter and discussed how it can provide either a simple filter showing only a single protocol, or a combination of field values. We then reviewed how you can edit the display filters, along with how you can create your own and store them for easy reference in the bookmarks.

We then covered how capture filters are applied prior to gathering traffic to display only a specific type of traffic. We also saw how to drill down on a specific field value by building an expression. In addition, we saw that when carrying out a granular investigation, we can create filters that include logical operators and specific field values. With the many ways to filter traffic, we looked at the shortcuts to build filters on the fly while conducting analysis. We then closed by discovering the benefits of having several useful filters in your arsenal.

In the next chapter, we will take a look at the **Open Systems Interconnection (OSI)** model, an essential concept to grasp in order to be effective at packet analysis. So that you have a better understanding of the OSI model, we'll review each of the seven layers. We'll discuss addressing, the protocol data units, and the protocols in each layer. In addition, we'll step through the process of encapsulation as the data is readied for frame formation in order to be sent via the appropriate media.

## Questions

Now, it's time to check your knowledge. Select the best response, then check your answers with those provided in the *Assessments* appendix:

1. When creating a display filter, if the background is \_\_\_\_, then you have entered a valid filter.
  - A. red
  - B. green
  - C. yellow
  - D. cyan

2. When creating an expression using the expression builder, you can refine the filter by modifying any of the four variables listed on the right-hand side, which are **Relation**, \_\_\_\_\_, **Predefined Values**, and **Range**.
  - A. **Float**
  - B. **Integer**
  - C. **Value**
  - D. **Boolean**
3. To create a *capture* filter to see only DNS requests and responses, you would enter \_\_\_\_\_ in the capture filter.
  - A. `tcp port 53`
  - B. `DNS`
  - C. `dns`
  - D. `udp port 53`
4. If you need to build a filter to zero in on specific field values, you can use the \_\_\_\_\_ builder.
  - A. Expression
  - B. Berkeley
  - C. EPAN
  - D. Dissector
5. When using either **Apply as a Filter** or **Prepare as Filter**, you will see additional choices to create a simple filter or add logical operators. \_\_\_\_\_ adds a filter that removes the selected field.
  - A. **...or Selected**
  - B. **...and Selected**
  - C. **Not Selected**
  - D. **...and not Selected**

6. In Wireshark, the capture filter uses the \_\_\_\_\_ syntax.
  - A. Expression
  - B. BPF
  - C. EPAN
  - D. Dissector
7. When selecting the shortcut \_\_\_\_\_, it will create and place the selected field value in the display filter area, giving you a chance to make any modifications or add to the filter.
  - A. **Prepare as Filter**
  - B. **Apply as Filter**
  - C. **Build a Filter**
  - D. **Generate a Filter**

## Further reading

Please refer to the following links for more information:

- Read more about the Berkeley Packet Filter, *The BSD Packet Filter: A New Architecture for User-level Packet Capture*, which is found at <https://www.tcpdump.org/papers/bpf-usenix93.pdf>.
- To view a list of handy 802.11 filters, visit [https://semfionetworks.com/wp-content/uploads/2021/04/wireshark\\_802.11\\_filters\\_-\\_reference\\_sheet.pdf](https://semfionetworks.com/wp-content/uploads/2021/04/wireshark_802.11_filters_-_reference_sheet.pdf).
- To learn more about wireless filters along with several examples, visit <https://tiebing.blogspot.com/2015/02/wireshark-display-filters.html>.
- For an extensive list of capture filter examples, go to the Wireshark wiki at <https://wiki.wireshark.org/CaptureFilters>.
- To find a complete list of built-in field values used in building filters, go to <https://www.wireshark.org/docs/dfref/>, where you can select a protocol link and learn more about the supported field values.



# 8

## Outlining the OSI Model

Effective packet analysis begins with a solid understanding of the **Open Systems Interconnection (OSI)** model. The OSI model is a seven-layer framework that outlines how the **operating system (OS)** transforms, encapsulates, and prepares data for transport on the network. In this chapter, we'll cover the seven layers and the role and purpose of each layer. Additionally, you'll learn how within the OSI model, there are three layers that use a specific address or identifier. The Transport layer has an associated port, The Network layer uses an **Internet Protocol (IP)** address, and the Data Link layer requires a **Media Access Control (MAC)** address. So that you are able to differentiate identifiers for each of the layers, we'll review and describe the significance of each one.

Then, we'll take a look at the **Protocol Data Unit (PDU)** for each layer.

Once done, you will be more familiar with the terminology and have a better understanding of some of the protocols in each layer. From the **Hypertext Transfer Protocol (HTTP)** request when retrieving data from a web page to the bits as it travels across the network, you'll know how data transforms. In addition, we'll learn how each PDU feeds into the next layer to properly format the frame so that the data can be sent using the appropriate media.



In this chapter, we will cover the following main topics:

- An overview of the OSI model
- Discovering the purpose of each layer, the protocols, and the PDUs
- Exploring the encapsulation process
- Demonstrating frame formation in Wireshark

## An overview of the OSI model

The OSI model is a seven-layer framework that outlines the main functions of each layer, which represents a grouping of protocols that perform a specific function. Understanding the model will better prepare you for analyzing traffic with Wireshark. In addition, you'll be able to recognize the role of the protocols involved during a transaction.

### Note

All protocols have a specific purpose on the network. Today, Wireshark has dissectors for most protocols, and new ones are added all the time.

In this section, we'll take a step back and learn how the framework began, along with recognizing who benefits from using this model.

Let's start with the inspiration behind the OSI model.

## Developing the framework

We started this journey during the period of the late 1960s to the mid-1970s, where we saw an expansion in computing, along with advances in technology in general. In addition to this, we witnessed the development of computers, from small personal computers to supercomputers, such as the Cray in 1976, and video games, such as Pong, in 1972.

Concurrent to this development, two international organizations, the **International Organization for Standardization (ISO)** and the **International Telegraph and Telephone Consultative Committee (CCIT)**, began working on a reference model to define and standardize networking interoperability. Ultimately, in 1983, the two organizations developed the OSI model.

The OSI model serves many purposes, including the following:

- Providing a common framework for developers
- Narrowing down problems for network administrators

- Enabling interoperability among layers and devices communicating with one another.
- Breaking down each layer to help students better understand the overall process of data encapsulation.

As a result, many individuals benefit from using the OSI model, as we will outline next.

## Using the framework

While the OSI model is a reference model, there are many groups that utilize it. These groups include the following:

- Developers reference the OSI model to outline how systems communicate with one another.
- Network administrators refer to issues according to the layer they feel is responsible for the malfunction when troubleshooting network problems.
- Equipment manufacturers rely on the OSI model to ensure their products will work across all layers.

Additionally, students use the model to begin their journey into networking. A staple of every college freshman networking class is an introduction to the OSI model. In most cases, these students have never heard of this, so presenting a complex topic in a simple manner can be difficult. As this is their first encounter, it's important to convey this information in an easy-to-learn manner.

In addition, the model provides a visual description of what is going on in each layer. The model defines the protocols, PDUs, and the purpose of each layer, as outlined in the following section.

## Discovering the purpose of each layer, the protocols, and the PDUs

In this section, we'll take a closer look at each of the seven layers of the OSI model and describe the following:

- The layers of the OSI model, from layer seven to layer one: **Application**, **Presentation**, **Session**, **Transport**, **Network**, **Data Link**, and **Physical**.
- The three layers that use source addressing and destination addressing during encapsulation: the **Transport**, **Network**, and **Data Link** layers.

- The PDU for each layer, which defines what shape the data is in as it is passed to the layer above or the layer below.

In the following diagram, we can see a summary of the OSI model:

	OSI	Address	PDU
7	Application		Data
6	Presentation		
5	Session		
4	Transport	Port	Segment
3	Network	IP	Packet
2	Data Link	Mac	Frame
1	Physical		Bits

Figure 8.1 – The OSI model

Before diving into the layers, it is helpful to use a mnemonic device to remember the first letter of each layer. With the OSI, we have two, as shown in the following diagram:

Top-down Mnemonic	Bottom-up Mnemonic
All	Please
People	Do
Seem	Not
To	Throw
Need	Sausage
Data	Pizza
Processing	Away

Figure 8.2 – OSI mnemonics

Once you know the names of the layers, the next step is to tackle the role, purpose, and protocols of each of the layers.

When outlining each layer, usually, I start with the **Application** layer, as this is where we initiate contact with the network. Let's discuss this next.

## Evaluating the Application layer

The **Application** layer (or layer 7) contains the protocols that allow process-to-process communications, which enable us to do the following:

- Retrieve a web page
- Fetch or send email
- Upload files to an FTP server
- Request a dynamically assigned IP address

On a TCP/IP network, each **Application** layer protocol follows specific recommendations, requirements, and options, according to its function. Let's talk about a few of the protocols alongside the PDU for this layer.

## Exploring the Application layer protocols

The Application layer has hundreds of protocols. Many common or well-known protocols were developed and standardized very early in the 1980s. Some are deprecated and we rarely see them, such as Telnet and **Simple Authentication and Security Layer (SASL)**. However, new protocols are developed, as needed, to keep up with today's demands, such as **Constrained Application Protocol (CoAP)** and **Message Queuing Telemetry Transport (MQTT)**, which are both used with **Internet of Things (IoT)** devices. The following is a shortlist of Application layer protocols:

Protocol	Purpose
Simple Mail Transfer Protocol (SMTP)	This transports email.
Hypertext Transfer Protocol (HTTP)	This ensures the delivery (transfer) of web pages in the proper format.
File Transfer Protocol (FTP)	This transfers files between a client and server across the network.
Message Queuing Telemetry Transport (MQTT)	This is used with IoT devices, sensors, and mobile devices as a lightweight messaging protocol.

Table 8.1 – The Application layer protocols

In many cases, the protocols in this layer are involved in a series of client requests and server responses. Next, let's talk about the PDU for this layer.

## Understanding the Application layer PDU

The **PDU** for the **Application** layer is data that is specific to the protocol in use.

The header structure for each protocol will vary, as each is application-specific. In addition to this, the header for the client is generally different from the header for the server.

Most protocols will have an associated port, which will be found in the Transport layer header. The following is a summary of the Application layer:

Layer	Purpose	Protocols	PDU
Application	This initiates contact with the network.	HTTP, DNS, and FTP	Data

Table 8. 2 – Summary of the Application layer

After the data leaves the Application layer, it is then passed to the **Presentation** layer, in order to properly format the data. Let's explore this next.

## Dissecting the Presentation layer

The **Presentation** layer (or layer 6) is responsible for proper data formatting, along with optional compression and encryption. This layer ensures that the data is in the proper format, either before presenting the data to the application (such as a Word document) or prior to sending it out onto the network. For example, if you download a file from the internet with the .gz extension, then the **Presentation** layer will search for an application to associate it with, so the OS can open the file correctly. If the application is not installed, then you will see a message, as shown in the following screenshot:

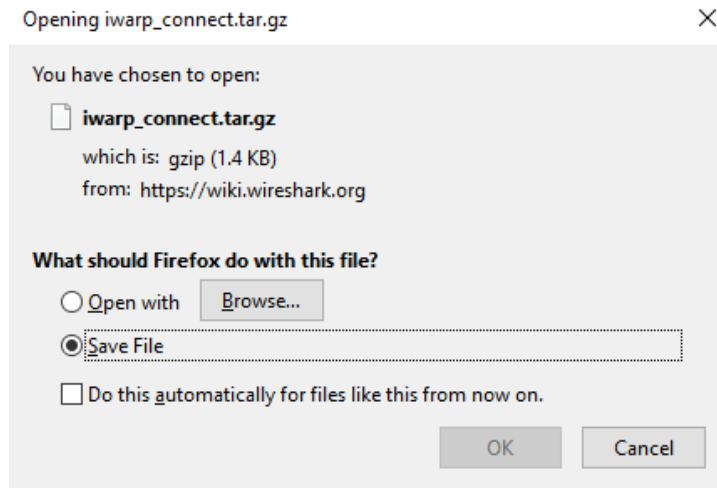


Figure 8.3 – The dialog box to select an application

If you do not have the application installed, then you can go in and manually select the application with which you want to open the file or obtain and install the correct application.

The **Presentation** layer also provides optional services in which to compress and decompress data. Compression removes redundancy and makes data smaller. This function is optional, as not all data is compressed.

Additionally, this layer handles encryption, which involves scrambling data using a key so that it is in an unreadable format that does not make sense to anyone unless they have the key. Because encryption is also an optional function, this might not be required.

In the **Presentation** layer, there are a few protocols, which we'll investigate in the following section, along with the PDU.

## Describing the protocols and the PDU

In this layer, the protocols deal with proper data translation and encoding/decoding, such as **External Data Representation (XDR)**. In addition, because of the **Presentation** layer's role in encryption, you'll find the following protocols:

- **Transport Layer Security (TLS)/Secure Socket Layer (SSL)**: These protocols secure end-to-end communications such as bank transactions and web page retrieval using encryption.
- **Secure/Multipart Internet Mail Extensions (S/MIME)**: These protocols digitally sign and encrypt email messages.

At the **Presentation** layer, the **PDU** is still data, and we see the data being translated or converted into the correct format. The following is a summary of the presentation layer:

Layer	Purpose	Protocols	PDU
Presentation	Formatting and optional compression and encryption	TLS/SSL and S/MIME	Data

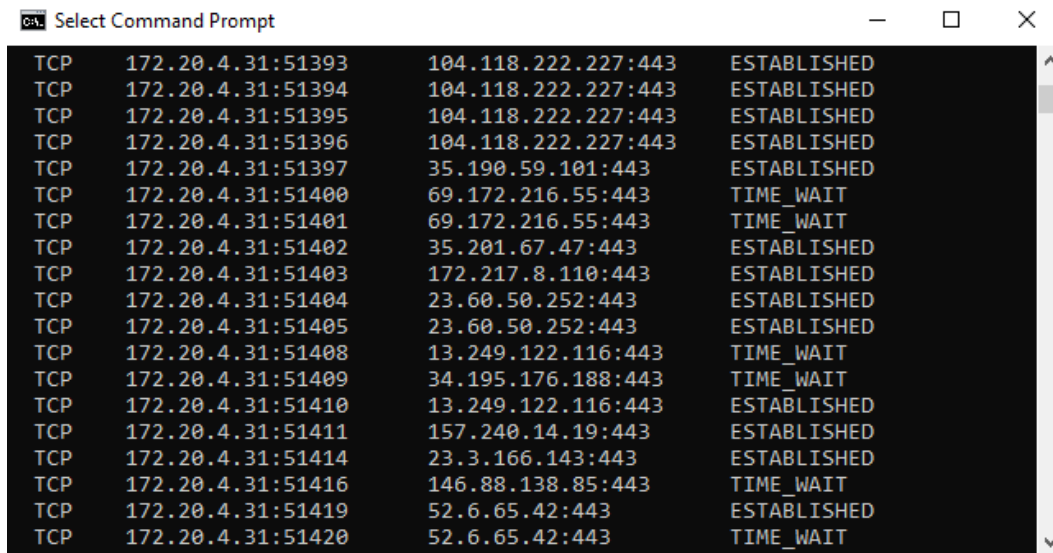
Table 8.3 – Summary of the Presentation layer

In many ways, the Presentation layer is an extension of the Application layer. Next is the Session layer, where we see all of the key elements of session management.

## Learning about the Session layer

The **Session** layer (or layer 5) is responsible for setting up, maintaining, and tearing down a session. Before any data is exchanged after initiating contact with the network, the OS must establish a session. When creating a session, the OS will create the appropriate socket, which is an IP address, and a port. This is so that the two endpoints can exchange data with one another.

When communicating on the network, you will have multiple concurrent sessions and connections established. You can see your active connections by going to the command line and running `netstat`, as shown in the following screenshot:



```

Select Command Prompt
TCP    172.20.4.31:51393    104.118.222.227:443    ESTABLISHED
TCP    172.20.4.31:51394    104.118.222.227:443    ESTABLISHED
TCP    172.20.4.31:51395    104.118.222.227:443    ESTABLISHED
TCP    172.20.4.31:51396    104.118.222.227:443    ESTABLISHED
TCP    172.20.4.31:51397    35.190.59.101:443      ESTABLISHED
TCP    172.20.4.31:51400    69.172.216.55:443      TIME_WAIT
TCP    172.20.4.31:51401    69.172.216.55:443      TIME_WAIT
TCP    172.20.4.31:51402    35.201.67.47:443       ESTABLISHED
TCP    172.20.4.31:51403    172.217.8.110:443      ESTABLISHED
TCP    172.20.4.31:51404    23.60.50.252:443       ESTABLISHED
TCP    172.20.4.31:51405    23.60.50.252:443       ESTABLISHED
TCP    172.20.4.31:51408    13.249.122.116:443     TIME_WAIT
TCP    172.20.4.31:51409    34.195.176.188:443     TIME_WAIT
TCP    172.20.4.31:51410    13.249.122.116:443     ESTABLISHED
TCP    172.20.4.31:51411    157.240.14.19:443      ESTABLISHED
TCP    172.20.4.31:51414    23.3.166.143:443       ESTABLISHED
TCP    172.20.4.31:51416    146.88.138.85:443      TIME_WAIT
TCP    172.20.4.31:51419    52.6.65.42:443         ESTABLISHED
TCP    172.20.4.31:51420    52.6.65.42:443         TIME_WAIT

```

Figure 8.4 – Netstat showing active connections

In the preceding screenshot, we only see **Transmission Control Protocol (TCP)** connections. TCP is a connection-oriented protocol and both endpoints need to communicate the status of the data transaction with one another. As a result, you will see a local and foreign address, along with the state of the transaction. For example, the following represents a connection:

```
TCP 172.20.4.31:51393 104.118.222.227:443 ESTABLISHED
```

In addition to managing the communication between two endpoints, the **Session** layer provides other services that include the following:

- **Authentication:** This validates and identifies an entity by requiring a password or another form of authentication.
- **Authorization:** This allows access to system resources if the entity has the appropriate permissions.
- **Checkpointing:** This monitors the session for errors and ensures that all data has been received. If there are errors in transmission, the Session layer might re-request any missing data.

After the communication stream has ended, the **Session** layer safely closes the session. Next, let's look at some of the key protocols in this layer.

## Recognizing the protocols and the PDU

There are several protocols that exist in the **Session** layer. Although most protocols might originate in other layers, these protocols begin, in part, in the Session layer:

- **Real-Time Transport Control Protocol (RTCP):** This works alongside the **Real-Time Transport Protocol (RTP)** to deliver control information to all participants in a **Voice over IP (VoIP)** call.
- **Domain Name System (DNS):** This resolves a hostname to an IP address in order for a session to take place.
- **Point-to-Point Tunneling Protocol (PPTP):** This creates a VPN by using a generic routing encapsulation tunnel to provide a more secure way to deliver data than using plain text.
- **Remote Procedure Call (RPC):** This allows a program to run a subroutine on another host on a shared network.



At the **Session** layer, the PDU is data. The following is a summary of the Session layer:

Layer	Purpose	Protocols	PDU
Session	To set up, maintain, and tear down a session	DNS and RPC	Data

Table 8.4 – Summary of the Session layer

The **Session** layer manages all aspects of a session, enabling hosts to communicate in a conversation with one another. At this point, the data then moves to the **Transport** layer, where it now becomes a segment that has the necessary port addressing in the Transport layer header.

## Appreciating the Transport layer

The **Transport** layer (or layer 4) is responsible for transporting the data, either using a connectionless or connection-oriented protocol across the network. The encapsulation process starts at this layer. The data will have additional headers added as it traverses down the layers to become a frame, ready to be sent on the network.

The transport protocol that is selected will depend on the application. Data is primarily transported using either TCP or **User Datagram Protocol (UDP)**. However, the Transport layer has several other protocols. Let's take a look at them.

## Differentiating the protocols and the PDU

The Transport layer has several protocols to transport data, including the following:

Protocol	Purpose
TCP	This is a connection-oriented protocol that ensures reliable data transfer.
UDP	This is a connectionless protocol that is used when speed, not reliability, is required.
Stream Control Transmission Protocol (SCTP)	This reliably transmits data streams that have more than one IP address.
Reliable User Datagram Protocol (RUDP)	This extends UDP by providing TCP-like qualities, such as flow control, acknowledgments, and re-transmitting lost packets.

Table 8.5 – Common protocols in the Transport layer

Although there are other lesser-known Transport layer protocols, we will discuss the two predominant protocols, TCP and UDP, starting with the more widely used protocol, TCP.

### Providing reliability with TCP

TCP is a connection-oriented protocol that has end-to-end reliability. TCP begins a session with a three-way handshake and ends the session with an exchange of **finis** (FIN) packets.

TCP actively sequences and acknowledges data, using the field values in the 11-field header, to ensure that all the data arrives at the end device.

Once inside a connection, TCP progresses through a series of states. For example, in *Figure 8.4*, you can see the ESTABLISHED and TIME\_WAIT states.

The TCP states are listed as follows:

- **LISTEN**: The system waits for a request from a remote host to connect.
- **SYN\_SENT**: After the client sends a request for a connection, the system waits for a response.
- **SYN\_RECEIVED**: After the SYN request has been returned to the client, the server waits for a final ACK to start the connection.
- **ESTABLISHED**: This is a normal state where the two endpoints are actively communicating.
- **FIN\_WAIT\_1**: The host waits for either an ACK in response to a FIN sent to the remote host or a FIN request from the remote host.
- **FIN\_WAIT\_2**: The host waits for a FIN request from the remote host.
- **CLOSE\_WAIT**: This means the server has received a FIN packet from the client and is waiting to end the session.
- **CLOSING**: After the FIN packet has been sent, the host begins closing the connection and waits for a corresponding acknowledgment, in order to fully close the session.
- **LAST\_ACK**: A final acknowledgment after sending the FIN packet is made to ensure the remote host has received the termination request.
- **TIME\_WAIT**: After sending a termination request, this state waits to ensure that the remote host has received the request to end the conversation.
- **CLOSED**: This is not a state at all; it represents a closed connection.

For a connection-oriented session where it is important to get all the parts of the communication stream, TCP is the Transport layer protocol of choice. However, when speed in data transport is required, UDP is the better choice. UDP is a connectionless protocol with only four field values, as we will learn next.

### Ensuring a timely delivery with UDP

UDP is a connectionless and lightweight Transport layer protocol that has a four-field header. UDP doesn't have any handshake or connection process, ordering or reliability services, and there's no teardown. As a lightweight protocol, it's ideal where speed is an issue and is used with time-sensitive applications, such as the following:

- **Dynamic Host Configuration Protocol (DHCP)**
- **Routing Information Protocol (RIP)**
- **Voice over Internet Protocol (VoIP)**
- **Trivial File Transfer Protocol (TFTP).**

Whether TCP or UDP is in use, the Transport layer is a critical component to ensure data transport. During the encapsulation process, the data begins to transform, and the PDU is now a segment. At this point, the Transport layer requires a port number (or address) that is associated with an application or process that is in use. This is discussed in the following section.

### Providing port addressing

At the Transport layer, we add a port address, which is used to identify a specific application or process. Port numbers fall into three main groups:

- **Well-known** ports range between 1 and 1,023 and include protocols such as HTTP, DNS, and SMTP.
- **Registered** ports range between 1,024 and 49,151 and are assigned and used for specific services such as gaming applications, OpenVPN, and IPsec.
- **Dynamic, private, or ephemeral** ports are in the range of 49,152–65,535 and are not assigned to any specific application. They are used temporarily during a session, generally by the client.

When the Transport layer header is applied to the data, a source and destination port are added. The type of port used depends on whether the packet is coming from the client or the server:

- If a **client** sends a packet, then the source port will be (in most cases) a randomly assigned dynamic or ephemeral port that is used. So, when the server delivers a packet to the host, it uses that port to deliver the data.
- If a **server** sends a packet, then the source port will be either a well-known port or a registered port.

The Transport layer provides inter-host communication between endpoints. The following outlines a summary of the Transport layer:

Layer	Purpose	Protocols	PDU
Transport	Transports the data	TCP/UDP	Segment

Table 8.6 – Summary of the Transport layer

After the Transport layer, the next layer is the Network layer. As we'll see in the next section, this layer is all about getting the data to the correct network.

## Explaining the Network layer

The **Network** layer (or layer 3) has two key roles: addressing and routing data. This layer provides addressing using a logical IP address. In addition, the **Network** layer determines the best logical path to take for packets that travel through other networks, so they can get to their destination. It does this by communicating with other devices during the routing process.

In addition to data forwarding, the Network layer communicates errors in transmission. In order to achieve this, the Network layer has a few key protocols, as we will see in the following section.

## Distinguishing the protocols and the PDU

The Network layer is responsible for addressing and routing. There are three main protocols in this layer: IP, **Address Resolution Protocol (ARP)**, and **Internet Control Message Protocol (ICMP)**. Let's start with IP.

## Routing packets with the IP

IP is a best-effort, connectionless protocol that routes packets from the source to the destination using a logical IP address.

Over the years, many of the original protocols in the TCP/IP suite have had minor changes, updates, and modifications. However, IP had to make a major change, which was mainly due to a lack of address space. As a result, there are two versions of IP: IPv4 and IPv6. The following outlines a brief comparison of the two:

- **IPv4** has a 32-bit address space. The use of private IP addresses has extended IPv4's lifespan on a LAN, but there is a slow migration to IPv6.
- **IPv6** has a 128-bit address space and offers enhancements to the protocol in general, such as simplified network configuration and more efficient routing.

Next, we will look at ARP, which resolves an IP address to a MAC address.

### Resolving addresses with ARP

IP routes traffic through networks to its destination LAN. When a packet arrives on the LAN, it no longer needs an IP address. It requires a physical or MAC address to go to its destination. ARP issues a broadcast to resolve an IPv4 address to a MAC address on a **local area network (LAN)** so that the frame can be delivered.

#### Note

ARP is an unusual protocol because it appears between layer three and layer two of the OSI model. ARP resolves an IP address (the Network layer) to a MAC address (the Data Link layer) address. However, many consider it to be a layer three protocol.

In addition to IP and ARP, we also need ICMP to help report any problems that might have occurred during data transport. This is discussed in the following segment.

### Conveying messages with ICMP

ICMP is another critical network protocol. However, ICMP does not exchange or transport data. Its primary role is error reporting. Because IP is a best-effort, unreliable protocol, ICMP must be implemented by every IP module, as outlined in the original **Request for Comment (RFC)**, which can be found at <https://tools.ietf.org/html/rfc792>. ICMP reports on any issues encountered during transit such as the network being unreachable and the host being unreachable. Because there are two IP versions, there are two versions of ICMP:

- **IPv4** uses ICMP.
- **IPv6** uses ICMPv6.

During the encapsulation process, the PDU at the Network layer is a packet. The Network layer is responsible for routing and addressing data. One key element is an IP (or logical) address, as described next.

## Supplying an IP address for the packet

In this layer, the IP header will hold the source and destination addresses in either IPv4 format or IPv6 format. Both are referred to as logical addresses and are represented as follows:

- An IPv4 address has 32 bits, which Wireshark will display using dotted decimal notation.
- An IPv6 address has 128 bits, which Wireshark will display using hexadecimal numbers, separated by colons.

In the previous section, we discussed two other protocols: ARP and ICMP. Let's discuss both of these in relation to the need for addressing.

We know that IP uses a header that houses an IP address. However, ARP and ICMP are both unique, as outlined here:

- **ARP** does not have an IP header. ARP is a service protocol that resolves an IPv4 address to a MAC address.
- **ICMP** is the partner protocol to IP, and it reports on matters encountered during transit, such as the network being unreachable and the host being unreachable. ICMP itself does not need an IP address, as it is encapsulated in an IP header, as shown in the following screenshot of an ICMP echo request:

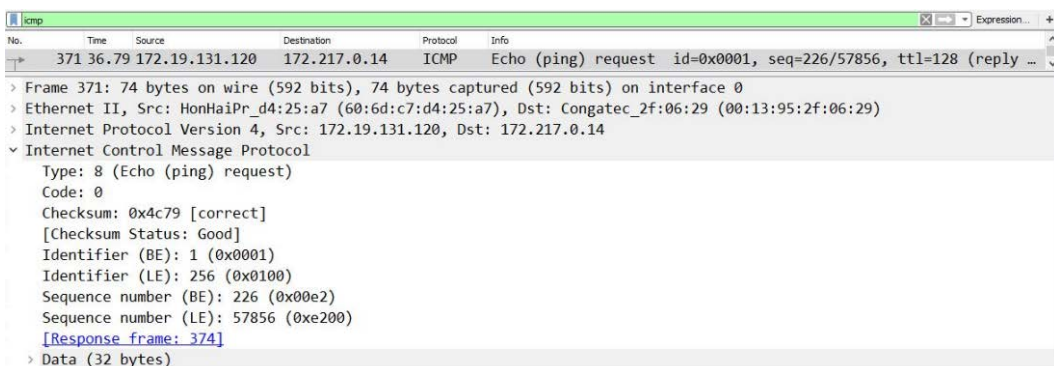


Figure 8.5 – An ICMP echo request

By now, you should have understood that the Network layer allows hosts on separate networks to communicate with one another by providing logical addressing and path determination.

The following table outlines a summary of the Network layer:

Layer	Purpose	Protocols	PDU
Network	Addressing and routing	IP and ICMP	Packet

Table 8.7 – Summary of the Network layer

As data is encapsulated and passed down the OSI model, the next step is the **Data Link** layer, where a key role is proper frame formation so that the frame can travel on the LAN. Let's take a look.

## Examining the Data Link layer

The **Data Link** layer (or layer 2) is primarily concerned with proper frame formation and prepares the data before it is sent out on the network. Within the **Data Link** layer, there are several protocols that are responsible for properly formatting the data so that it can successfully traverse on the destination network. Let's focus on a few key **Data Link** layer protocols next.

### Investigating protocols and the PDU

The Data Link layer is the final stop as data travels down the OSI model, as this layer adds a frame header and trailer to ready the frame for the network. Protocols used in this layer include the following:

- **Ethernet II** is the most widely used Ethernet technology today. It establishes connections on a LAN using the physical (or MAC) address.
- **High-Level Data Link Control (HDLC)** uses frames to deliver data from point to point.

At this layer, the PDU is a frame. Each frame requires an address, which we will outline next.

## Describing the Data Link layer address

On a LAN, the **Data Link** layer uses the MAC address of the destination machine rather than the IP address. The Data Link layer has a frame header that contains the source and destination MAC address, which is also referred to as a physical address. The trailer, or frame check sequence, holds a value called a **Cyclic Redundancy Check (CRC)**, which is used for error detection on a network.

The following table provides a summary of the Data Link layer:

Layer	Purpose	Protocols	PDU
Data Link	Frame formation	Ethernet and HDLC	Frame

Table 8.8 – Summary of the Data Link layer

The Data Link layer ensures proper frame formation and link access, along with error detection, while traveling across the network media. Data then travels to the Physical layer, as we will see in the following section.

## Traveling over the Physical layer

The **Physical** layer (or layer 1) transmits data over media in a stream of bits.

Once data is formatted into a frame, the **Network Interface Card (NIC)** sends it on to the network media in a stream of bits.

## Exemplifying protocols

In the Physical layer, there are several different protocols used to transmit data across the network media:

- **Digital Subscriber Line (DSL)** provides broadband to residences and businesses using a phone line.
- **Integrated Services Digital Network (ISDN)** transmits voice, video, and data using the **Public Switched Telephone Network (PSTN)**. ISDN is primarily used in the broadcasting industry.
- **IEEE 802.3**—the Ethernet Physical layer—defines the transmission properties according to the media type, such as fast Ethernet and **Gigabit (GB)** Ethernet.

The Physical layer is where binary transmission takes place across the network media. Let's review the PDU for this layer.



## Describing the PDU

At this layer, the frame formation is complete and ready to travel over the media. The PDU is in the most basic form, which is bits. The method of transmission will depend on the medium.

Network media includes the following:

- **Copper cable** using **Unshielded Twisted Pairs (UTP)**, **Shielded Twisted Pairs (STP)**, or coaxial; transmits with pulses of electricity
- **Fiber** using a multimode or single-mode cable; transmits with pulses of light
- **Wireless** using 802.11 specifications; transmits over radio waves

The following table provides a summary of the Physical layer:

Layer	Purpose	Protocols	PDU
Physical	Binary transmission	DSL, ISDN, and 802.3	Bits

Table 8.9 – Summary of the Physical layer

Prior to traveling through the network, the data must be in the correct format. The next section explores the encapsulation process, which adds headers and addresses and readies the data for transport through the media.

## Exploring the encapsulation process

Now that we have gained an understanding of the layers, let's look at how each of the layers works together during the encapsulation process to create a frame.

During frame formation, the process begins with data. As the data moves down the layers, a header is added, one by one, until the frame is complete. Each frame has the following components:

- Data and an appropriate Application layer header (if applicable)
- A segment header
- A packet (or IP) header
- A frame header

We'll start with the data portion of the frame.

## Viewing the data

In most cases, when frame formation begins and encapsulation takes place, we start with the data, as shown here:

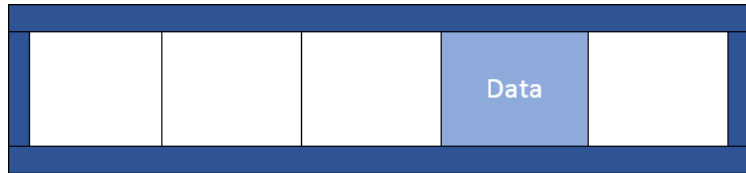


Figure 8.6 – The encapsulation process—data

The data might be any of the following:

- An HTTP GET request
- A DNS request to resolve a hostname to an IP address
- A DHCP broadcast to request a dynamically assigned IP address

The data then continues its journey to becoming a segment.

## Identifying the segment

The next thing that happens is that the data will (in most cases) become a segment that will use either a TCP header or a UDP header:

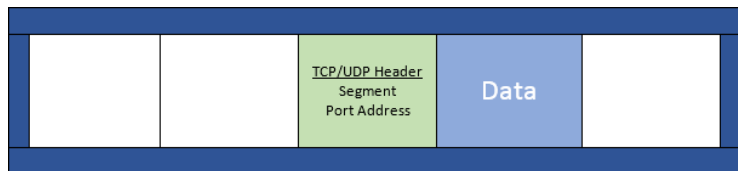


Figure 8.7 – The encapsulation process—segment

The segment holds a source and destination port address, as shown in *Figure 8.7*. The next stop in the encapsulation process is to add an IP header in order for it to become a packet. We will discuss this next.

## Characterizing the packet

As data is encapsulated, we now have data, along with a segment that holds either a TCP or UDP port address. The next part of encapsulation is to create a packet by adding the source and destination IPv4 or IPv6 address to the IP header, as shown here:

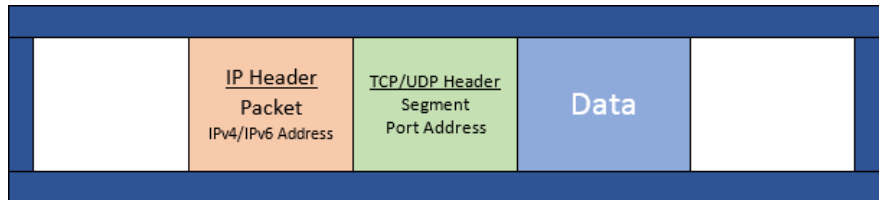


Figure 8.8 – The encapsulation process—packet

The final part of the encapsulation process is the addition of the frame header, which is demonstrated in the next section.

## Forming the frame

The last stop on the journey of creating a frame is the addition of the header. Within the frame, we have the following:

- The data, such as an HTTP request or DNS reply
- A Transport layer header (or segment)
- A Network layer header (or packet)

At this point, we complete the frame by adding the source and destination MAC addresses, as shown here:

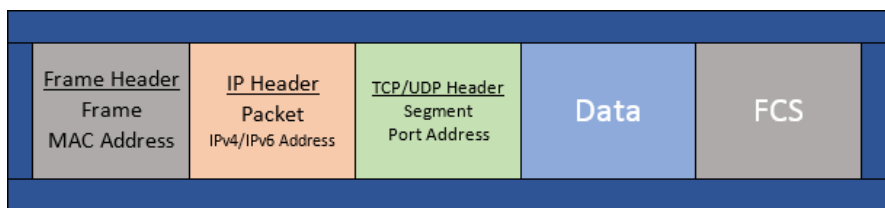


Figure 8.9 – The encapsulation process—frame

With a frame, not only do we have a header, but we also have a trailer, which is called the **Frame Check Sequence (FCS)**. The FCS holds a value called a cyclic redundancy check, which is used for error detection on the network and is checked while traveling along on its journey.

**Note**

The FCS is used for error *detection*, not error *correction*.

The next section covers how the frame formation looks when Wireshark captures the traffic and presents it to the user.

## Demonstrating frame formation in Wireshark

Once you understand encapsulation and frame formation, you will be able to learn how Wireshark represents frame formation, as shown in the following screenshot:

```
Frame 4371: 401 bytes on wire (3208 bits), 401 bytes captured (3208 bits) on interface 0
Ethernet II, Src: HonHaiPr_d4:25:a7 (60:6d:c7:d4:25:a7), Dst: Viasat_ad:3b:50 (00:a0:bc:ad:3b:50)
Internet Protocol Version 4, Src: 172.19.0.42, Dst: 172.217.2.1
Transmission Control Protocol, Src Port: 53770, Dst Port: 80, Seq: 1, Ack: 1, Len: 347
Hypertext Transfer Protocol
```

Figure 8.10 – Frame formation in Wireshark

**Note**

Not all frames contain data; however, this one does, so it's a good example of a fully encapsulated frame.

When looking at a single frame, you will see at the top of *Figure 8.10*, the Frame 4371 line, which is the *metadata* about that single frame that summarizes the contents of the frame. The metadata for this frame includes information such as 401 bytes on wire and 401 bytes captured.

After the frame metadata, you will have the following:

- **Frame:** The frame header shows Ethernet II, and after that are the source and destination MAC addresses.
- **Packet:** The IP header represents the Network layer, which holds the source and destination IP addresses.
- **Segment:** The TCP header represents the Transport layer, which holds the source and destination port addresses.
- **Data:** The HTTP header represents the Application layer. In this case, it is a web request.

This is an example of how Wireshark displays the encapsulation process and how it relates to the OSI model.

Now that you have learned about the encapsulation process and frame formation in Wireshark, let's take a look at the NIC and see the OSI model in action on your own system.

## Examining the network bindings

On your own laptop or desktop, you can easily see the OSI model in action. If you check your network connections and then select the properties of your network interface card, as shown in the following screenshot, you can see how the layers in the OSI model are represented:

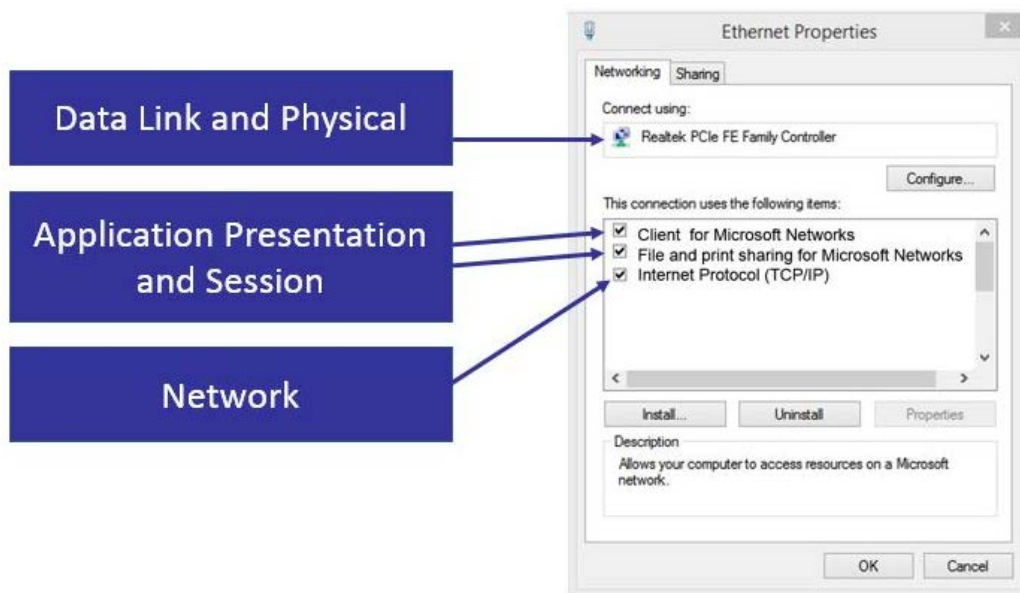


Figure 8.11 – Network bindings

The following represents the various layers and how they are represented in **Ethernet Properties**:

- The Data Link and Physical layers are represented in the NIC.
- The Application, Presentation, and Session layers are represented in **Client for Microsoft Networks** and **File and print sharing for Microsoft Networks**.
- The Network layers are shown as **Internet Protocol (TCP/IP)**.

By now, you should be able to recognize each of the layers of the OSI model, along with how each layer is represented in Wireshark.

## Summary

In this chapter, we took a closer look at an important concept, the OSI model and the encapsulation process. The OSI model is a framework that serves many purposes. The model provides a common framework for developers and a method to help students understand what process occurs at each layer. In addition to this, understanding each of the layers, the PDU, and the addressing will help you to better comprehend the process in Wireshark and improve your data analysis skills.

By now, you should have a better understanding of the role, purpose, protocols, and PDU of each layer. We explored the encapsulation process and took a look at frame formation as it is seen in Wireshark. To help you understand how your system uses the OSI model, we looked at how the model is represented in the network bindings.

In the next chapter, we'll take a closer look at how to decode the two main Transport layers: TCP and UDP. We'll begin by reviewing the purpose of the Transport layer and then discuss TCP and examine the 11-field header format in Wireshark. Then, we'll go through an overview of the purpose of UDP and examine the four-field header and the significance of each field value.

## Questions

Now it's time to check your knowledge. Select the best responses, and then check your answers with those listed in the *Assessment* appendix:

1. The \_\_\_\_\_ layer, or layer 5, is responsible for setting up, maintaining, and tearing down a session.
  - A. Transport
  - B. Application
  - C. Session
  - D. Presentation

2. The \_\_\_\_\_ layer, or layer 4, is responsible for transporting the data, either using a connectionless protocol or a connection-oriented protocol.
  - A. Transport
  - B. Application
  - C. Session
  - D. Presentation
3. The \_\_\_\_\_ layer, or layer 6, is responsible for proper data formatting along with optional compression and encryption.
  - A. Transport
  - B. Application
  - C. Session
  - D. Presentation
4. TCP port 3344 is in the range of the \_\_\_\_\_ ports.
  - A. Ephemeral
  - B. Well-known
  - C. Registered
  - D. Secure
5. The PDU at the transport layer is \_\_\_\_\_.
  - A. Data
  - B. Frame
  - C. Packet
  - D. Segment
6. ARP does not have a(n) \_\_\_\_\_ header because it is a service protocol that resolves IPv4 addresses to MAC addresses.
  - A. Data Link
  - B. Frame
  - C. IP
  - D. Segment

7. On a LAN, the Data Link layer uses the \_\_\_\_\_ address of the destination machine rather than the IP address.
- A. MAC
  - B. Ephemeral
  - C. Packet
  - D. Segment





# Part 3

# The Internet Suite

# TCP/IP

In this section, we'll study the internet suite and examine the protocols that move data, TCP, UDP, IP, and ICMP. We'll take a closer look at how TCP manages a connection and dissect the handshake and teardown process. You'll see the differences between IPv4 and IPv6 and discover why ICMP is an integral part of the data delivery process.

The following chapters will be covered under this section:

- *Chapter 9, Decoding TCP and UDP*
- *Chapter 10, Managing TCP Connections*
- *Chapter 11, Analyzing IPv4 and IPv6*
- *Chapter 12, Discovering ICMP*



# 9

# Decoding TCP and UDP

Since being standardized in the early 1980s, the **Transmission Control Protocol/Internet Protocol (TCP/IP)** suite has defined how data is addressed, packetized, transmitted, and routed. Over the years, modifications have been made to the TCP/IP suite to provide more efficiency in today's changing network. This chapter will focus on the TCP portion of the suite, that is, the transport layer (or Layer 4) of the **Open Systems Interconnection (OSI)** model. Layer 4 has several protocols to transport data; however, we will focus on the most widely used transport layer protocols, TCP and UDP.

In this chapter, we'll begin by reviewing the role and purpose of the transport layer. We'll then take a closer look at TCP, a connection-oriented protocol, and its uses. So that you can easily analyze TCP, we'll examine the header format and field values in detail, such as sequence number, offset, **window size (WS)**, and TCP flags. We'll summarize with a review of the **User Datagram Protocol (UDP)**, and discuss common uses for this lightweight, connectionless protocol. We'll finish by examining UDP's streamlined four-field header.

This chapter will cover the following:

- Reviewing the transport layer
- Describing TCP

- Examining the 11-field TCP header
- Understanding UDP
- Discovering the four-field UDP header

## Reviewing the transport layer

The transport layer of the OSI model is responsible for providing end-to-end data transport, by either using a connectionless or connection-oriented protocol across an IP network. The transport protocol used will depend on the application.

In addition to TCP and UDP, there are several protocols in this layer, including the following:

- **Reliable data protocol (RDP)**: Used to transfer data in a connection-oriented manner
- **Stream control transmission protocol (SCTP)**: Provides the reliable transmission of data streams that have more than one IP address

While there are other transport layer protocols, the two predominant protocols are TCP and UDP, as shown in the following figure:

### OSI Model

Layer	Name	Role	Protocols	PDU	Address
7	Application	Initiate contact with the network	HTTP, FTP, SMTP	Data	
6	Presentation	Formats data, optional compression and encryption		Data	
5	Session	Initiates, maintains and tear down session		Data	
4	Transport	Transports data	TCP, UDP	Segment	Port
3	Network	Addressing, routing	IP, ICMP	Packet	IP
2	Data Link	Frame formation	Ethernet II	Frame	MAC
1	Physical	Data is transmitted on the media		Bits	

Figure 9.1 – The OSI model—Transport layer

UDP is connectionless and is used when data transport needs to be fast. UDP has a lightweight four-field header that is always 8 bytes in length.

**Note**

Unlike TCP, UDP currently does not have any header options. However, because of the changing nature of the internet, there has been an active discussion on possibly including options for UDP. You can review the draft called *Transport Options for UDP*, here: <https://www.ietf.org/archive/id/draft-ietf-tsvwg-udp-options-13.txt>.

In contrast to UDP, TCP is a connection-oriented protocol. It has an 11-field header that has the ability to include options. In the next section, we'll learn how impressive this protocol is in its ability to ensure the complete delivery of data while monitoring for congestion and providing flow control.

## Describing TCP

TCP is a connection-oriented protocol that has end-to-end reliability. These two significant features are described as follows:

- **Connection-oriented** means that both endpoints must set up a connection before any data is transferred. To begin a session, TCP starts with a (three-way) handshake and ends with a series of **finis (FIN)** packets to close the session.
- **End-to-end reliability** means that the data is sequenced and acknowledged during the data transfer. All data is carefully monitored, and if there are any gaps in transmission, TCP will request any missing packets.

Once two hosts establish a connection, TCP monitors the session to ensure the data transfer is complete, without causing undue strain on the host or the network. In the next section, we'll summarize what happens during a TCP connection.

## Establishing and maintaining a connection

When using TCP, prior to any exchange of data, the session must begin with a three-way handshake between hosts. The handshake establishes the parameters of the conversation. Both client and server must agree on all parameters before commencing communication.

In many cases, in addition to the TCP header, there are header options that outline and further define the parameters of the conversation.

Any TCP options will be found in the first two packets of the three-way handshake. For example, two common options are as follows:

- **Window scaling:** A value that expands the *stated* WS by providing a multiplier that more accurately reflects the *true* WS.
- **Selective Acknowledgements (SACK):** When SACK is enabled, the receiver will notify the sender only if there are any missing packets.

**Note**

TCP options will be covered in detail in *Chapter 10, Managing TCP Connections*.

Once a connection is established between two hosts, the operating system for both local and foreign (or remote) hosts will create a socket, which is an IP address and a port.

To see all your active TCP connections on a Windows machine, open a command-line prompt and run `netstat -anp tcp`. This will show your active connections.

Across the top, you will see the following headers:

Proto	Local Address	Foreign Address	State
-------	---------------	-----------------	-------

These are defined as follows:

- Proto (protocol)
- Local Address (local IP address and port)
- Foreign Address (remote IP address and port)
- State of the connection

After running the `netstat` command, my output was as follows:

TCP	10.0.0.148:49559	17.249.124.141:5223	ESTABLISHED
TCP	10.0.0.148:49768	34.212.110.138:443	ESTABLISHED
TCP	10.0.0.148:62310	13.89.217.116:443	ESTABLISHED
TCP	10.0.0.148:62789	23.55.20.137:443	CLOSE_WAIT
TCP	10.0.0.148:62790	204.13.192.141:80	CLOSE_WAIT

Figure 9.2 – Netstat showing TCP connection status

In *Figure 9.2*, you see a *local* IP address and port, along with a *foreign* (or remote) IP address and port. For example, in the first line, we see the following:

- Proto: TCP
- Local Address: 10.0.0.148:49559
- Foreign Address: 172.249.124.141:5223
- State: ESTABLISHED (or actively communicating)

During the conversation, TCP acknowledges all the packets received to ensure the complete delivery of the data. Every time TCP receives data, the receiving host sends an **acknowledgment (ACK)** packet back to the sender, notifying the sender of what data was received.

Once the conversation is over, TCP ends the session with an exchange of FIN packets.

Next, let's see how this powerful protocol has methods to assist in controlling the data flow and minimizing network congestion.

## Managing the flow

During a connection, TCP actively monitors the connection, and both sender and receiver constantly communicate with each other. During the data exchange, TCP will manage the data to both the host and the network as follows:

- **Flow control** is an end-to-end control method using the WS, so the sender doesn't transmit too much data and *overwhelm the host*.
- **Congestion control** prevents a node from sending too much data and *overwhelming the network*.

When providing congestion control, there are two state variables that are calculated during data transmission:

- **Congestion Window (Cwnd)**: The sender-side limit defines the amount of data a host can send *before receiving* an acknowledgment.
- **Receiver Window (Rwnd)**: The receiver-side limit defines the amount of data a host can receive.



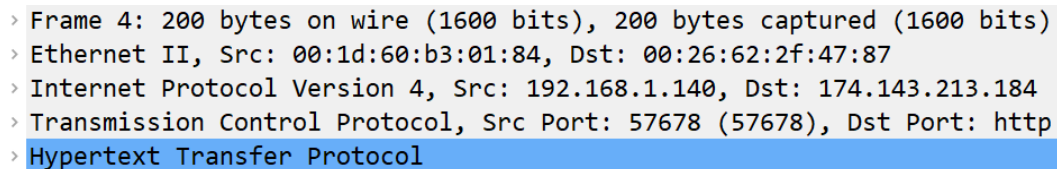
The two variables work together in a TCP connection to regulate the flow of data, minimize congestion, and improve network performance.

It's hard to believe, but there is a great deal of detail in one single frame, and each frame can contain many components. Components can include the various headers, field values within the headers, and optional data. In the next section, we'll look at all the information that's found in a single TCP frame.

## Exploring a single TCP frame

For a deep dive into the TCP header, go to <https://www.cloudshark.org/captures/0012f52602a3>. Download and open the HTTP.cap packet capture file in Wireshark.

To follow along, select Frame 4 and focus on the packet details pane, as shown in the following screenshot:



```
> Frame 4: 200 bytes on wire (1600 bits), 200 bytes captured (1600 bits)
> Ethernet II, Src: 00:1d:60:b3:01:84, Dst: 00:26:62:2f:47:87
> Internet Protocol Version 4, Src: 192.168.1.140, Dst: 174.143.213.184
> Transmission Control Protocol, Src Port: 57678 (57678), Dst Port: http
> Hypertext Transfer Protocol
```

Figure 9.3 – The packet details pane for Frame 4

Each header has a summary, followed by the details of the header. To see the protocol summary, place your cursor on a protocol, right-click, and then select **Protocol Preferences**, as shown here:

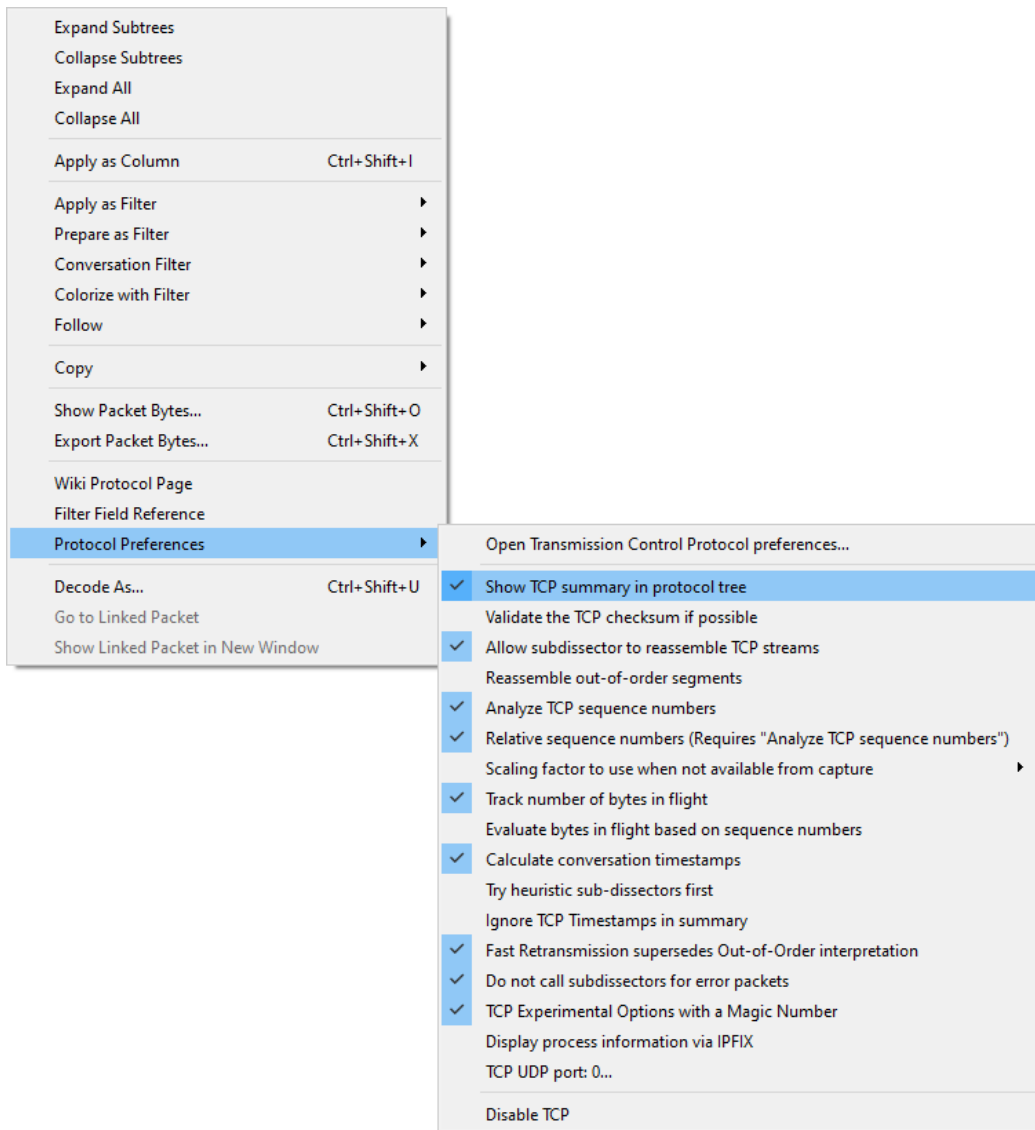


Figure 9.4 – Protocol preferences for TCP

Once there, select **Show TCP summary in protocol tree**, as shown. In addition, you can expand the header by clicking on the arrow (or caret, >) on the right-hand side to see the details.

Let's break down each element of Frame 4, starting with the frame details.

## Describing the frame details

Starting from the top of *Figure 9.3*, we see Frame 4. Expand the frame identifier by clicking on the arrow on the left-hand side to see the details as follows:

```

v Frame 4: 200 bytes on wire (1600 bits), 200 bytes captured (1600 bits)
  Encapsulation type: Ethernet (1)
  Arrival Time: Mar 1, 2011 15:45:13.313889000 Eastern Standard Time
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1299012313.313889000 seconds
  [Time delta from previous captured frame: 0.000112000 seconds]
  [Time delta from previous displayed frame: 0.000112000 seconds]
  [Time since reference or first frame: 0.047068000 seconds]
  Frame Number: 4
  Frame Length: 200 bytes (1600 bits)
  Capture Length: 200 bytes (1600 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertype:ip:tcp:http]
  [Coloring Rule Name: HTTP]
  [Coloring Rule String: http || tcp.port == 80 || http2]

```

Figure 9.5 – Frame metadata on a single frame

### Note

The identifier Frame is not a protocol, it is a list of values generated by Wireshark that describes information (or metadata) about a single frame.

As shown in *Figure 9.5*, we see the metadata about Frame 4, which summarizes the contents of the frame, such as the frame length and any coloring rules.

The next identifier begins the true frame encapsulation, as we see the Ethernet II header.

## Expanding the Ethernet II header

The (true) frame header is the Ethernet II frame, which is a common frame format found on a **Local Area Network (LAN)**. The frame header follows the metadata summary and provides information about the source and destination **Media Access Control (MAC)** address, as shown in the following screenshot:

```

~ Ethernet II, Src: 00:1d:60:b3:01:84, Dst: 00:26:62:2f:47:87
  > Destination: 00:26:62:2f:47:87
  > Source: 00:1d:60:b3:01:84
    Type: IPv4 (0x0800)

```

Figure 9.6 – Frame header

Within the header, we see the following in the last line of the frame: `Type: IPv4 (0X0800)`. This means the next header to follow is an IPv4 header, which we'll investigate next.

## Viewing the IP header

The Internet Protocol Version 4 header begins with a summary that includes the source and destination IP address, followed by the IPv4 field values.

### Note

The IPv4 header will be covered in detail in *Chapter 11, Analyzing IPv4 and IPv6*.

The IPv4 header for Frame 4 is as shown in the following screenshot:

```

~ Internet Protocol Version 4, Src: 192.168.1.140, Dst: 174.143.213.184
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 186
    Identification: 0xcb5d (52061)
  > Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0x2864 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.140
    Destination Address: 174.143.213.184

```

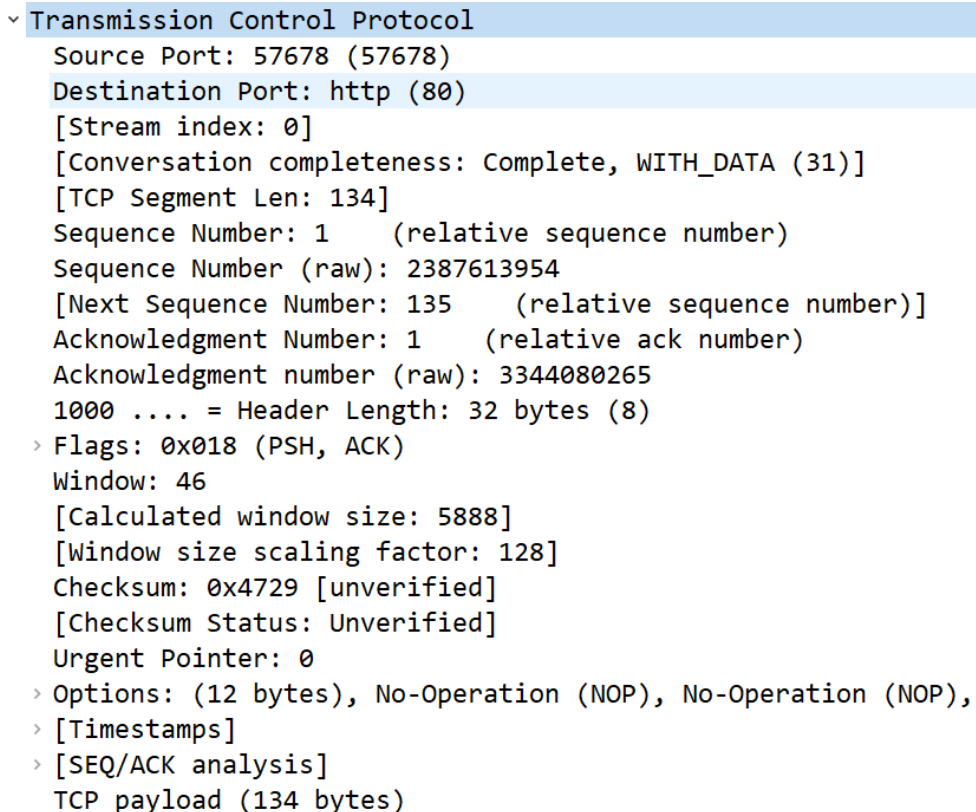
Figure 9.7 – IPv4 header

Similar to the frame header, we see the following within the frame: `Protocol: TCP (6)`. This means the next header to follow is a TCP header.

## Navigating the TCP header

The TCP header begins with the identifier `Transmission Control Protocol` and lists the source and destination ports, sequence, and acknowledgment numbers. In addition, you will also see Wireshark generated data, shown in [ ], such as [Stream index: 0].

The TCP header for Frame 4 is as shown in the following screenshot:



```

v Transmission Control Protocol
  Source Port: 57678 (57678)
  Destination Port: http (80)
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 134]
  Sequence Number: 1      (relative sequence number)
  Sequence Number (raw): 2387613954
  [Next Sequence Number: 135      (relative sequence number)]
  Acknowledgment Number: 1      (relative ack number)
  Acknowledgment number (raw): 3344080265
  1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x018 (PSH, ACK)
  Window: 46
  [Calculated window size: 5888]
  [Window size scaling factor: 128]
  Checksum: 0x4729 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP),
  > [Timestamps]
  > [SEQ/ACK analysis]
  TCP payload (134 bytes)
```

Figure 9.8 – TCP header

Here we see the details of the TCP header. In this case, I have disabled the **Show TCP summary in protocol tree** protocol preference, so the entire header was able to fit in the page.

The last component of Frame 4 is the **Hypertext Transfer Protocol (HTTP)** header.

## Checking out the HTTP header

Because the client is requesting data from a web server, the application in use is HTTP. In this case, we see the HTTP header, as shown here:

```

v Hypertext Transfer Protocol
  > GET /images/layout/logo.png HTTP/1.0\r\n
    User-Agent: Wget/1.12 (linux-gnu)\r\n
    Accept: */*\r\n
    Host: packetlife.net\r\n
    Connection: Keep-Alive\r\n
    \r\n
    [Full request URI: http://packetlife.net/images/layout/logo.png]
    [HTTP request 1/1]
    [Response in frame: 36]

```

Figure 9.9 – HTTP header

### Note

The HTTP protocol will be covered in detail in *Chapter 15, Decoding HTTP*.

Now that we have covered the details that are found in a single frame, let's examine the TCP header and each of the field values in detail.

## Examining the 11-field TCP header

TCP has an 11-field header. The fields hold the values that keep track of the conversation, as shown in the following diagram:

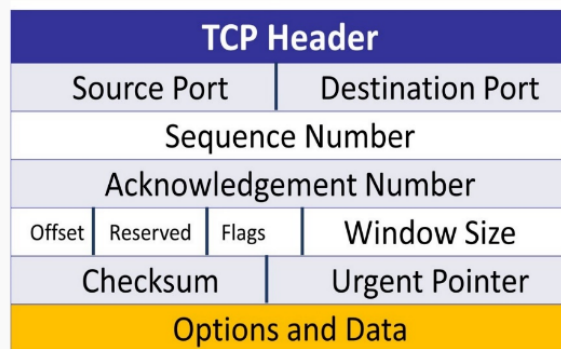


Figure 9.10 – The TCP header

TCP uses the field values to monitor the communication. Once all bytes have been received, TCP will indicate that the end device has successfully received all of the data. If there is trouble during the data transport, TCP will alert the other host of any missing segments.

In this section, we'll take a look at each of the header fields so that you have a better understanding of how TCP is able to provide reliable communication between hosts. I'll list the field and the size in the following manner, [Field: Size], when appropriate, so that you understand the values that comprise a TCP header.

Starting at the top of the TCP header in Frame 4 of HTTP.cap, we can see the label `Transmission Control Protocol`, as shown in *Figure 9.8*. Below the label, you can see the TCP header fields.

When looking at the TCP header contents, many times, any information that Wireshark generates will be shown in brackets. This generated content will help you better understand the details of the header, such as [Timestamps] and [SEQ/ACK analysis], as shown near the bottom of *Figure 9.8*.

Let's step through the first few header fields, starting with the ports and TCP segment length.

## Exploring TCP ports

In Wireshark, you can resolve physical, network, and transport layer addresses. The packet capture HTTP.cap uses the transport layer name resolution. As a result, whenever a well-known or registered port is used, Wireshark will identify the application associated with the port number. For example, HTTP uses port 80, and when port 80 is used, it will be identified as HTTP.

The following lists the port numbers, along with the generated information Wireshark provides, such as `Stream Index` and `Segment Length`:

- **Source port 16-bit:** Frame 4 is a request from the client to the server. By indicating a port number, that tells the server, *When you deliver the data, use this port*. In this case, the value is `Source Port: 57678 (57678)`, which is not associated with any application; it is an ephemeral (or temporarily assigned) port that is used in this connection. As a result, you will not see a protocol listed before the port number.
- **Destination port 16-bit:** Frame 4 is a request from the client to the server, which is most likely a web server, as the value is `Destination Port: http (80)`.

- **Stream index:** This value is shown in brackets, as Wireshark calculates this to keep track of the streams, where each stream is a communication between two endpoints. In Frame 4, we can see [Stream index: 0], which means this is the first stream in this capture. This value is a useful tool when doing an analysis, as you can easily right-click on a frame and select **Follow**, and then select a stream (such as TCP or HTTP) as shown in the following screenshot:

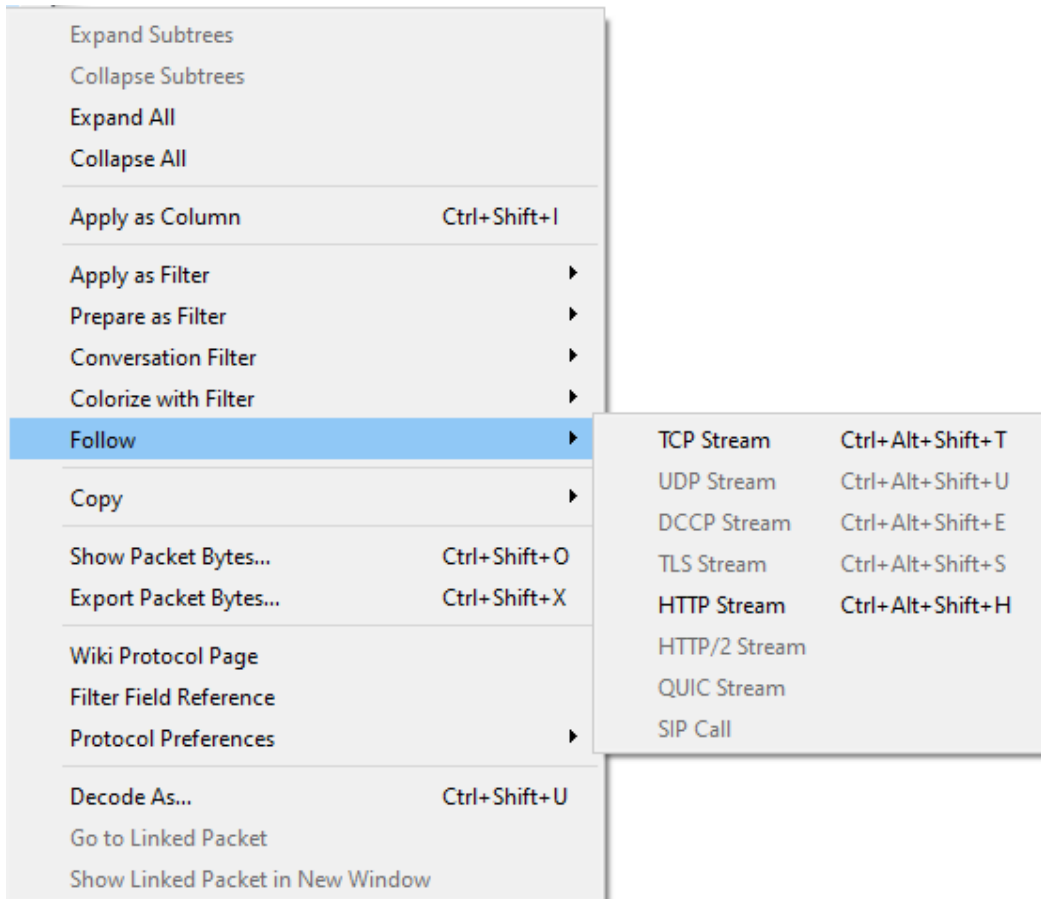


Figure 9.11 – Following the stream

- **Conversation completeness:** This generated value indicates whether or not the conversation is complete. In this case, we see `Complete, WITH_DATA (31) ]`. This means the captured conversation is complete, as it has all the elements of a complete conversation, such as the handshake, data transfer, and FIN packet exchange.



- **TCP segment length:** In the transport layer, the **Protocol Data Unit (PDU)** is a segment. The segment length is the value of the TCP payload, which is the data that follows the TCP header, and any options. Wireshark calculates this value as shown in brackets. In Frame 4, we can see [TCP Segment Len: 134], which is the same value as TCP payload (134 bytes), as shown in the last line of *Figure 9.8*.

Next, we'll take a look at the fields that keep track of the data that's sent and received during data transmission.

## Sequencing bytes

Because TCP is a connection-oriented protocol, the operating system keeps track of every byte (or octet) of data that is received. Each byte is sequenced, and once received, is acknowledged. The three-way handshake starts the sequencing process.

Before taking a look at the sequence numbers, let's discuss how they are calculated.

## Synchronizing the sequence numbers

Prior to exchanging data, the client and server must synchronize the sequence numbers. This is achieved by using the **Synchronization (SYN)** flag in the TCP header. The SYN packets are found in the first two packets of the three-way handshake and are responsible for synchronizing the sequence numbers used during the connection.

For example, as shown in the following screenshot, a client sends a SYN packet to the server with a **Sequence (SEQ)** number of 100:

Client	<SYN><SEQ=100> -->	Server
Client	<-- <SEQ=300><ACK=101><SYN,ACK>	Server
Client	<SEQ=101><ACK=301><ACK>-->	Server

Figure 9.12 – The three-way handshake

**Note**

The sequence number of 100 was randomly selected for this example. In an actual TCP connection, the OS will generate a random sequence number to start the process.

The server responds by sending a **Synchronization Acknowledgement (SYN, ACK)** with a sequence number of 300 and an ACK of 101. The client sends a final ACK with a sequence number of 301 and an ACK of 101. The sequence numbers are now synchronized.

**Note**

You can see an example of the three-way handshake in the first three packets of `HTTP.cap`.

Next, we'll evaluate how Wireshark can represent the sequence numbers when analyzing data.

## Understanding the sequence numbers

When working with a packet capture, you have a choice as to how you would like your sequence numbers represented. Wireshark can generate *relative* sequence numbers, mainly because the *actual* (or absolute) sequence number is very large. The relative sequence number is easy to understand and represents a value *in relation* to a specific conversation.

To switch, right-click anywhere in the TCP header, select **Protocol Preferences**, and then select **Relative Sequence Numbers**.

Keep in mind, in order to display relative sequence numbers, you must also select **Analyze TCP sequence numbers**, as shown in the following screenshot:

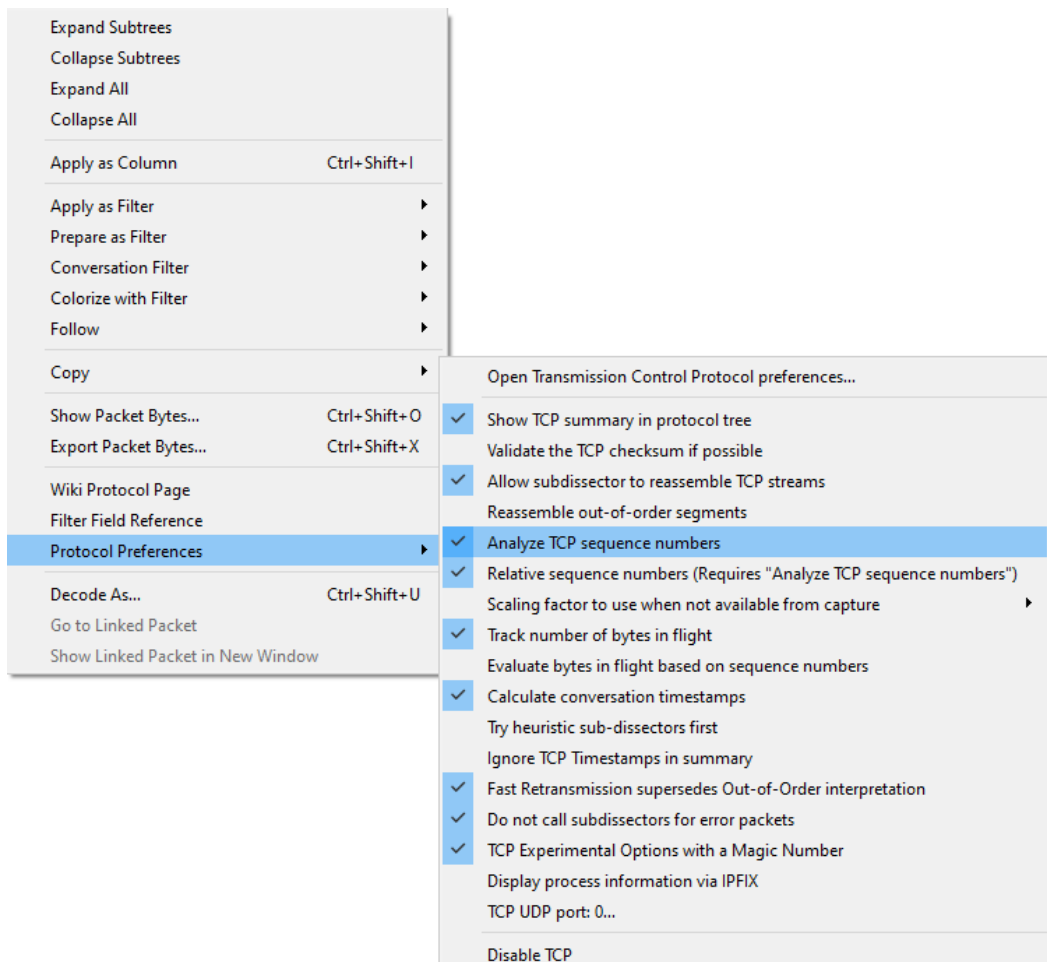


Figure 9.13 – Protocol preferences—relative sequence numbers

This will adjust the sequence numbers to a more understandable value.

When using a relative sequence number, the absolute sequence number is displayed below as Sequence Number (raw) : 3344099089, as shown in the following screenshot:

```

v Transmission Control Protocol, Src Port: http (80), Dst Port: 57678
  Source Port: http (80)
  Destination Port: 57678 (57678)
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 1448]
  Sequence Number: 18825      (relative sequence number)
  Sequence Number (raw): 3344099089
  [Next Sequence Number: 20273      (relative sequence number)]
  Acknowledgment Number: 135      (relative ack number)
  Acknowledgment number (raw): 2387614088
  1000 .... = Header Length: 32 bytes (8)

```

Figure 9.14 – Viewing the sequence numbers

Now that you understand how the sequence numbers are generated and displayed, let's see how they are represented during data transfer.

## Reviewing the sequence field values

As we know, the field values that deal with sequencing help provide a snapshot of the data that's exchanged during a TCP connection. The fields are as follows:

- **Sequence number (32-bit):** After the handshake, the data flow begins. In Frame 4, we can see Sequence number: 18825 (relative sequence number).
- **Sequence number (raw):** This value represents the unaltered sequence number. In Frame 4, we can see the value as 3344099089.
- **Next sequence number:** The value is in brackets, as it is calculated. Wireshark adds the current sequence number to the TCP segment length to get the next sequence number.

In addition to sequencing data, TCP also acknowledges receipt of all data.

## Acknowledging data

During data transfer, the OS keeps track of all bytes and reordering by using the sequence numbers. Every time the OS receives data, the receiving host acknowledges that the data was received and that they are ready to accept more, starting with the next expected byte.

The process occurs concurrent to the server sending data. As a result, it is called an **expectational acknowledgment**. As shown in the following diagram, the client sends an ACK to the server stating that they have received 524 bytes of data and they are ready for more, starting with 525:

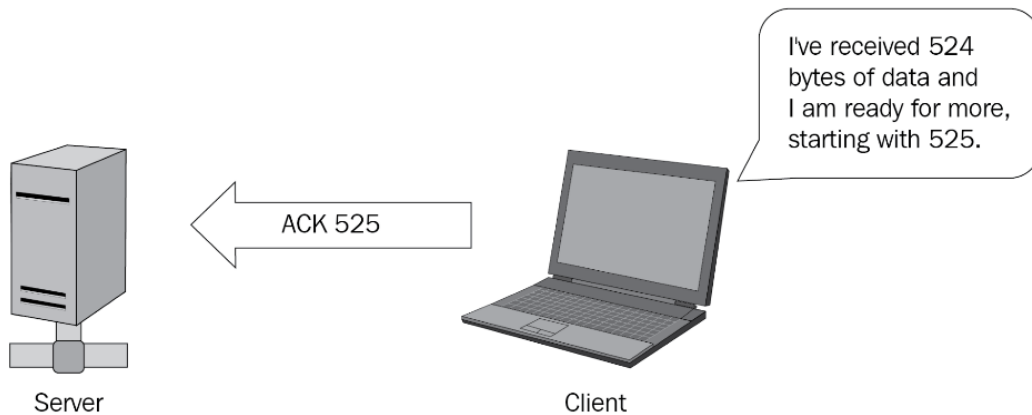


Figure 9.15 – Acknowledging the data

Within the TCP header, there are a couple of fields that are used to acknowledge receipt of the data. The fields are as follows:

- **Acknowledgment number 32-bit:** In Frame 4, we can see the value as Acknowledgment Number: 1 (relative ack number). The value is 1 because it is the first packet from the client.
- **Acknowledgment number (raw):** This value represents the unaltered acknowledgment number. In Frame 4, we can see the value as 3344080265.

In the line right after the acknowledgment number in *Figure 9.14*, you see 1000. This line represents the data **offset** field, which indicates the length of the TCP header in 32-bit multiples. Let's talk about how this value is calculated.

## Calculating the offset

The size of a fixed TCP header field is 20 bytes. However, many times in today's networks, the TCP header has additional options. As a result, the value is not always consistent. So that the receiving device knows when the data begins, the **offset** field will indicate the *length* of the header, as immediately after the TCP header, the data portion begins.

The TCP header for Frame 4 is calculated in the following manner:

1. The offset value is 1000, which is equal to eight (8) in binary.
2. The offset is in increments of four (4) bytes.
3. Calculated TCP header length:  $8 * 4 = 32$  bytes.

The offset value (or TCP header) for Frame 4 is 32 bytes.

Keep in mind the following when evaluating a TCP header:

- The *minimum* TCP header length is 20 bytes.
- The *maximum* TCP header length is 60 bytes.

If you want to do a quick check of the header length, place your cursor on the TCP header for Frame 4, and the value will be reflected in the status bar as shown:

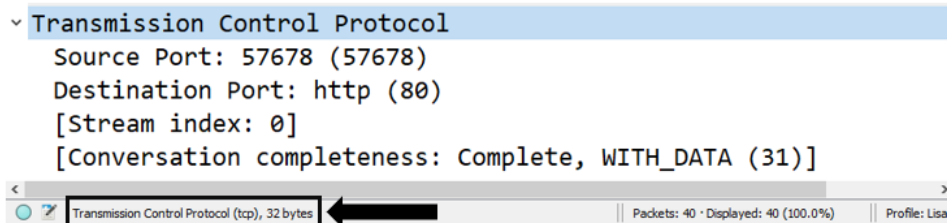


Figure 9.16 – Viewing the TCP header length

While keeping track of the data exchange, another important element in the TCP header is the use of flags, as discussed in the next section.

## Following the flags

TCP flags are used to indicate a particular state during a conversation. Some are commonly seen, such as ACK, FIN, and SYN; however, some are rarely seen in practical applications. TCP has eight control flags, as shown here:

Control flag	Bits	Function
<b>Reserved</b>	3	The <b>reserved</b> flag is for future use and should be set to zero.
<b>Nonce</b>	1	<b>Nonce</b> is experimental—possibly use with ECN.
<b>CWR</b>	1	<b>Congestion Window Reduced</b> : When set, this indicates that the sender is responding to indications of network congestion with congestion avoidance.
<b>ECE</b>	1	The <b>Explicit Congestion Notification (ECN)</b> Echo will notify the endpoints of any network congestion to avoid dropping packets. Both endpoints must be ECN capable in order for ECN to work. If this flag is set during the handshake, this means the endpoint is ECN capable.
<b>URG</b>	1	<b>Urgent</b> indicates a packet that should have priority. Rarely seen.
<b>ACK</b>	1	<b>Acknowledgment</b> acknowledges that the data was received and that the client is ready to accept more. Any packets sent by the client after the initial SYN packet will have the ACK flag set.
<b>PSH</b>	1	Normally, a buffer will hold data until it has a decent-sized packet to send. <b>Push</b> informs TCP that data should be sent immediately and not wait until the buffer is full.
<b>RST</b>	1	When set, the sender and receiver will abort the TCP connection. A <b>reset</b> can happen for a number of reasons. Some applications (such as Outlook and Exchange) will close TCP connections with RST (and FIN) as a matter of course, to try and prevent port exhaustion. In addition, many times, it is used to close an abnormal or malicious connection.
<b>SYN</b>	1	<b>Synchronization</b> synchronizes the sequence numbers. Only the first two packets of the handshake will have this flag set.
<b>FIN</b>	1	<b>Finis</b> means the communication has ended and there is no more data—close the connection.

Table 9.1 – Defining the TCP control flags

The TCP flags, when set, will tell the story of the TCP connection. Wireshark will reflect this state in the `Info` column of the packet list pane:

```

~ Flags: 0x018 (PSH, ACK)
  000. .... = Reserved: Not set
  ...0 .... = Nonce: Not set
  .... 0... = Congestion Window Reduced (CWR): Not set
  .... .0.. = ECN-Echo: Not set
  .... ..0. = Urgent: Not set
  .... ...1 = Acknowledgment: Set
  .... .... 1... = Push: Set
  .... .... .0.. = Reset: Not set
  .... .... ..0. = Syn: Not set
  .... .... ...0 = Fin: Not set
[TCP Flags: .....AP....]

```

Figure 9.17 – Viewing the TCP flags

#### Note

TCP is widely used, and the flags are important to control each session. However, TCP flags can be used in a malicious way to launch an attack or evade detection. As a result, the security analyst should make sure devices are tuned to monitor for non-standard and inappropriate use of TCP flags.

The TCP flags provide an indication of what is happening during a conversation. It's important to keep the data moving. As we'll see in the following section, the WS is used to notify the sender about just how much data a host can receive at any given time.

## Dissecting the window size

TCP is a full-duplex communication protocol, in which the sender and receiver constantly communicate with each other. During an active connection, the following takes place:

- The server sends data to the client.
- The client responds with an ACK along with a WS value (in bytes), which indicates how much data the client can accept.

Transmitting a WS with every ACK helps control the flow of data. Let's talk about how this works.



## Controlling the flow

Flow control is a protocol employed by TCP. In this process, the client sends a WS with each ACK so that the sender does not transmit too much data and overwhelm the host:

1. If the client cannot process all the data, the client will send an ACK with a lower WS value.
2. Once the client advertises a smaller WS, the server throttles back the data transfer.
3. When the client recovers and is able to accept more data, the client sends an ACK with a window update that reflects the new value.
4. The server can then continue to send data, until all data is sent to the client.

Keep in mind, the WS value can change at any time during a conversation.

### Note

A related term for flow control is called **sliding window**, as the value will slide back and forth as the endpoint adjusts the amount of traffic it can accept.

In the case of Frame 4, we can see Window: 46:

```
Window: 46  
[Calculated window size: 5888]  
[Window size scaling factor: 128]
```

Figure 9.18 – Viewing the window size

Although the client has indicated the WS as being 46 bytes, this value is most likely larger. Let's talk about why this value is different.

## Calculating the window size

Following the Window: 46 line, as shown in *Figure 9.18*, we can see [Calculated window size: 5888]. This value is larger than the actual WS because this stream uses a *scaling factor*, which changes the value of the window size. This is due to the difference in the WS over time.

The original TCP **Request for Comment (RFC)** 793 was written in 1981. At that time, buffer space was smaller, and the 16-bit window size value field would accommodate the actual window size that was available during the 1980s.

If all 16 bits are used, this would mean the window size is equal to  $2^{16}$ , or 65,536 bytes. As time passed, hardware improved, and the buffer space expanded beyond that limit. Because of this, options were used to expand the WS value in the TCP header. In the early 1990s, RFCs were written to address the larger buffer sizes, and a window scaling option provides a way to address the actual WS.

As a result, Wireshark calculates the window size by multiplying the *scaling factor* of 128 by the listed window size field value of 46 bytes, which gives us a calculated window size of 5,888 bytes.

Next, let's take a look at how the scaling factor is determined.

## Scaling the window value

[Window Size scaling factor: 128] is calculated by Wireshark. This reflects the scaling factor from the TCP options exchanged during the three-way handshake. As shown in the first packet of the three-way handshake (Frame 1), we can see that the TCP options sent by the server list the last option as Window scale: 7 (multiply by 128), as shown here:

- Options: (20 bytes), Maximum segment size, SACK permitted
  - TCP Option - Maximum segment size: 1460 bytes
  - TCP Option - SACK permitted
  - TCP Option - Timestamps: TSval 2216538, TSecr 0
  - TCP Option - No-Operation (NOP)
  - TCP Option - Window scale: 7 (multiply by 128)

Figure 9.19 – TCP options

Wireshark will determine the *calculated* WS by multiplying the scaling factor by the window size field value.

If the capture began after the three-way handshake, Wireshark has no way of knowing what the scaling factor is and will display [Window Size scaling factor: -1 (unknown)]. You may also see [Window size scaling factor: -2 (no window scaling used)]. In that case, Wireshark will display the *actual* window size.

If you do know the scaling factor, you can modify the value. Right-click anywhere in the TCP header and select **Protocol Preferences | Scaling factor to use when not available from capture**. Once there, select the appropriate value, as shown in the following screenshot:

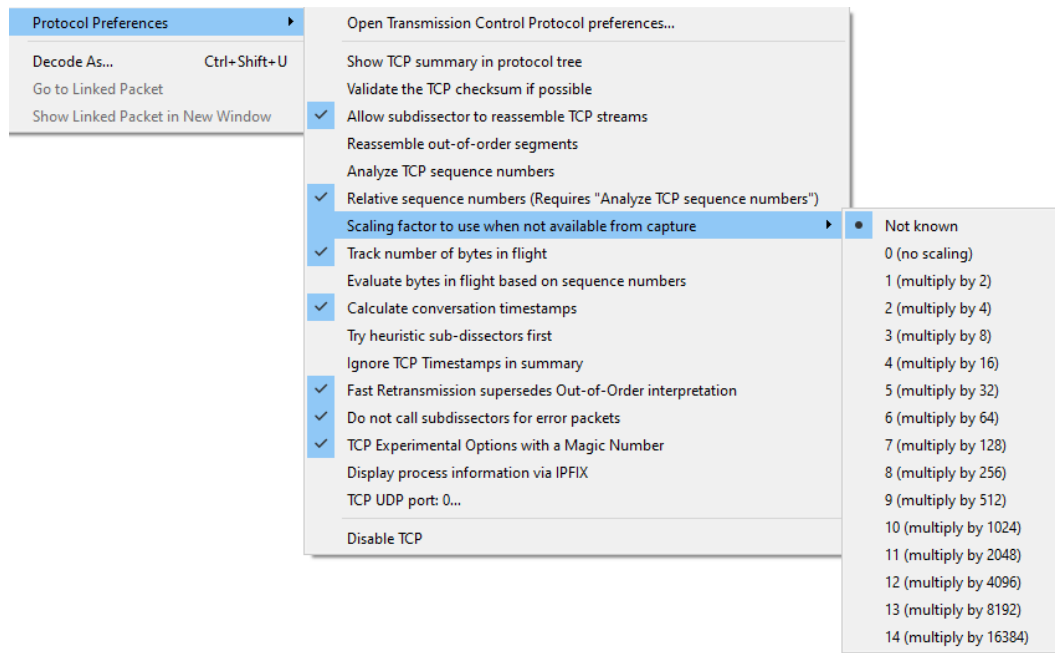


Figure 9.20 – TCP protocol preferences

In addition to the field values that monitor the communication, there are some other field values and options in the TCP header. Let's take a look.

## Viewing additional header values

The last part of the header lists additional values and options that help keep a TCP connection on track.

During data transmission, errors may occur. A checksum is a calculated value of the data portion of the packet that is periodically recalculated during transmission to ensure data integrity. The checksum is used for error detection, not correction. If the checksum is not accurate during recalculation, the packet is dropped.

In Frame 4, we can see the value of the checksum as Checksum: 0x4729 [unverified]. When doing a packet capture, the captured packet is presented to Wireshark before the hardware or network driver calculates the checksum. It may be incorrectly calculated, which will result in an error.

To avoid a checksum error, you can disable checksum validation by right-clicking anywhere in the TCP header, selecting **Protocol Preferences**, and *unchecking* **Validate the TCP checksum if possible**.

The TCP flags, which include the URG flag, indicate the packet that should have priority. If the URG flag is set, the receiving host will need to examine the frame to obtain relevant data. This is rarely used. A more commonly used flag is PSH, as it informs the TCP that data should be sent up the stack immediately.

## Adding TCP options

Before any TCP conversation, there is a three-way handshake. During the handshake, TCP generally has several options. The options will be listed during the SYN packet exchange. In Frame 4, the options include Timestamps and **No-Operation (NOP)**, as shown here:

```

  ▾ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Timestamps: TSval 2216543, TSecr 835172936

```

Figure 9.21 – TCP options for Frame 4

During capture, Wireshark gathers statistics and calculates values that can be used for analysis.

## Calculating values

At the bottom of the TCP header, there are two calculations, [Timestamps] and [SEQ/ACK analysis], as shown in the following screenshot:

```

  ▾ [Timestamps]
    [Time since first frame in this TCP stream: 0.047068000 seconds]
    [Time since previous frame in this TCP stream: 0.000112000 seconds]
  ▾ [SEQ/ACK analysis]
    [iRTT: 0.046956000 seconds]
    [Bytes in flight: 134]
    [Bytes sent since last PSH flag: 134]

```

Figure 9.22 – Calculated values in the TCP header

The [Timestamps] calculation indicates the elapsed time and is used to provide details about the capture found in **Statistics | Capture File Properties**.

[SEQ/ACK analysis] is a calculated field that includes information such as what frame was acknowledged and the **Round-Trip Time (RTT)**, which is used in functions such as the time-sequence graphs found under **Statistics | TCP Stream Graphs**.

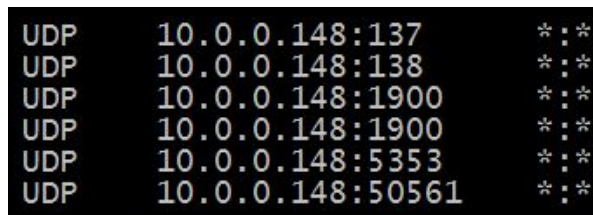
The last component listed is TCP payload (134 bytes), which (as discussed earlier) is the calculated offset value.

As you can see, there is a lot going on with TCP, which provides reliable data transport. In the next section, we will take a look at UDP, a transport protocol to use when speed—not reliability—is required for data transport.

## Understanding UDP

UDP is a lightweight, connectionless protocol used for data transfer. UDP does not have a handshake or connection process, nor does it have a teardown.

To see all your active UDP connections on a Windows machine, open a command line and run `netstat -anp udp`, as shown in the following screenshot:



```
UDP    10.0.0.148:137      *.*
UDP    10.0.0.148:138      *.*
UDP    10.0.0.148:1900     *.*
UDP    10.0.0.148:1900     *.*
UDP    10.0.0.148:5353     *.*
UDP    10.0.0.148:50561    *.*
```

Figure 9.23 – Netstat command showing UDP connection status

UDP doesn't have any ordering or reliability services; it simply delivers the data. Because of this, there isn't a need for a foreign (or remote) IP address and port. As a result, as shown in *Figure 9.23*, you will see only a *local* IP address and port for UDP.

Because of UDP's streamlined nature, it is an appropriate protocol for time-sensitive applications such as the following:

- **Dynamic Host Configuration Protocol (DHCP)**
- **Domain Name System (DNS)**
- **Trivial File Transfer Protocol (TFTP)**
- **Voice over Internet Protocol (VoIP)**

UDP is a lightweight protocol that is only 8 bytes in length. In the next section, let's look at the elements of a single UDP frame.

## Studying a single UDP frame

Unlike TCP, UDP is a lightweight protocol with a very simple header. UDP has only four fields and no options.

To examine UDP and for a deep dive into the UDP header, go to <https://www.cloudshark.org/captures/0320b9b57d35> and download the `DNS Question & Answer.pcapng` file. Go to Frame 1 so you can follow along. Once selected, you will see the following details:

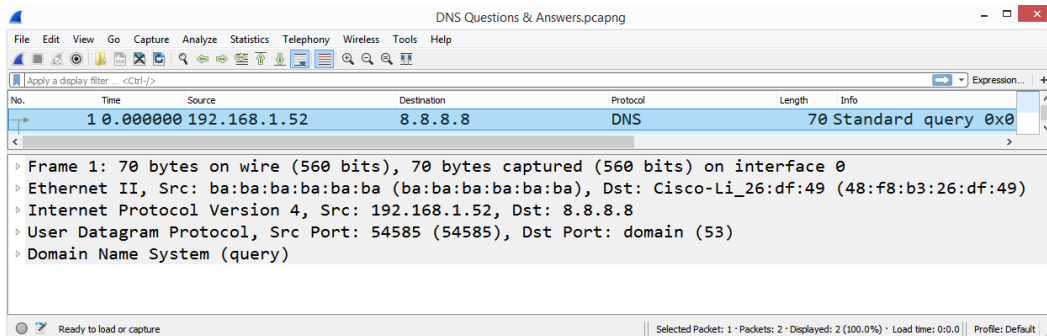


Figure 9.24 – Frame 1 showing the packet details pane

Starting from the top, Wireshark lists the contents of this single frame.

In Frame 1, we can see the following:

- **Frame 1:** The Frame label is not a protocol. It is a list of values or metadata generated by Wireshark that describes information about a single frame.
- **Ethernet II:** The frame header follows the metadata and provides information about the source and destination MAC address.
- **Internet Protocol Version 4:** Provides the details of the IP protocol used and includes the source and destination IP address.
- **User Datagram Protocol:** Provides the details of the UDP header.

Now that we have taken a look at the information that's found in a single UDP frame, let's examine the UDP header and each of the field values.

## Discovering the four-field UDP header

UDP has a four-field header that holds the values that keep track of the conversation, as shown in the following diagram:

UDP Header	
Source Port	Destination Port
Length	Checksum

Figure 9.25 – The UDP header

Now, let's take a look at each of the four UDP headers.

## Analyzing the UDP header fields

Starting at the top of the UDP header, we can see `User Datagram Protocol`, followed by a summary of what the header represents. Below the header and summary are the UDP header fields, as shown here:

```
• User Datagram Protocol, Src Port: 54585 (54585), Dst Port: domain (53)
  Source Port: 54585 (54585)
  Destination Port: domain (53)
  Length: 36
  Checksum: 0x448f [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
```

Figure 9.26 – The UDP header as shown in Wireshark

Unlike TCP, UDP has a simple header, with no additional communication details listed, such as `Timestamps` or `SEQ/ACK analysis`.

After the header, you will see the following:

- **Source Port 16-bit:** The source port field is the port on the sender's side. In `Frame 1`, the sender is a DNS client, using `Source Port: 54585`, which is not associated with any application; it is an ephemeral or temporarily assigned port that is used in this connection.
- **Destination Port 16-bit:** The destination port field is the port on the receiver's side. In this case, the port in `Frame 1` is `Destination Port: domain (53)`, which is the port on the DNS server accepting requests to resolve a domain name.

- **Length 16-bit:** In a UDP packet, the length represents the number of bytes in the UDP header and any data that follows. In `Frame 1`, we can see `Length: 36`, which is equal to the UDP header (8 bytes) and the DNS header (28 bytes).
- **Checksum 16-bit:** The UDP checksum is a calculated value of the data portion of the packet that is periodically recalculated during transmission to ensure data integrity.

**Note**

Using the UDP checksum is *optional* when using IPv4; however, it is *required* when using IPv6. The reason the checksum is required when using IPv6 is that IPv6 does not have a checksum, and the value in the UDP header is used to ensure data integrity.

UDP is always 8 bytes long as it does not have any header options. In addition, UDP is connectionless. Therefore, if there is trouble during the data transport, it's up to a higher-level protocol to communicate any issues or request any missing data.

## Summary

In this chapter, we focused on the transport layer (or Layer 4) of the OSI model, specifically the two predominant protocols in this layer, TCP and UDP. We started with an overview of the transport layer. We then evaluated TCP, a connection-oriented protocol. We saw that in order to achieve reliability, TCP sequences and acknowledges every octet. We now understand how, in addition to transporting data, TCP monitors the transmission, and provides not only flow control, but congestion control as well. We also took a closer look at the header, along with the eight control flags, then reviewed how window scaling works.

Along with TCP, we looked at the other predominant transport layer protocol, UDP, which ensures fast transportation of time-sensitive data and protocols such as DHCP and DNS. We reviewed the details of the four-field UDP header, which provides enough information to deliver data with no additional overhead.

Now that you have a solid understanding of TCP, the next step is to gain a better understanding of how TCP establishes and tears down a connection in more detail. In the next chapter, we will step through the process of starting a TCP conversation. We'll examine the TCP three-way handshake and resultant socket creation. In addition, we'll then study the packet exchange and have a closer look at the TCP options. Finally, we'll outline how TCP ends data transmission by exchanging FIN packets.



## Questions

Now, it's time to check your knowledge. Select the best response, then check your answers, which can be found in the *Assessment* appendix:

1. A \_\_\_\_\_ is defined as an IP address and a port.
  - A. window
  - B. socket
  - C. checksum
  - D. sequence
2. ACK 725 means *I have received \_\_\_\_\_ bytes of data and I am ready for more, starting with \_\_\_\_\_*.
  - A. 700 and 800
  - B. 724 and 725
  - C. 725 and 726
  - D. 725 and 725
3. The \_\_\_\_\_ flag informs TCP that data should be sent immediately.
  - A. RST
  - B. SYN
  - C. ACK
  - D. PSH
4. If the offset value is 0101, the TCP header length = \_\_\_\_ bytes.
  - A. 32
  - B. 8
  - C. 20
  - D. 64
5. In a normal connection, \_\_\_\_ will use UDP.
  - A. HTTP
  - B. SMTP
  - C. DHCP
  - D. FTP

6. Unlike TCP, a UDP header does not have any \_\_\_\_\_.
  - A. ports
  - B. sockets
  - C. checksums
  - D. options
7. If the client advertises a scaling factor of 9, the OS will need to multiply the WS by \_\_\_\_\_.
  - A. 2
  - B. 9
  - C. 128
  - D. 512

## Further reading

To read more about TCP analysis in Wireshark, visit [https://www.wireshark.org/docs/wsug\\_html\\_chunked/ChAdvTCPAnalysis.html](https://www.wireshark.org/docs/wsug_html_chunked/ChAdvTCPAnalysis.html).



# 10

# Managing TCP Connections

One of the most important, yet least understood, **Transmission Control Protocol (TCP)** concepts is the three-way handshake. A TCP handshake initiates the connection and sets up the parameters. No data is exchanged until this process is complete. Similar to the handshake is the teardown when the two endpoints exchange a series of **finis (FIN)** packets, which indicates the session is complete.

In this chapter, we'll take a more detailed look at the handshake and resultant socket creation. So that you can home in on a single TCP stream, we'll take a large capture, and subset, mark, and filter the packets, so we can examine the TCP handshake. As we move through the chapter, we'll have a greater understanding of the TCP options exchanged during the handshake. We'll learn what they mean and why they are required to have a conversation on today's networks. In addition, we'll see how we can easily modify protocol preferences, such as analyzing TCP sequence numbers with a simple right-click. Finally, we will examine the TCP teardown process and see how the FIN flag indicates the end of data transmission.

This chapter will address all of this by covering the following:

- Dissecting the three-way handshake
- Discovering TCP options
- Understanding TCP protocol preferences
- Identifying a TCP teardown

## Dissecting the three-way handshake

In computing, a handshake is an exchange of information between devices that sets up the parameters of the conversation. Each side sends what is available, and the two endpoints agree on the terms before any data is exchanged.

In most cases, the *client* will initiate the conversation with a *server* by sending a **synchronization (SYN)** packet; the *server* responds with a **synchronization acknowledgment (SYN-ACK)**, and the *client* then completes the handshake with an **acknowledgment (ACK)**. The TCP handshake is as follows:

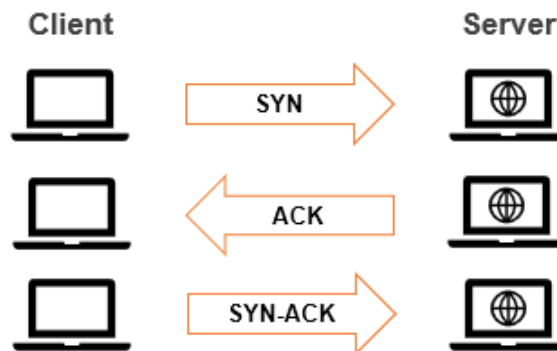


Figure 10.1 – The TCP three-way handshake

After the handshake is complete, the data exchange will follow.

For a closer look at the three-way handshake, go to <http://tcpreplay.appneta.com/wiki/captures.html#bigflows-pcap>. Once there, download `bigFlows.pcap` so that you can follow along. BigFlows is a large capture that has many protocols and conversations, as shown in the following screenshot:

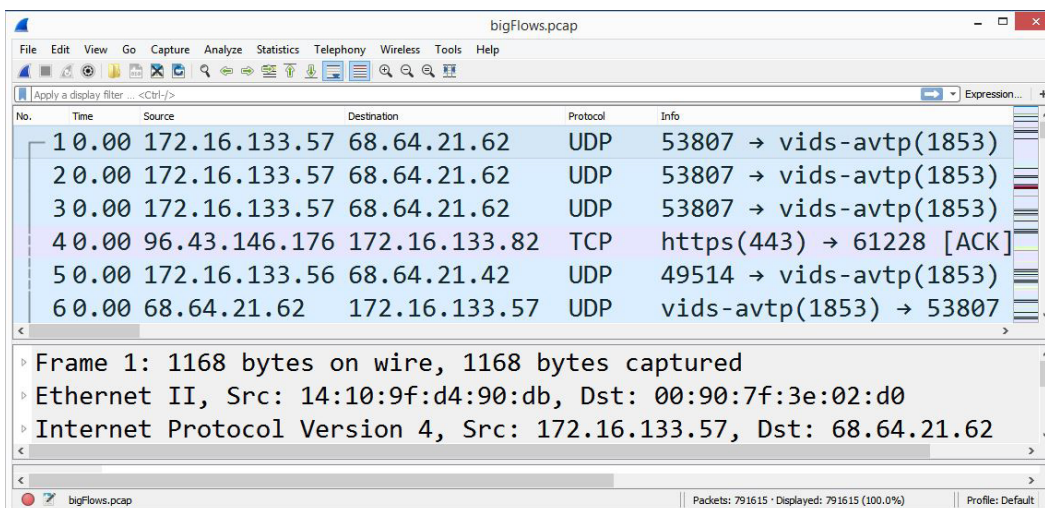


Figure 10.2 – bigFlows.pcap

BigFlows has 791,615 packets, as shown in the lower right-hand side of the status bar in *Figure 10.2*. Although you could technically work with the entire capture, in the next section, we will isolate a single stream and then create a smaller, more manageable file.

## Isolating a single stream

When capturing traffic, Wireshark keeps track of all the streams. In a file the size of `bigFlows.pcap`, there will be many TCP and **User Datagram Protocol (UDP)** streams. Although we can filter on any of the streams, for now, let's use the TCP stream 312, as this includes the handshake, options, data, and then the FIN exchange to end the session.

To show only stream 312, go to the display filter and enter `tcp.stream eq 312`, and then press *Enter*. Because this is such a large file, it will take Wireshark several seconds to run the filter. Once you are done, you will see an example of a complete TCP stream, as shown here:

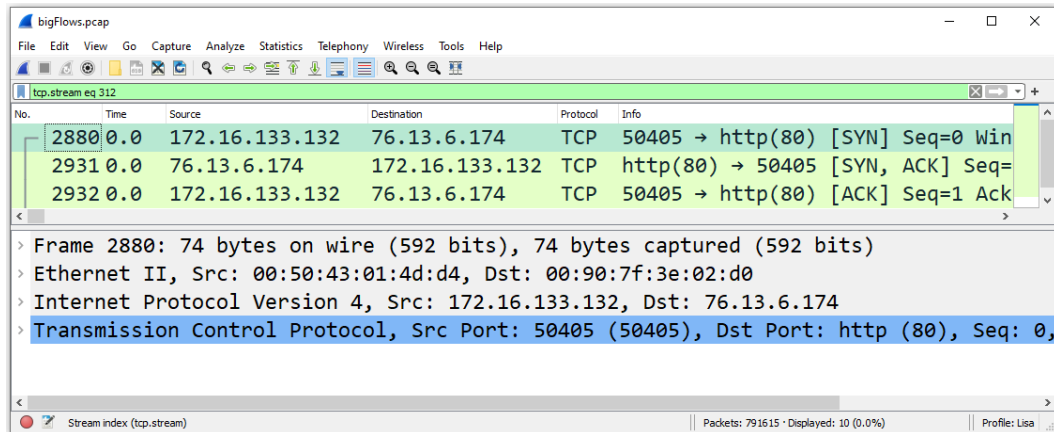


Figure 10.3 – `tcp.stream eq 312`

Now that we have a single isolated stream, we'll want to subset the capture and save it as a smaller file. To subset *only* the TCP stream 312, go to **File | Export Specified Packets**.

This will open a dialog box that offers various ways to export specified packets, as shown in the following screenshot:

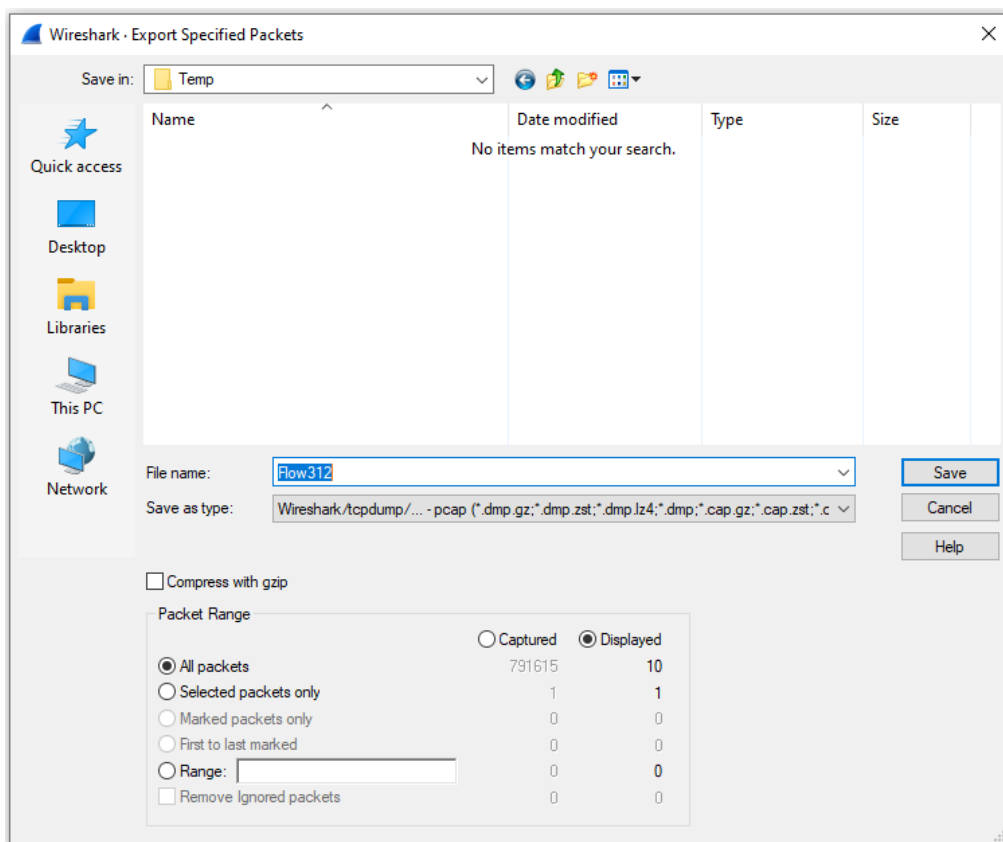


Figure 10.4 – Select the All packets and Displayed options

Near the bottom of the dialog box, you will see a header, **Packet Range**, where you will make your selections. If you have filtered the capture, Wireshark will assume that you would like to export *only* the displayed packets, and the radio button for **Displayed** will be active.

Select **All packets** and **Displayed**, as shown in the screenshot, and then save as `Flow312.pcapng`.



Close `bigFlows.pcap` and clear the display filter, and then open the newly created file. In the next section, let's zero in on the handshake and examine each of the packets exchanged.

## Marking the TCP handshake

One of the ways you can isolate a series of packets within Wireshark is by marking them, which will modify the packet to have a black background with white text. Once we mark them, we can filter according to the marked packets to focus on the handshake.

In the file, we'll identify the handshake by marking the packets. We know that to begin a session, TCP starts with a handshake that uses three packets as follows:

1. The client sends a SYN packet to the server.
2. The server responds by sending a SYN ACK packet.
3. The client sends a final ACK packet.

Once the handshake is complete, the data flow begins.

Wireshark will identify the three-way handshake and the exchange of packets by showing the transaction details in the info column (if you have this column header active). In the `Flow312.pcapng` capture, packets 1, 2, and 3 represent the handshake.

Once the handshake is identified, we'll mark each of the three packets. To mark the packets, select each of the packets and right-click the **Mark/Unmark Packet** submenu choice, as shown in the following screenshot:

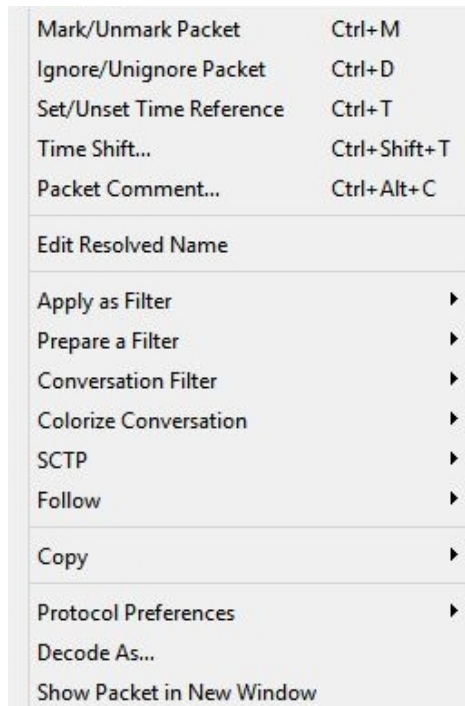


Figure 10.5 – Mark/Unmark Packet

Once you have marked the packet, the background will turn black and the text will be white, as shown here:

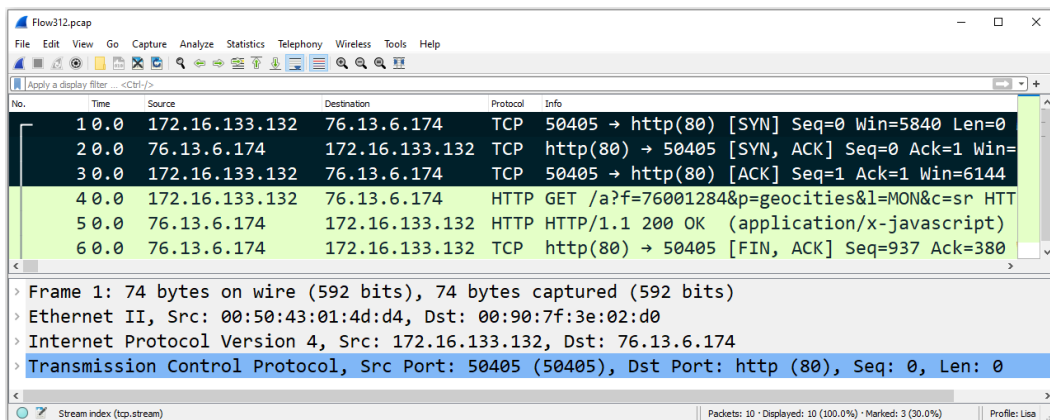


Figure 10.6 – Results of marking a packet

After that, we'll want to view only the marked packets by entering `frame.marked==1` in the display filter and pressing *Enter*. Clear the marks by going to **Edit | Unmark all Displayed** so that we can begin to dissect the handshake.

Now that we have singled out the three-way handshake, let's take a look at each of the three packets.

## Identifying the handshake packets

In this section, we'll take a look at each of the packets and examine the flags, along with the sequence and acknowledgment numbers. Let's start with the SYN packet.

### Sending the SYN packet

In the first packet, the client will initiate the connection by sending a SYN packet to the server and then wait for a response.

To see all the field values, go to **Frame 1**, and expand the TCP header as shown in the following screenshot:

```

Transmission Control Protocol, Src Port: 50405 (50405), Dst Port: http (80), Seq: 0, Len: 0
  Source Port: 50405 (50405)
  Destination Port: http (80)
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 1040466690
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 0
  Acknowledgment number (raw): 0
  1010 .... = Header Length: 40 bytes (10)
  > Flags: 0x002 (SYN)
  Window: 5840
  [Calculated window size: 5840]
  Checksum: 0x9222 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  > Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP),
  > [Timestamps]
```

Figure 10.7 – The Flow312.pcapng TCP header in Frame 1

In **Frame 1**, you'll see the source and destination ports along with other field values in the header. This includes the sequence and ACK numbers, as follows:

- Sequence number: 0 (relative sequence number)
- Acknowledgment Number: 0

If we expand the flags, we see that the *Syn* flag is set, as shown in the following screenshot. Keep in mind that the *Syn* flag will *only* be used to synchronize the sequence numbers during the first two packets of the handshake:

```

Flags: 0x002 (SYN)
 000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...0 = Acknowledgment: Not set
.... .... 0... = Push: Not set
.... ..... 0.. = Reset: Not set
> .... .... ..1. = Syn: Set
.... .... ...0 = Fin: Not set
[ TCP Flags: .....S. ]

```

Figure 10.8 – The TCP Syn flag set

Although all three packets of a handshake are important, the first two packets set up the parameters of the conversation. Select **Frame 1** and view the TCP header in the **Packet Details** pane. Then, select the TCP header, and we see Transmission Control Protocol (tcp), 40 bytes in the status bar along the bottom, as shown in the following screenshot:

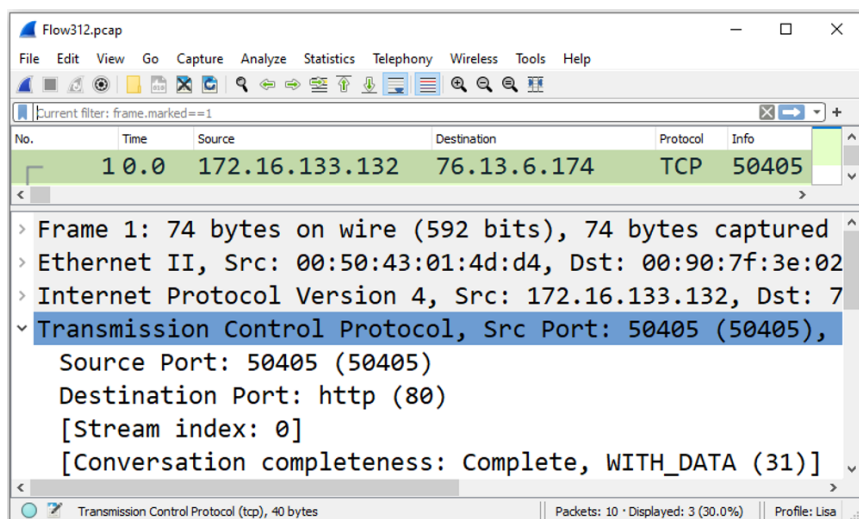


Figure 10.9 – TCP header 40 bytes

While a normal TCP header is 20 bytes, this header is 40 bytes. The TCP header size is larger because it contains options that are added to the header.

**Note**

In most cases, you will see a larger header size in the first two packets of the three-way handshake. You may also see a larger header size in subsequent frames when the TCP header contains options.

In addition, within the field values, we see that Wireshark has identified this conversation as [Stream: 0]. On the client side, the operating system will create a local socket with an IP address and a port number combination of 172.16.133.132: 50405 to receive data.

Now that we have seen the first packet of the three-way handshake, let's examine the second packet, the SYN-ACK packet.

## Returning the SYN-ACK packet

Once the server agrees to take part in the connection, the server will return a SYN-ACK and wait for a final ACK to start the connection.

In **Frame 2**, you will see the field values. In this case, the ACK number has changed. The sequence and ACK numbers in **Frame 2** are as follows:

- Sequence number: 0 (relative sequence number)
- Acknowledgment Number: 1

In addition, the TCP flags are now set to SYN-ACK, as shown in the following screenshot:

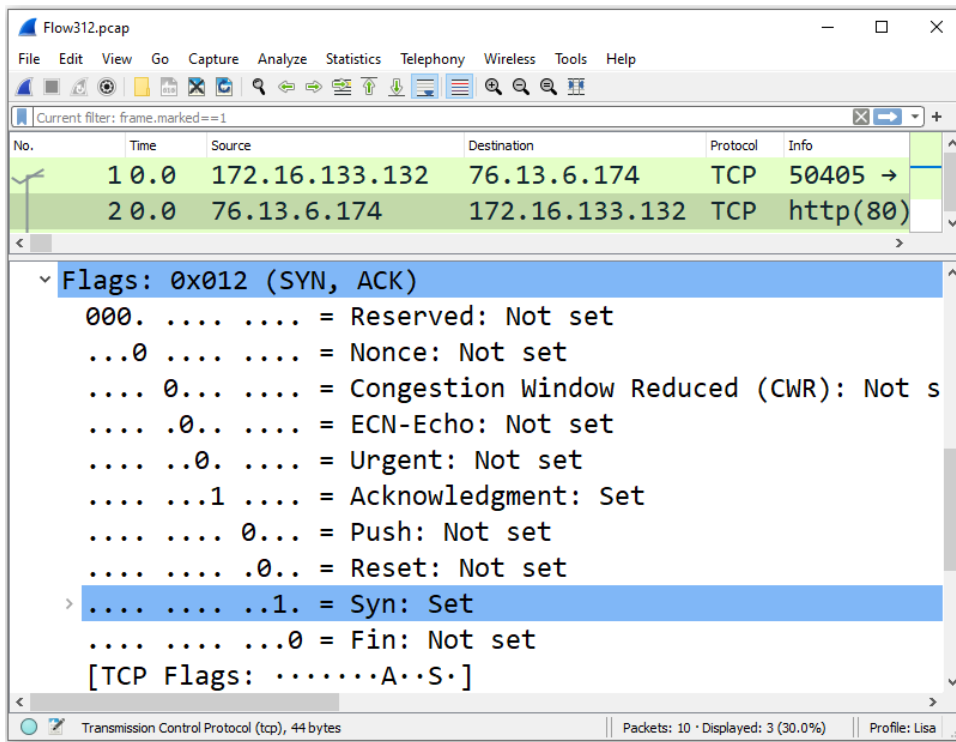


Figure 10.10 – Viewing the TCP SYN-ACK flags set

Before any data is exchanged, the handshake must be complete with the ACK packet, as discussed next.

## Finalizing with an ACK packet

**Frame 3** is the final packet in the three-way handshake. At this point, the sequence and ACK numbers are as follows:

- Sequence number: 1 (relative sequence number)
- Acknowledgment Number: 1

To see the details, go to **Frame 3** and then to the TCP header, and then expand the TCP flags, which are now set at ACK, as shown in the following screenshot:

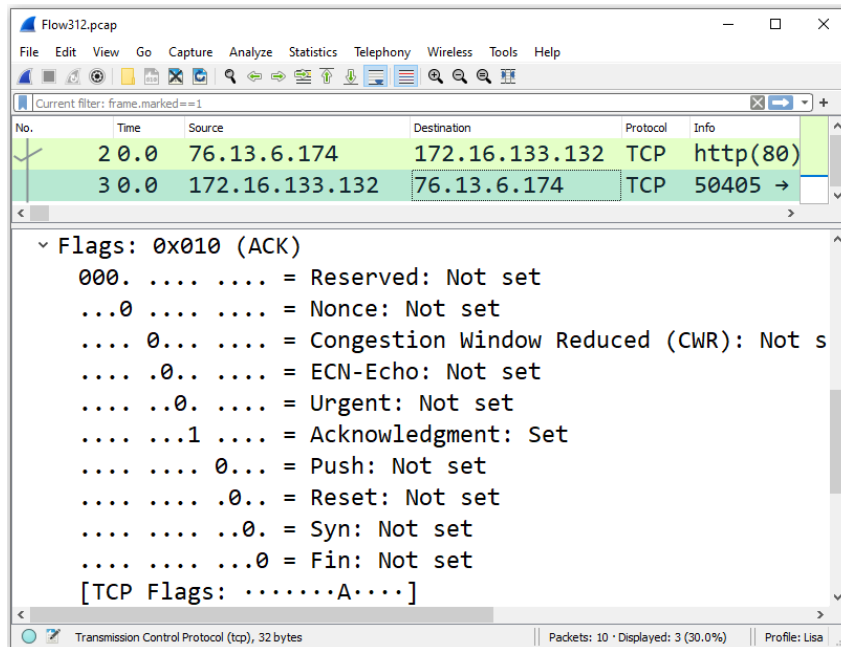


Figure 10.11 – The TCP ACK flag set

In addition, we also see Transmission Control Protocol (tcp), 32 Bytes in the status bar along the bottom, as shown in the preceding screenshot. Again, this is because this connection has TCP options, which adds to the length of the TCP header.

In many cases, there are TCP header options that outline and further define the parameters of the conversation. In the next segment, we'll look at TCP options in general, and then focus on the options of the TCP conversation in Flow312.pcapng.

## Learning TCP options

While TCP is already an amazing protocol, it also permits various options that can be added to the TCP header to extend the functionality. The complete list, last updated in February 2021, can be found at <https://www.iana.org/assignments/tcp-parameters/tcp-parameters.txt>.

Keep in mind the following about TCP options:

- Not all options are used.
- Some of the options are experimental.
- Some are used for specific reasons and do not have an associated **Request for Comment (RFC)**.
- Some have been developed and used without the proper **Internet Assigned Numbers Authority (IANA)** assignment.

The seven most common options are listed in the following table:

Kind	Length	Meaning	Reference
0	1	<b>End of option list (EOL)</b>	[RFC793]
1	1	<b>No-Operation (NOP)</b>	[RFC793]
2	4	<b>Maximum Segment Size (MSS)</b>	[RFC793]
3	3	Window Scale	[RFC7323]
4	2	<b>Selective Acknowledgment (SACK)</b> permitted	[RFC2018]
5	N	SACK	[RFC2018]
8	10	Timestamps	[RFC7323]

Table 10.1 – TCP options

The first three, EOL, NOP, and MSS, are from the original TCP RFC 793. The others were developed over time. Any options will follow the TCP header and are in multiples of 8 bits (or 1 byte). The entire header must be a multiple of 32 bits, or 4 bytes, for memory alignment. Therefore, in some cases, padding is required to ensure that the header is a multiple of 4 bytes. The entire header must be a multiple of 32- bits, (or four 4 bytes), for memory alignment.



To see the TCP options for `Flow312.pcap`, select the options header in **Frame 1**, where additional conversation parameters are listed, as shown here:

```
Options: (20 bytes), Maximum segment size, SACK permissive
  TCP Option - Maximum segment size: 1460 bytes
  TCP Option - SACK permitted
  TCP Option - Timestamps: TSval 131517608, TSecr 0
  TCP Option - No-Operation (NOP)
  TCP Option - Window scale: 10 (multiply by 1024)
```

Figure 10.12 – Flow312 Frame 1 options

So that you have a better understanding of each of the seven common options, let's take a look at each of them, starting with the EOL option.

## Grasping the EOL option

EOL is a single byte used at the end of the options list. To see an example, open the `Flow312.pcap` packet capture, put `tcp.option_kind == 0` in the display filter, and press *Enter*. Wireshark will display **Frame 2**. Expand the TCP header options, which will show the EOL at the end of the list:

```
Options: (24 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation
  TCP Option - Maximum segment size: 1460 bytes
  TCP Option - No-Operation (NOP)
  TCP Option - Window scale: 1 (multiply by 2)
  TCP Option - No-Operation (NOP)
  TCP Option - No-Operation (NOP)
  TCP Option - Timestamps: TSval 1707407197, TSecr 131517608
  TCP Option - SACK permitted
  TCP Option - End of Option List (EOL)
```

Figure 10.13 – Flow312 Frame 2 options list

Another TCP option is NOP, which isn't an option at all, but a way to occupy space, as we'll see next.

## Using NOP

NOP is essentially a placeholder to separate different options. How and where NOP is used is dependent on the operating system. For example, as shown in the preceding screenshot, we can see that two NOPs are placed between the Window Scale and Timestamps options.

In addition to using NOP to separate the various options, NOP is used to ensure that the options header is a multiple of 32- bits, (or four 4 bytes), for memory alignment.

For example, if there is an option with 3 bytes, such as a window scale, a single 1-byte NOP will be added so that the option's length totals 4 bytes.

Next, we'll look at an option that helps outline the acceptable receive segment size, which is significant in many ways.

## Defining the MSS

The MSS is an option that defines the maximum (receive) segment size. This value is important for several reasons.

During a conversation between endpoints, TCP monitors the connection to ensure that the optimal data is sent, so as not to waste bandwidth. When looking at a typical frame, we can see the differences between the TCP MSS and the **Internet Protocol (IP) Maximum Transmission Unit (MTU)** as shown in the following figure:

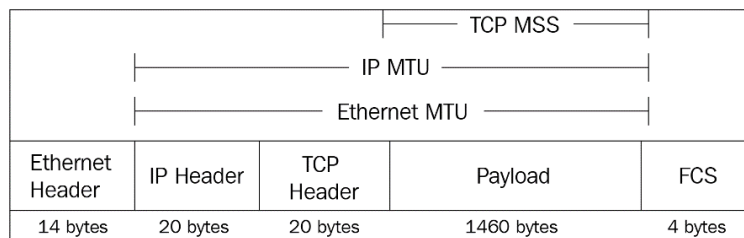


Figure 10.14 – Comparing the MSS with the MTU

During a data transaction, TCP keeps track of the following values:

- **Window size (WS)**, for flow control, so as not to overwhelm the receiving host.
- **Network**, for evidence of congestion by using the **congestion window (CWND)**. When necessary, the server will throttle the data transfer.
- **MTU** so the host sends only the amount of data that the *network* can handle so as to prevent the need to fragment the datagram.

On the network, the sending host monitors these values, as it can only send the *smallest* of the three values.

For example, a host needs to send 1,800 bytes of data. The network limits are as follows:

- **CWND:** 900 bytes
- **MTU:** 1,500 bytes
- **WS:** 1,800 bytes

According to the values listed, TCP must send the smallest value, which is **CWND**: 900 bytes.

The MSS is not always included in the options header. If this option is not used, the server can send a segment of any size, while keeping in line with the network limits.

As we can see, the MSS provides essential information to ensure optimal data flow. Let's now review another common option, the WS.

## Scaling the WS

TCP is a full-duplex communication protocol, in which the sender and receiver constantly communicate with each other. **Flow control** is an end-to-end control method where a host transmits a WS with every ACK, indicating how many bytes it can accept, so the sender does not transmit too much data and overwhelm the host.

Window Scale is an option that allows the WS to be expanded according to a scaling factor obtained from the TCP options exchanged during the three-way handshake. The WS value is used to increase the maximum WS that is allowed. Although optional, this provides information to the server of a more accurate WS value.

Let's outline why this is an important option. In the TCP header, the window field value is 16 bits, which permits a maximum size of  $2^{16}$  (or 65,535) bytes. The original RFC for TCP was written in the early 1980s when buffer sizes were small, so this value made sense. However, as time passed, it became evident that a larger value would be required, and the Window Scale option provided a way to truly represent that value.

When using the Window Scale option, the total value can be up to  $2^{30}$ . This value as a metric is beneficial when data travels, especially on high-bandwidth WAN links; a larger WS will improve performance. Intermediary devices can be tuned to accept larger WS; for example, when configuring a Cisco router that supports window scaling, you can adjust this value up to 1,073,741,823 bytes.

To illustrate this, the following diagram shows a connection where the client advertises a WS of 35,000 bytes. The server begins to send the packets to the client in line with the WS:

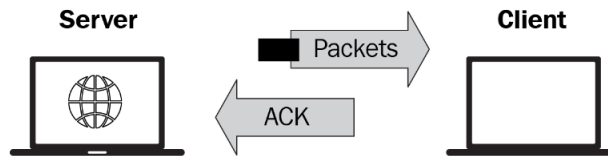


Figure 10.15 – Server sending data in line with a smaller WS

While in the connection, if the WS starts to drop, the server will need to throttle back the data so as not to overwhelm the client. However, if the client uses scaling and now advertises a WS of 70,000 bytes, the server can send twice as many packets, as shown in the following diagram:

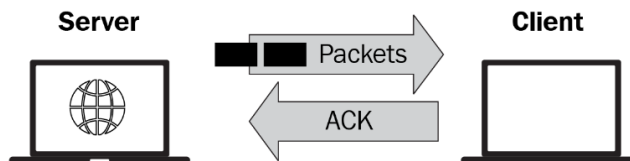


Figure 10.16 – A larger WS allows the server to send more data

Increasing the WS will utilize all available bandwidth and provide a more efficient data transfer.

Networks can be unstable, and data transfer does not always go in an orderly fashion. This next option provides an overview of how a client can selectively acknowledge the data it has received, so the server only has to retransmit the missing bytes.

## Permitting SACK

Over time, there have been improvements to the TCP/IP suite of protocols. One such improvement is SACK. In the case of SACK, the client will notify the server *only* if there are any missing packets, with the goal of keeping the data flowing. Let's discuss how this option improves data flow.

TCP is a connection-oriented protocol. During transmission, all data is sequenced and acknowledged to ensure the complete transfer of all data required for that session. This is achieved in the following manner.

After the server sends data to the client, the client will acknowledge that the data was received and is ready to accept more. This is communicated to the server by sending an ACK that indicates the next expected byte. To ensure complete data transfer, the server monitors the acknowledgments. However, sometimes there is a gap in transmission.

For example, the server has received ACK 1-100 and then ACK 151-200. In this case, there is a perceived gap of ACK 101-150, and the server believes the client is missing bytes 101-150.

The server has no idea why there is a gap in transmission and will resend all the data that it believes is missing. The gap may be due to one of the following reasons:

- The client *did not* receive the data.
- The client *did* receive the data, but the server did not receive the ACK.

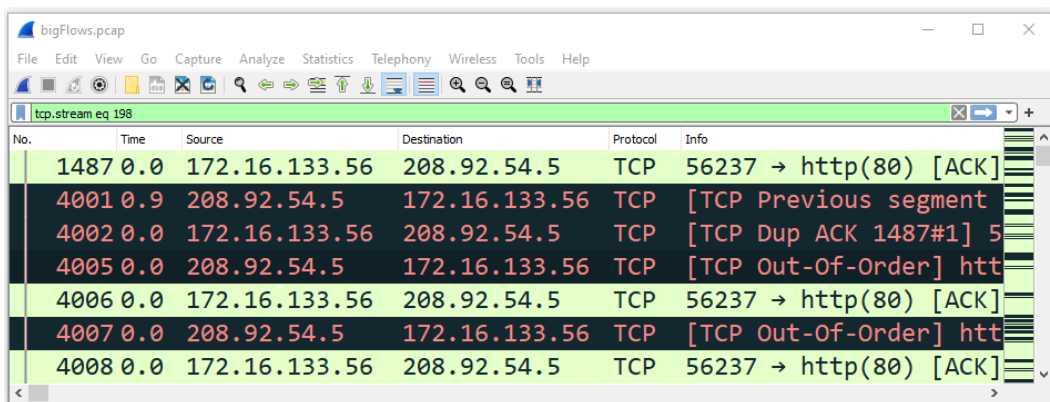
This can lead to unnecessary retransmission of data.

By using SACK, the server only has to resend any missing data. While using SACK, there are two options that are sent in the first two packets of the three-way handshake, as shown here:

- The SACK-permitted option indicates that SACK can be used after establishing a connection.
- The SACK option permits selective acknowledgment of data.

The TCP SACK option uses two 16-bit fields, so the client can indicate the bytes it has received.

For example, open the `bigFlows.pcap` capture, and use the `tcp.stream eq 198` filter. Within `tcp.stream 198`, we see some indication of problems, such as a **Duplicate Acknowledgement (Dup ACK)** along with a couple of Out-Of-Order packets, as shown in the following screenshot:



No.	Time	Source	Destination	Protocol	Info
1487	0.0	172.16.133.56	208.92.54.5	TCP	56237 → http(80) [ACK]
4001	0.9	208.92.54.5	172.16.133.56	TCP	[TCP Previous segment
4002	0.0	172.16.133.56	208.92.54.5	TCP	[TCP Dup ACK 1487#1] 5
4005	0.0	208.92.54.5	172.16.133.56	TCP	[TCP Out-Of-Order] htt
4006	0.0	172.16.133.56	208.92.54.5	TCP	56237 → http(80) [ACK]
4007	0.0	208.92.54.5	172.16.133.56	TCP	[TCP Out-Of-Order] htt
4008	0.0	172.16.133.56	208.92.54.5	TCP	56237 → http(80) [ACK]

Figure 10.17 – Stream 198

In **Frame 4006**, the client used TCP option - SACK 10852-11096, as shown in the following figure:

```

^ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), SACK
  ^ TCP Option - No-Operation (NOP)
  ^ TCP Option - No-Operation (NOP)
  ^ TCP Option - SACK 10852-11096
    Kind: SACK (5)
    Length: 10
    left edge = 10852 (relative)
    right edge = 11096 (relative)
    [TCP SACK Count: 1]

```

Figure 10.18 – TCP SACK option evident in bigFlows.pcap

Within the SACK option, we see the following values, which indicate the gap in transmission:

- left edge = 10852 (relative)
- right edge = 11096 (relative)

By using SACK, the server only needs to send the data from sequence number 10852 to 11096, which can prevent unnecessary retransmissions and keep the data flowing.

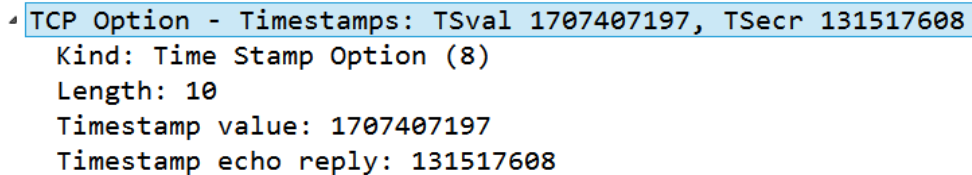
Another TCP option is timestamps, which monitors the transmission and keeps track of the round-trip time during the data exchange.

## Using timestamps

TCP relies on time as part of the functions of flow control and reliable data transfer. Data travels through LANs and over WANs. Every network is different, and TCP needs to understand the degree of latency on each network in order to set appropriate ACK timeout values.

Using the Timestamps option, TCP can monitor the round-trip times. This is so the sending host can retransmit a packet if it does not receive an ACK in a timely manner.

In the `Flow312.pcap` capture in **Frame 2**, open the options to view the Timestamps option, as shown in the following screenshot:



The screenshot shows the details pane of a packet in Wireshark. The selected item is 'TCP Option - Timestamps: TSval 1707407197, TSecr 131517608'. Below this, the following fields are listed: 'Kind: Time Stamp Option (8)', 'Length: 10', 'Timestamp value: 1707407197', and 'Timestamp echo reply: 131517608'.

Figure 10.19 – Viewing the TCP Timestamps option

Within this option, there are the following values:

- **Kind:** (1 byte) This field indicates the type of option, in this case, the value is (8), which is the Timestamps option.
- **Length:** (1 byte) This indicates the length of this option's header in bytes; in this case, it is 10 bytes.
- **Timestamp value (TSval):** (4 bytes) This is the timestamp clock on the sender's side. The value is 1707407197.
- **Timestamp echo reply (TSecr):** (4 bytes) This is the echo reply sent by the remote host. The value is in **Frame 2**, and is 131517608.

TCP uses the Timestamps value to monitor the round-trip time in various segments in the path. The Timestamps option must be set during the handshake. After that, you will see the option reporting back during the conversation.

As we can see, TCP can set various options during the handshake that further define the parameters of the conversation.

While working with Wireshark, there may be a preference in the way the protocol responds or is configured. We can modify many of the protocol preferences, as we'll see next.

## Understanding TCP protocol preferences

In Wireshark, there are several protocols that we can modify to display the data, according to our preferences. To modify the way a protocol is displayed, select a header and right-click to view **Protocol Preferences**, as shown in the following screenshot:

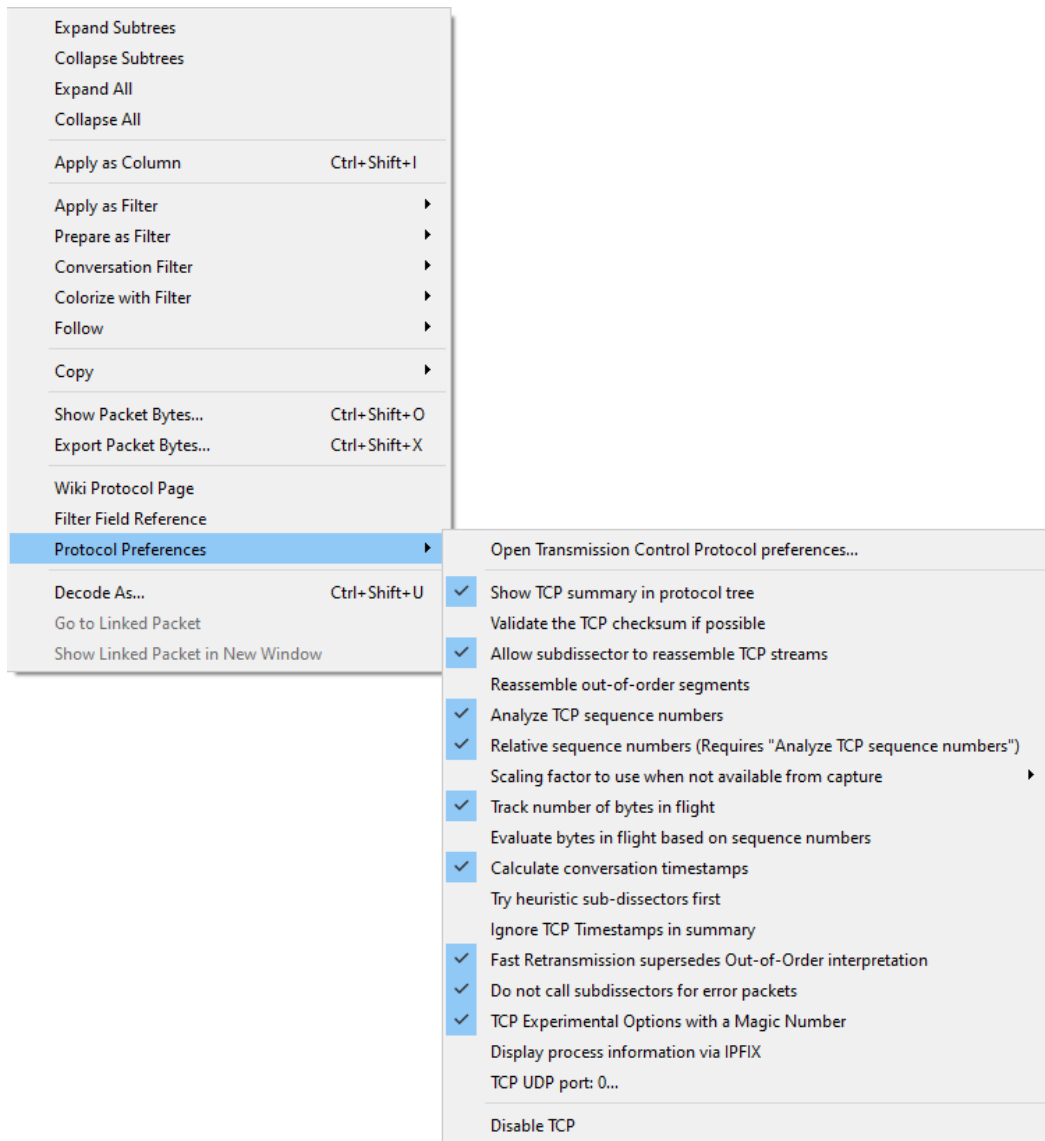


Figure 10.20 – Viewing TCP protocol preferences



At the top of the list of preferences, there is another option, **Open Transmission Control Protocol preferences...**, which will open the Wireshark preferences, as shown in the following screenshot:

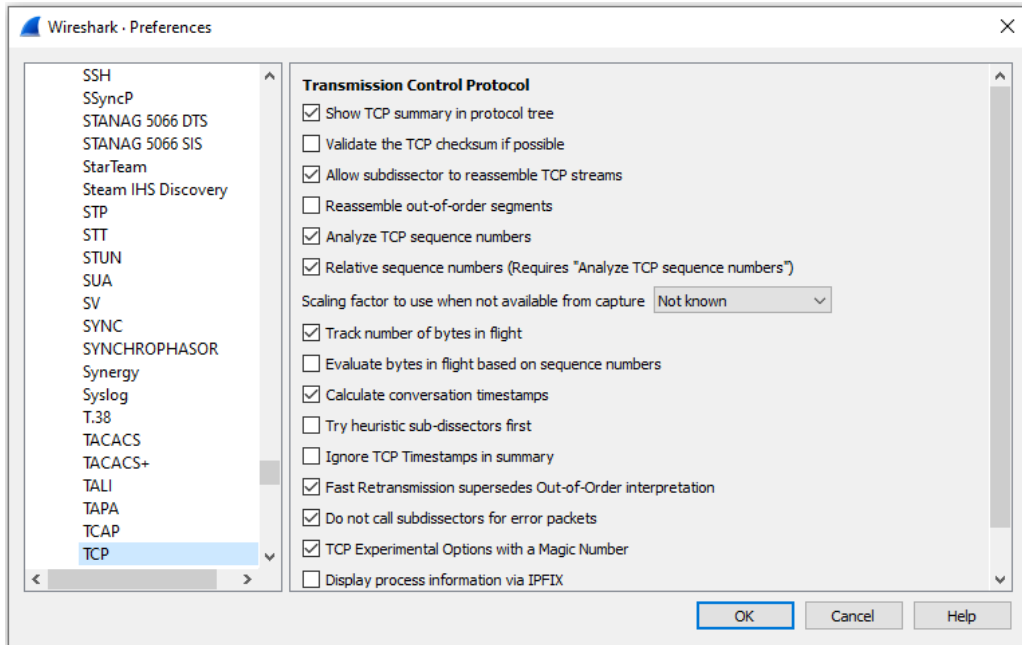


Figure 10.21 – TCP preferences

Once the **Preferences** dialog box is open, you can select and modify some of the TCP options.

## Modifying TCP preferences

When in the preferences dialog box for TCP, you will see a list that outlines your choices, as follows:

- **Show TCP summary in protocol tree:** When selected, this option will show a summary of what has transpired in that packet.
- **Validate the TCP checksum if possible:** TCP has a checksum that is used for error detection. In most cases, this option is not selected, as the checksum will offload to the NIC and the value will be invalid and indicate an error.
- **Allow subdissector to reassemble TCP streams:** When selected, this will allow an upper-layer protocol to reassemble the TCP stream.

- **Reassemble out-of-order segments:** When selected, this will put the segments in the correct order to display in their ordered form. This option is disabled by default as, when used, it can consume excessive memory.
- **Analyze TCP sequence numbers:** This option is helpful with analysis, as Wireshark will monitor the sequence numbers that help identify trouble, such as TCP retransmission, duplicate ACKs, and zero windows.
- **Relative sequence numbers (Requires "Analyze TCP sequence numbers"):** When used, this feature helps make the sequence numbers easier to read and compare. The relative sequence numbers start with zero for the first packet in each stream, and then increment from that point.
- **Scaling factor to use when not available from capture:** Window Scale is used to increase the maximum WS that is allowed. This option was covered in detail in *Chapter 9, Decoding TCP and UDP*.
- **Track number of bytes in flight:** To see the bytes in flight, in `Flow312.pcapng`, use the `tcp.analysis.bytes_in_flight` display filter, which will result in two frames. Select **Frame 5** and expand the SEQ/ACK analysis to see the bytes in flight, as shown here:

```

^ [SEQ/ACK analysis]
  [This is an ACK to the segment in frame: 4]
  [The RTT to ACK the segment was: 0.090943000 seconds]
  [iRTT: 0.026754000 seconds]
  [Bytes in flight: 936]
  [Bytes sent since last PSH flag: 936]

```

Figure 10.22 – Calculating TCP bytes in flight

- **Calculate conversation timestamps:** This option will monitor time values and can help to find delays during TCP conversations.
- **Try heuristic sub-dissectors first:** This option helps Wireshark attempt to identify what type of application is used by using the port number to properly dissect the packet. This is done according to the *behavior* exhibited, and what Wireshark *believes* is the appropriate protocol.
- **Ignore TCP timestamps in summary:** Wireshark obtains the timestamp from the operating system kernel. Use this option if you feel the timestamp may not be accurate.

- **Fast Retransmission supersedes Out-Of-Order interpretation:** On a busy network, packets get lost and delays occur. With this option, you can modify the way Wireshark prioritizes during analysis, and characterize a Fast Retransmission to be interpreted before any Out-Of-Order packets.
- **Do not call subdissectors for error packets:** Wireshark does its best to properly dissect each protocol according to the RFC. In some cases, the dissector may have incorrectly identified an error. Therefore, it's best to check this option so that Wireshark does not continue to incorrectly dissect the packet and throw more errors.
- **TCP experimental options with a Magic Number:** In some cases, the capture may include a conversation where the TCP option is experimental, possibly used for testing. Because the option is experimental and not a standard, Wireshark needs to use a magic number to identify the option, so it can be properly dissected.
- **Display process information via IPFIX: Internet Protocol Flow Information Export (IPFIX)** is a format used to analyze network traffic. When selected, Wireshark will display the process information that can be used to analyze and troubleshoot IPFIX flows.

**Note**

For any of the options that change the default values, use caution! What you enter may *stick* and may not allow you to undo the option without a reinstall.

During an analysis, Wireshark will attempt to investigate all aspects of a protocol's behavior. As you can see, it's possible to personalize your preferences and tailor the way TCP is interpreted.

This final section provides an overview of the TCP teardown, which properly closes the connection between two endpoints.

## Tearing down a connection

When a TCP connection is complete, TCP tears down the connection by exchanging a series of FIN packets, closing the port, and refusing any more requests to communicate. Let's walk through the entire process.

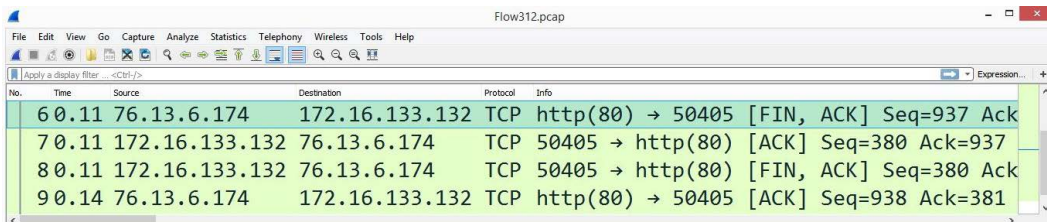
When two hosts are communicating, a TCP conversation goes through several stages:

- TCP starts with a (three-way) handshake to set up the session. In many cases, there are additional header options that further define the parameters.

- During the conversation, TCP monitors the communication and acknowledges all bytes received to ensure the complete delivery of the data.
- Once the conversation is over, TCP ends the session with an exchange of FIN packets between the two endpoints, which indicates that the session is complete.

Let's now take a look at how session teardown is represented in Wireshark.

In the `Flows312.pcapng` capture, packets 6, 7, 8, and 9 represent the session teardown, as shown here:



No.	Time	Source	Destination	Protocol	Info
6	0.11	76.13.6.174	172.16.133.132	TCP	http(80) → 50405 [FIN, ACK] Seq=937 Ack
7	0.11	172.16.133.132	76.13.6.174	TCP	50405 → http(80) [ACK] Seq=380 Ack=937
8	0.11	172.16.133.132	76.13.6.174	TCP	50405 → http(80) [FIN, ACK] Seq=380 Ack
9	0.14	76.13.6.174	172.16.133.132	TCP	http(80) → 50405 [ACK] Seq=938 Ack=381

Figure 10.23 – The four-packet FIN exchange

To close the session, TCP uses a FIN flag, as shown in the following screenshot, which indicates that there is no more data to be sent:

```

Flags: 0x011 (FIN, ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
.... .... ..0. = Syn: Not set
.... .... ...1 = Fin: Set

```

Figure 10.24 – The TCP FIN flag set

To completely close a connection, TCP progresses from an established state to FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and then CLOSED, as stated in RFC 793.

**Note**

Some applications (such as Outlook and Exchange) will close a TCP connection with **Reset (RST)** and FIN to try and prevent port exhaustion.

TCP will wait until both sides have said their final goodbyes and have sent a FIN packet, and then the operating system will close the socket. Any future attempts at communicating will be refused.

## Summary

An important concept in establishing a connection-oriented session is to outline the parameters of the conversation before any data is exchanged. In this chapter, we studied how TCP begins a conversation by using a three-way handshake and then took a closer look at each step of the handshake. We saw how, once the handshake is complete, the operating system creates a socket so that data exchange can take place.

In addition, we reviewed the TCP options that are exchanged during the three-way handshake, such as SACK, MSS, and timestamps. This chapter also explained the TCP protocol preferences and outlined how you can modify the TCP preferences in Wireshark. We then finished by seeing how TCP ends a session by exchanging FIN packets that signal each host to close the session.

IP is the other dominant protocol in the TCP/IP suite. In the next chapter, we will take a closer look at IPv4 and IPv6. So that you have a better understanding of this network layer protocol, we'll begin with a thorough overview of IPv4 and examine the header format along with each of the field values. We will then take a look at IPv6 along with the corresponding header format and the field values. In addition, because IPv4 is a completely different format from IPv6, we will address how the two can coexist by using various tunneling protocols when in a dual-stack environment.

## Questions

Now it's time to check your knowledge. Select the best response, and then check your answers, which can be found in the *Assessments* appendix:

1. To filter only in packets that you have marked in Wireshark, use \_\_\_\_\_ in the display filter.
  - A. `marked:all`
  - B. `frame = black`

- C. `frame.marked==1`
  - D. `marked: on`
2. \_\_\_\_ is used to increase the maximum WS that is allowed.
- A. NOP
  - B. Window scale
  - C. Timestamp
  - D. SACK
3. When using \_\_\_\_, the receiver will notify the sender if there are any missing packets.
- A. NOP
  - B. Window scale
  - C. Timestamp
  - D. SACK
4. TCP ends the session by exchanging packets indicating that each side should close their respective socket. TCP uses the \_\_\_\_ flag to indicate the end of a conversation.
- A. END
  - B. SYN
  - C. FIN
  - D. URG
5. If, in the TCP header, the sequence number is 1 and the next sequence number is 937, the packet has \_\_\_\_ bytes of data.
- A. 32
  - B. 380
  - C. 936
  - D. 33,304

6. When using TCP options, \_\_\_\_\_ is a single byte used at the end of the options.
  - A. EOL
  - B. SACK
  - C. NOP
  - D. WS
  
7. Using the \_\_\_\_\_ option, TCP can monitor the round-trip times. This is so the sending host can retransmit a packet if it does not receive an ACK in a timely manner.
  - A. EOL
  - B. SACK
  - C. NOP
  - D. Timestamps

## Further reading

Please refer to the following links for more information:

- The TCP three-way handshake is outlined in detail in the original RFC 793 found at <https://tools.ietf.org/html/rfc793>.
- Read more on SACK by visiting <https://packetlife.net/blog/2010/jun/17/tcp-selective-acknowledgments-sack/>.
- Read more about how Wireshark helps troubleshoot by providing advanced analysis of the data transfer by visiting [https://www.wireshark.org/docs/wsug\\_html\\_chunked/ChAdvTCPAnalysis.html](https://www.wireshark.org/docs/wsug_html_chunked/ChAdvTCPAnalysis.html).

# 11

# Analyzing IPv4 and IPv6

Anyone involved in networking will need to understand the **Internet Protocol (IP)**, a network layer protocol that is responsible for delivering data over a network. In this chapter, we'll begin by reviewing the network layer, and more specifically, the purpose of IP as a predominant network protocol. We'll then take a closer look at IPv4 and IPv6, which are responsible for two key roles: addressing and routing data. Wireshark provides exceptional support for both IPv4 and IPv6. To strengthen your analytical skills, we'll start with a thorough overview of both versions and examine the header format of each protocol.

You'll begin to understand how the field values in each of the versions compare and contrast, and be able to recognize the significance of each of the fields. Since addressing is an important concept, we'll examine the use of special and private IPv4 addressing. In addition, we'll outline the different address types used in IPv6. So that you can learn how to customize how Wireshark displays IP, we'll evaluate the protocol preferences. Finally, because IPv4 has a completely different header format than IPv6, we'll investigate how the two can coexist by using various tunneling protocols when in a dual-stack environment.



This chapter will address all of this by covering the following:

- Reviewing the network layer
- Outlining IPv4
- Exploring IPv6
- Editing protocol preferences
- Discovering tunneling protocols

## Reviewing the network layer

The network layer (or layer 3) has two key roles: addressing and routing data. This layer provides addressing using a logical IP address. In addition, the network layer determines the best logical path to take for packets that travel through other networks so they can get to their destination. It does this by communicating with other devices during the routing process.

As shown in the figure, the network layer has three main protocols, IP, ARP, and ICMP, which are essential in delivering data:

### OSI Model

Layer	Name	Role	Protocols	PDU	Address
7	Application	Initiate contact with the network	HTTP, FTP, SMTP	Data	
6	Presentation	Formats data, optional compression and encryption		Data	
5	Session	Initiates, maintains, and tears down a session		Data	
4	Transport	Transports data	TCP, UDP	Segment	Port
3	Network	Addressing, routing	IP, ICMP, ARP	Packet	IP
2	Data Link	Frame formation	Ethernet II	Frame	MAC
1	Physical	Data is transmitted on the media		Bits	

Figure 11.1 – The OSI model—network layer

In addition to IP, the other protocols include the following:

- **Address Resolution Protocol (ARP)**: ARP resolves an IPv4 address (network layer) to a **Media Access Control (MAC)** (data link layer) address on a local area network so the frame can be delivered to the appropriate host. ARP is shown between layer 3 and layer 2, and as a result, many consider ARP as a layer 3 protocol.
- **Internet Control Message Protocol (ICMP)**: This reports on issues encountered during transit, such as network unreachable or host unreachable.

Next, we'll discuss the role and purpose of IP and how it helps packets find their way when having to pass through networks to reach their final destination.

## Understanding the purpose of IP

IP is a network layer protocol that has two key roles: addressing, using a logical IP address, and routing traffic. While the transport layer *transports* the data, the network layer communicates with other devices to determine the *best logical path* for the packets.

One of the main protocols in the network layer is IP, which provides a best-effort, connectionless service, as outlined here:

- **Best-effort** means that there is no guarantee the data will be delivered. It's similar to mailing a letter using general delivery. Although some mail is lost, most of the time it reaches its final destination.
- **Connectionless** means that IP does not retain any state information; that process is left to the higher-level protocols, such as the **Transmission Control Protocol (TCP)**.

Although IP can't guarantee delivery, it can *prioritize* traffic, so that time-sensitive data, such as **Voice over IP (VoIP)** and streaming media, is delivered at a higher precedence than email or web pages.

Because of the unpredictable nature of the internet, prioritizing certain types of traffic will help the data to be delivered faster. The priority is marked in the following ways:

- In **IPv4**, by using the **Differentiated Services (DiffServ)** field
- In **IPv6**, by using the **Traffic Class** field

To examine the IP headers in detail, we will use the `bigFlows.pcap` packet capture found at <http://tcpreplay.appneta.com/wiki/captures.html#bigflows-pcap>. Download the file and open it in Wireshark.

Let's begin with an evaluation of IPv4.

## Outlining IPv4

In 1981, **Request for Comments (RFC)** 791 outlined the specifications for IPv4. The RFC outlined that IPv4 had two principal tasks, addressing and fragmentation, as defined in *Section 1.4. Operation*, found at <https://tools.ietf.org/html/rfc791#section-1.4>.

As stated, one of the original roles of IPv4 was fragmentation, which breaks packets apart. At the time, this was necessary because, in the early 1980s, most of the networks had limited bandwidth and were unable to transmit large packets.

Over time, efforts have been made to upgrade and replace the antiquated data pathways, and much of the internet has been replaced by high-speed, fiber optic cables. As a result, in today's networks, fragmentation is rarely used.

### Note

Fragmentation is used when the **maximum transmission unit (MTU)** is less than 1,500 bytes.

As time has passed, we can see that IPv4 is still influential in addressing, along with the role of routing, in order to get data to its final destination.

IPv4 was standardized in 1983 and uses a 32-bit address space. Scientists identified at an early stage the need for a larger address space. IPv6 has a 128-bit address space and provides enhancements to the protocol in general, such as simplified network configuration and more efficient routing. There is a slow migration to IPv6, mainly because the use of private IP addressing on a **local area network (LAN)** has extended IPv4's lifespan.

As a result, IPv4 is still widely used. So that you have the skills required to face everyday network-related issues when dealing with IPv4, in the next section, we'll examine the IPv4 header and each of the field values. Once you understand the field values, you will be more confident when looking at a packet capture, so you will be able to quickly drill down to the issue.

## Dissecting the IPv4 header

The IPv4 header has several fields, as shown in the following diagram:

IPv4 Header			
Version	IHL	DiffServ	Total Length
Identification		Flags	Fragment Offset
Time to Live		Protocol	Header Checksum
Source Address			
Destination Address			
Options and Data			

Figure 11.2 – IPv4 header

Some of the fields are rarely used, such as those that deal with fragmentation. Others provide information that can help with troubleshooting, such as the address fields when resolving network conflicts.

To examine an IPv4 header, open `bigFlows.pcap`, and go to **Frame 1**, as shown here:

```

4 Internet Protocol Version 4, Src: 172.16.133.57, Dst: 68.64.21.62
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1154
    Identification: 0xfd44 (64836)
  ▶ Flags: 0x0000
    Time to live: 64
    Protocol: UDP (17)
    Header checksum: 0xee5e [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.16.133.57
    Destination: 68.64.21.62

```

Figure 11.3 – bigFlows Frame 1—IPv4 header

The following section will list each field, how many bits or bytes are used, along with information on what each field represents. We'll start with the version and length fields.

## Discovering the version and the length

The first two fields in an IPv4 header are as follows:

- **Version (4-bit):** This field value indicates the version of IP that is in use. Many devices support both IPv4 and IPv6; therefore, it's important to obtain the version, so the device knows how to treat the traffic. In **Frame 1**, we see `Version: 4`, which means this is an IPv4 header.
- **Header length (4-bit):** The header length is in multiples of 4 bytes and is equal to the base header and any options. Although the length can vary (due to options), the *minimum* value must be five, which equals a header length of 20 bytes. In **Frame 1**, the value shows `Header Length: 20 bytes (5)`.

After these two fields, we see a field called `DiffServ`, which is covered in the next section.

## Breaking down the type of service

The internet can be unpredictable, and this can affect time-sensitive data, such as VoIP and streaming media. As a result, IPv4 has the `DiffServ` field to prioritize time-sensitive traffic, so that it is delivered at a higher precedence than email or web pages.

The `DiffServ` field is 8 bits and separated into two functions: **Quality of Service (QoS)** and **Explicit Congestion Notification (ECN)**.

In **Frame 1**, we can see `Differentiated Services Field: 0X00 (DSCP: CS0, ECN: Not-ECT)`.

As noted, the `DiffServ` field has two functions. Let's go through what this represents. We'll start with the first 6 bits of the `DiffServ` field, which is used to represent the QoS requested when traveling through a network.

## Ensuring QoS

QoS provides options to prioritize traffic. Most, but not all, devices support QoS. When the priority is requested, the field value will indicate this by using one of several different **class selector (CS)** values.

For example, `CS6` and `CS7` are used for network control protocols that include the following:

- **Enhanced Interior Gateway Routing Protocol (EIGRP)**
- **Open Shortest Path First (OSPF)**
- **Hot Standby Router Protocol (HSRP)**

- **Internet Key Exchange (IKE)**
- **Border Gateway Protocol (BGP)**

These network protocols are prioritized at a higher level, as any delays will impact network performance.

Other values are as shown in the following table:

DSCP	Binary	Decimal	Application	Uses
CS0	000 000	0	DEFAULT	Best effort traffic
CS1	001 000	8	Scavenger	YouTube, gaming, P2P
CS2	010 000	16	OAM	SNMP, SSH, Syslog
CS3	011 000	24	Signaling	SCCP, SIP, H.323
CS4	100 000	32	Real-time	Telepresence
CS5	101 000	40	Broadcast video	Cisco IPVS
CS6	110 000	48	Network control	EIGRP, OSPF, HSRP, IKE, BGP
CS7	111 000	56		

Table 11.1 – Differentiated services field values

The first column shows the **Differentiated Services Code Point (DSCP)**, which lists the CS. As shown in **Frame 1**, in *Figure 11.3*, this field value summary shows DSCP: CS0 (or Class Selector 0). CS0 is the default or best-effort setting, in that there is no priority assigned to this packet. Traffic with this setting is delivered normally.

To see an example of a CS that is higher than the best-effort, go to `bigFlows.pcap` and enter the `ip.dsfield.dscp > 0` display filter. Select **Frame 4**, where you will see the CS value listed, as shown here:

```

^ Differentiated Services Field: 0x20 (DSCP: CS1, ECN: Not-ECT)
  0010 00.. = Differentiated Services Codepoint: Class Selector 1 (8)
  .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)

```

Figure 11.4 – CS1

Class Selector 1 (8) is used in scavenger applications such as YouTube, gaming apps, and P2P, as this traffic would benefit from having a (slightly) higher priority when traveling over the internet.

**Note**

CS1 is listed as a scavenger application. This class means the application will *grab* bandwidth whenever it is available

The last 2 bits of the `DiffServ` field are used to identify ECN, which helps to manage congestion on the network. Let's see how this works.

## Sending an ECN

You may not have been aware of ECN and its significance; however, this can have an impact on how devices communicate congestion on the network. Let's take a look at how these 2 bits can improve data flow.

In the original RFC 791, the last 2 bits of the `DiffServ` field were Reserved for Future Use, as shown in the following screenshot:

```
Bits 0-2: Precedence.
Bit   3: 0 = Normal Delay,      1 = Low Delay.
Bits  4: 0 = Normal Throughput, 1 = High Throughput.
Bits  5: 0 = Normal Reliability, 1 = High Reliability.
Bit  6-7: Reserved for Future Use.
```

Figure 11.5 – Service bit assignments

In 2001, RFC 3168 found a use for the last 2 bits. RFC 3168 outlined ECN, which provides a congestion notification on the network and is an improvement over the classic method of managing network congestion.

Typically, when TCP experiences congestion, the hosts respond to dropped packets by going into congestion control, which results in the following:

- **The client** sends duplicate acknowledgments, indicating that there are missing packets.
- **The server** uses fast retransmission, which resends lost packets.

ECN is an improvement over this behavior by providing a notification that there is congestion on the network. This ultimately prevents the additional traffic that occurs when there are both duplicate acknowledgments and fast retransmissions.

ECN uses both the TCP and IP headers, as outlined here:

- **The IP header** uses the 2 bits at the end of the `DiffServ` field to indicate **ECN-Capable Transport (ECT)** and **Congestion Experienced (CE)**.
- **The TCP header** uses two flags: **Congestion Window Reduced (CWR)** and **ECN Echo (ECE)**.

When using ECN, the 2 bits of the `DiffServ` field identify the code point. In the following table you'll see the bits, what the combination indicates, and what you might see in Wireshark as an indicator when displaying the `DiffServ` field values:

ECN field	Indication	Identifier
00	Non-ECN-Capable Transport	Non-ECT
01	ECN-Capable Transport	ECT(1)
10	ECN-Capable Transport	ECT(0)
11	Congestion Experienced	CE

Table 11.2 – Identifiers in the `DiffServ` field

As you can see, code points with the values of 10 and 01 are basically the same.

In `bigFlows.pcap` under **Frame 1**, if we expand the IPv4 header, we see `Explicit Congestion Notification: Not ECN-Capable Transport (0)`, which means this connection doesn't support ECN, as shown here:

```

• Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  0000 00.. = Differentiated Services Codepoint: Default (0)
  .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)

```

Figure 11.6 – Not ECT-Capable

The devices involved in the connection will communicate with one another and, when available, use ECN, which helps notify endpoints on network congestion issues.

Although IP is a connectionless protocol, it provides methods to improve the priority of traffic, along with ways of notifying devices of congestion issues on the network.

In the next group of field values in the IPv4 header, we'll evaluate the fields that deal with using fragmentation.

## Fragmenting the data

In RFC 791, IP was responsible for addressing and fragmentation. We'll discuss addressing in a later section, but for now, let's outline what fragmentation is and why it may be necessary.

On the network, various values are monitored:

- The **maximum segment size (MSS)** is the data payload.
- The **maximum transmission unit (MTU)** is the MSS plus the transport layer headers.



When data is routed on the network, it may encounter a segment with an MTU that is smaller than the packet size. If allowed, fragmentation can be used; this divides a datagram into smaller pieces so that they can be sent on the network with a restrictive MTU.

The following three fields are related to fragmentation: **identification**, **flags**, and **fragment offset**, as described here:

- **Identification (16-bit)**: This **identification (ID)** field is used to identify the datagrams when data is fragmented. In that case, all fragments will have the same ID.
- **Flags**: In an IP header, there are three flags, as shown in the following screenshot:

```
Flags: 0x0000
0... .. = Reserved bit: Not set
.0.. .. = Don't fragment: Not set
..0. .. = More fragments: Not set
...0 0000 0000 0000 = Fragment offset: 0
```

Figure 11.7 – IP flags

- **Fragment offset (13-bit)**: After the three flags in the IP header, this field provides information on how to reassemble the fragments when using fragmentation.

In most cases, the IP header flags will be set at Don't Fragment. That is because in today's networks, fragmentation is not used as most pipelines have generous bandwidth with an acceptable MTU.

Although on today's networks, we rarely see fragmentation, it's a good idea to become familiar with the fields and flags dealing with fragmentation for a couple of scenarios:

- During *troubleshooting*, you may need to look at the fields when determining why data may not be getting through.
- During a *security assessment*, the use of the fragmentation fields could be an indication of malicious activity.

Network devices monitor datagram lengths and may impose size restrictions. In that case, if the packet is too large, it may have to be fragmented or rerouted in order to be delivered.

The **total length** IPv4 field provides a metric when evaluating size restrictions, as it indicates the value of the header length and any data. The field value is 16 bits, which means the entire length cannot exceed  $2^{16}$ , or 65,535 bytes.

**Internet Control Message Protocol (ICMP)** acts as a scout for IP. When ICMP encounters a network with an MTU that is *smaller* than the size of the packet, and when the Don't Fragment bit is set, the router will drop that packet. ICMP will then notify the source by sending a Type 3 Code 4 ICMP message: Destination Unreachable: Fragmentation Needed and Don't Fragment was Set.

If the packet is dropped on a network segment with restrictive bandwidth that doesn't allow fragmentation, the sending host must retransmit the data using a smaller MSS.

The next few fields are more administrative. They hold values related to the number of hops, the protocol that follows the IPv4 header, and the checksum, which is used for error detection.

## Viewing TTL, protocol, and checksum

When looking at the IPv4 header, there are a few fields that are not directly related to routing or addressing packets but provide a role that may influence other types of behavior. Our first example is the **Time to Live (TTL)** field, which exists because the fathers of the internet realized early on that there must be a way to stop a packet from continually traveling through the network. This can happen if there is a misconfiguration and/or the packet is in a routing loop.

### Note

The TTL field is 8 bits, so the maximum value is  $2^8$ , or 255 hops.

During regular network operations, this most likely won't happen. However, in case there is a routing loop, the TTL field value in an IP header is the number of routers or hops a packet can take before dropping the packet. The TTL works in the following manner:

- Every time the packet reaches a router, the number decrements by 1.
- When the TTL value reaches 0, the packet is dropped and an ICMP Type 11 (TTL expired in transit) is sent to the sender.

To see an example of a TTL value, open `bigFlows.pcap`, and go to **Frame 1**, where the TTL field is set at `Time to live: 64`, which is the default value for this field.

Two other fields that provide information related to managing traffic and informing devices are as follows:

- **Protocol (8-bit):** The `protocol` field identifies the higher-layer protocol that follows the IPv4 header. The field identifies the protocol (which is usually a transport layer protocol) that is carried in the datagram. In **Frame 1**, we see the value as `Protocol: UDP (17)`.
- **Header checksum (16-bit):** This field is used to house the checksum value. Similar to the checksum in the TCP header, this value is used for error detection. In **Frame 1**, we see the checksum and notification from Wireshark that the checksum validation is disabled.

```
Header checksum: 0xee5e [validation disabled]
[Header checksum status: Unverified]
```

In most cases, it's best to disable validation as the value will be incorrect due to the checksum offloading to the **network interface card (NIC)**.

When dealing with IPv4, there are additional considerations when dealing with addressing, such as special and private IP addresses, as we'll discuss in the following section.

## Learning about IPv4 addressing

In this section, we'll examine the last two fields in an IPv4 header. In addition, we'll review the different classes in IPv4, along with an overview of special and private IP addresses.

One of the more significant elements in the IP header is addressing. In the last two fields, we see the **source** and **destination address** (32 bits). Each field houses the source or destination IPv4 address, which is represented in an easy-to-understand dotted decimal format. In **Frame 1** we see the following values:

```
Source Address: 172.16.133.57
Destination Address: 68.64.21.62
```

IPv4 segments the entire address block into classes. Within each class, there are special and private IP addresses. Let's take a look at those concepts.

## Comparing IPv4 classes and addresses

When the RFC for IPv4 was written, developers had a concept to subdivide IP into five classes or formats of addresses. IPv4 addresses are divided into classes A-E, as shown here:

Class	Range	Use
A	0.0.0.0 to 127.255.255.255	Assignable (to companies)
B	128.0.0.0 to 191.255.255.255	Assignable (to companies)
C	192.0.0.0 to 223.255.255.255	Assignable (to companies)
D	224.0.0.0 to 239.255.255.255	Multicast
E	240.0.0.0 to 255.255.255.255	Experimental

Table 11.3 – Classes of IPv4 addresses

As outlined, classes A, B, and C are assigned mainly to companies. Class D is for **multicast** only, and class E is **experimental**, and not used.

In addition to having five classes of addressing, IPv4 has several ranges of special and private IPv4 addresses, as outlined next.

## Reviewing special and private IP addressing

One of the limitations of IPv4 is the restrictive address space. To extend the use of IPv4, a group of private IP addresses for classes A, B, and C were drafted, to be used only on an internal network. There is also a generous loopback range and a broadcast IP address.

The following table shows a list of the predominant special and private IPv4 addresses:

Purpose	Range
Class A Private IP	10.0.0.0 - 10.255.255.255
Class B Private IP	172.16.0.0 - 172.31.255.255
Class C Private IP	192.168.0.0 - 192.168.255.255
Loopback range	127.0.0.0 - 127.255.255.255
APIPA	169.254.0.0 - 169.254.255.255
Broadcast	255.255.255.255

Table 11.4 – Special and private IPv4 addresses

In addition, there is a range for **Automatic Private IP Addressing (APIPA)**, which gives a host an IP address when one isn't available from the **Dynamic Host Configuration Protocol (DHCP)** server.

While it is rare, options for IPv4 may be used, as discussed in the following section.

## Modifying options for IPv4

With IPv4, it may be necessary to use options that provide source routing information, timestamps, and others. Several of the IP options have been deprecated and are no longer used. For a more complete discussion on formally deprecated options, refer to RFC 6814.

When used, the `options` field must be a multiple of 32 bits, or 4 bytes. Padding may be required so that the header is a multiple of 32 bits.

Now that we have reviewed IPv4, let's take a closer look at IPv6.

## Exploring IPv6

Early on, scientists realized that IPv4's 32-bit address space would be exhausted. Although no one had an exact date, plans were made to replace IPv4 with an improved version, IPv6. In 1998, the RFC for IPv6 was published and can be found at <https://www.ietf.org/rfc/rfc2460.txt>.

IPv6 has a number of enhancements, including the following.

- **Streamlined header:** The header has fewer fields; however, it is larger, mainly due to the expanded address space.
- **Flow label:** In IPv6, there is a flow label. The field value is available for identifying streams that require specialized treatment, such as real-time traffic.

- **Support for extensions and options:** While IPv4 can add options, IPv6 does so with greater ease. IPv6 provides the ability to add options, such as fragmentation, which has parameters to fragment the data, and hop-by-hop, which ensures that all devices in the path read the option.

The IPv6 header has room for larger address spaces. However, as shown in the following diagram, the header is streamlined, in that there are not as many field values:

IPv6 Header		
Version	Traffic Class	Flow Label
Payload Length	Next Header	Hop Limit
Source Address		
Destination Address		

Figure 11.8 – IPv6 header

To follow along and examine an IPv6 header, open `bigFlows.pcap`, and go to **Frame 347**. The IPv6 header is as shown in the following screenshot:

```

Internet Protocol Version 6
  0110 .... = Version: 6
  .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
  .... 0000 0000 0000 0000 0000 = Flow Label: 0x000000
  Payload Length: 106
  Next Header: UDP (17)
  Hop Limit: 1
  Source: fe80::9186:dbbd:2a45:50c2
  Destination: ff02::1:2

```

Figure 11.9 – bigFlows frame 347—IPv6 header

Note that IPv6 addresses are significantly larger as they are 128-bit, as opposed to 32-bit for an IPv4 address. The address is shown using hexadecimal notation, as opposed to dotted-decimal notation, which is used in IPv4.

In the next section, we'll review each field in IPv6 and the number of bits or bytes each field contains, along with information on what each field represents.

## Navigating the IPv6 header fields

As we'll see, the IPv6 header removes unnecessary field values and adds only what is needed to transport data. Let's step through the field values and learn their significance. We'll start with the version, a field to house the traffic class, and a label dedicated to holding a value for a specific flow.

## Identifying the version, traffic class, and flow label

The first three fields in an IPv6 header are as follows:

- **Version (4-bit):** This field indicates the IP version that is in use. In **Frame 347**, we see `Version 6`.
- **Traffic class (8-bit):** When sending data on the internet, some traffic requires special handling and prioritization. Similar to IPv4, this field houses two values:
  - **TOS:** The first 6 bits of this field are used to communicate what type of service is requested. TOS uses the same DSCP values as IPv4. **Frame 347** uses the `Differentiated Services Codepoint: Default (0)` default value.
  - **ECN:** The last 2 bits of this field are used to indicate congestion on the network in the same way as IPv4. In **Frame 347**, this value is `Explicit Congestion Notification: Not ECN-Capable Transport (0)`.
- **Flow label (20-bit):** This field can be used to identify a specific flow of information in order to provide sequencing or request special handling by routers in the path. In **Frame 347**, we can see `Flow Label: 0x00000`. In RFC 2460, *Appendix A: Semantics and Usage of the Flow Label Field*, there is an expanded discussion on the flow label.

As time has passed, we are seeing more use of the flow label. Let's discuss how this is being used, along with examining a populated flow label.

## Managing the flow

The label can be used to prioritize traffic, such as real-time data (voice and video), but it can be used for other reasons as well. When used, a randomly assigned number is attached to the flow label, and then all traffic will belong to the same flow or stream.

To see an example of a flow label in use, run a capture on your network, and gather about 1,000 packets. Apply the `(!(ipv6.flow == 0x00000)) && (ipv6)` filter, which will show only IPv6 packets with a populated flow label.

In addition, you can go to <https://wiki.wireshark.org/SampleCaptures#ipv6-and-tunneling-mechanism>. Select the `sr-header.pcap` file and open it in Wireshark.

In this capture, there are 10 packets. Use the `ipv6.flow == 0xd684a` filter, which will display six packets that are all part of the same flow.

**Note**

The flow label is a 20-bit field. In Wireshark, you will see the bit values first followed by the hexadecimal value (identified by using 0x before the value). For example, if the field is populated, you will see the following value in Wireshark: . . . 1101 0110 1000 0100 1010 = Flow Label: 0xd684a.

The next three fields deal with similar values found in an IPv4 header, but have subtle differences, as shown next.

## Evaluating the length, next header, and hop limit

In an IPv6 header, the next three fields provide information on the length of the payload, the protocol that follows the IP header, and how many hops the packet can take before going away. The fields are as follows:

- **Payload length (16-bit):** The payload length represents the packet's payload, which includes higher-layer headers, data, and any extension headers. Similar to IPv4, the entire length cannot exceed  $2^{16}$ , or 65,535 bytes. In some cases, the payload may exceed 65,535 bytes, which can occur when using extension headers. If the value of this is greater than 65,535 bytes, the field value is set to zero (0).
- **Next header (8-bit):** This field identifies the higher-layer protocol that follows the IP header. This is similar to the `protocol` field in IPv4 and uses the same values to identify the higher-layer protocol. However, if there is an extension header, this field will indicate what extension header follows the IPv6 header.
- **Hop limit (8-bit):** In IPv4, the `TTL` field value in an IP header is the number of routers or hops a packet can take before dropping the packet. In IPv6, this is the same concept. However, the field is more reflective of what it does today. This field uses 8 bits, which can hold a value not greater than 255. If the hop limit reaches 0, the packet is discarded.

In **Frame 347**, the field value is `Hop Limit: 1`, which makes sense as this frame is DHCPv6 multicast from a host trying to get an IP address.

As with IPv4, the last two fields in an IPv6 header are the address fields, as discussed next.



## Examining IPv6 addresses and address types

IPv6 has specific addressing requirements. In this section, we'll examine the last two fields, **source** and **destination address (128-bit)**, along with an overview of IPv6 address types. The source and destination addresses are 128-bit fields to accommodate the IPv6 address. Wireshark displays the address in hexadecimal numbers separated by colons, as opposed to dotted-decimal notation, which is used in IPv4.

With IPv6, there are various address types, as opposed to classes. Let's now take a look at the different types you may encounter.

## Comparing IPv6 address types

IPv6 does not use a broadcast as in IPv4. However, there are several types of addresses, as listed here:

- **Global unicast** is like a public IPv4 address. The address is globally recognized and can be routed on the internet.
- **Link local** is used to communicate with hosts on the same sub-network. This address always starts with FE80.
- **Unicast** is a single host on a network.
- **Multicast** packets are delivered to all nodes on a network using a single multicast address.
- **Anycast** is used to send data to multiple locations with the same IP address. The packets are delivered to the closest (or nearest) destination.

In **Frame 347**, we see the source and destination addresses:

```
Source: fe80::9186:dbbd:2a45:50c2
Destination: ff02::1:2
```

When possible, Wireshark will use appropriate shortcut methods, as shown in the destination address. An IPv6 shortcut removes leading zeros and collapses two or more blocks that contain consecutive zeros.

For many protocols (but not all), Wireshark provides a means to modify the way in which Wireshark presents the data. The following gives us some insight into how to adjust preferences for both IPv4 and IPv6.

## Editing protocol preferences

In Wireshark, you can modify most protocols by doing either of the following:

- Right-clicking while on the IP header and selecting **Protocol Preferences**, where you will see a list of preferences
- Going to **Edit | Preferences | Protocols**, and then selecting the appropriate protocol

Let's start with the protocol preferences for IPv4, as this is currently the most commonly used protocol on a LAN today.

### Reviewing IPv4 preferences

To modify IPv4 preferences, you can use one of the methods listed previously, or you can right-click while on the header and select **Protocol Preferences**, and then select the **Open Internet Protocol Version 4 preferences...** shortcut, as shown here:

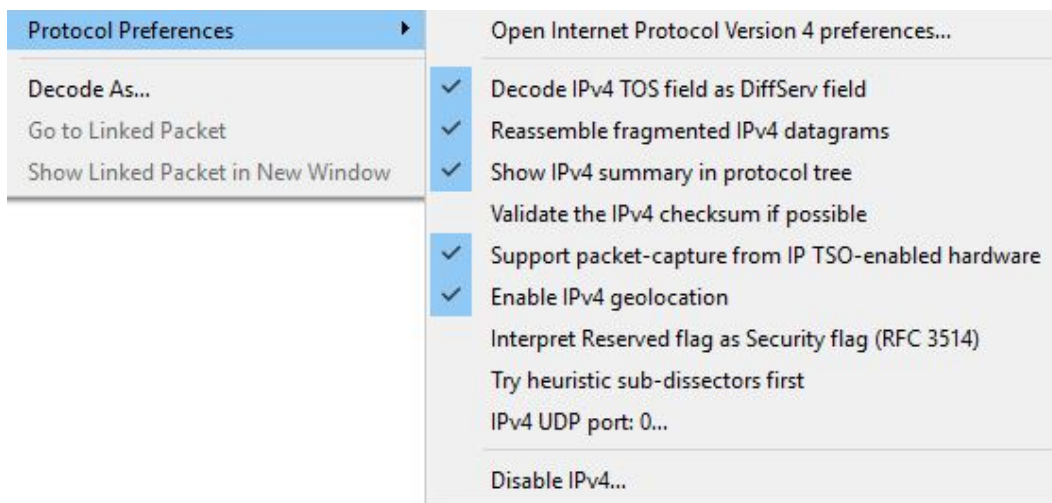


Figure 11.10 – IPv4 preference shortcut

Once you select the shortcut, a list of preferences will be listed, as shown in the following screenshot:

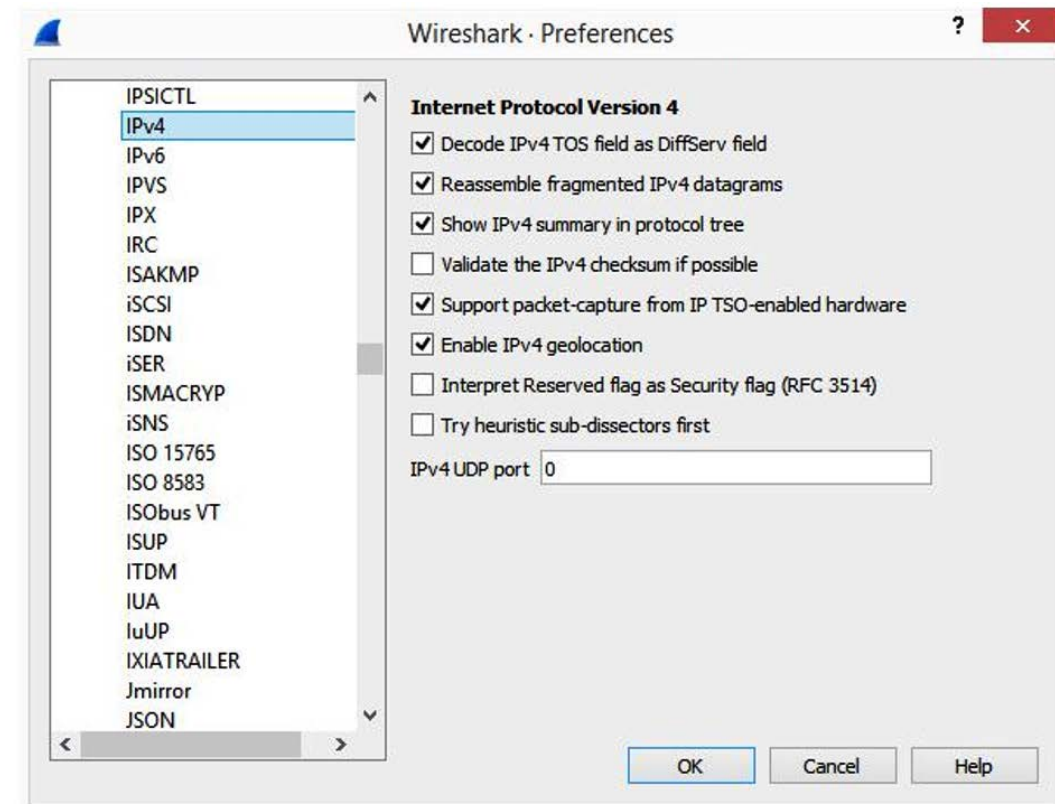


Figure 11.11 – IPv4 preferences

Once there, you can modify the selections as follows:

- **Decode IPv4 TOS field as DiffServ field:** RFC 791 used TOS to classify traffic. Over time, this field was modified to identify traffic using `DiffServ`, which allows for a wider range of classification. In most cases, this should be enabled.
- **Reassemble fragmented IPv4 datagrams:** When necessary, IPv4 packets may be fragmented. When enabled, this will reassemble fragmented IP datagrams.

- **Show IPv4 summary in protocol tree:** When enabled, this summarizes the header contents. For a large capture, enabling this may impact performance.
- **Validate the IPv4 checksum if possible:** In most cases, this is not enabled.
- **Support packet-capture from IP TSO-enabled hardware: TCP Segmentation Offload (TSO)** is a performance-boosting technique used in a virtualized environment. When used, the packet length may be inaccurate. Enabling this option will attempt to correct any errors.
- **Enable IPv4 geolocation:** Wireshark uses the IP addresses to identify packet origin using the [GeoIP] databases. Select if you want to use this option.
- **Interpret Reserved flag as Security flag (RFC 3514):** On April 1, 2003 (April Fool's Day), Steven M. Bellovin wrote an RFC that the reserved bit in the IP header should be used by malicious actors to flag the packet if it contains malware, so that IDS and firewalls will know it contains malware. If used, the bit is called the **evil bit**.
- **Try heuristic sub-dissectors first:** This option helps Wireshark attempt to identify what type of application is used by using the port number to properly dissect the packet.
- **IPv4 UDP port: 0...:** Use this option if you want to change the protocol behavior to a specific port when used on the LAN.
- **Disable IPv4:** Use this option (as shown in Figure 11.10) if you want to disable IPv4 and focus only on the frame headers. Keep in mind if this option is selected, IPv4 and any higher-level protocols (Layers 3-7) will not be dissected.

As you can see, there are many ways to customize the preferences for IPv4. Next, let's take a look at the options for IPv6.

## Adjusting preferences for IPv6

You can modify the preferences in IPv6 by going to **Edit | Preferences** and then selecting the **Open Internet Protocol Version 6 preferences...** shortcut. This will open a dialog box as shown here:

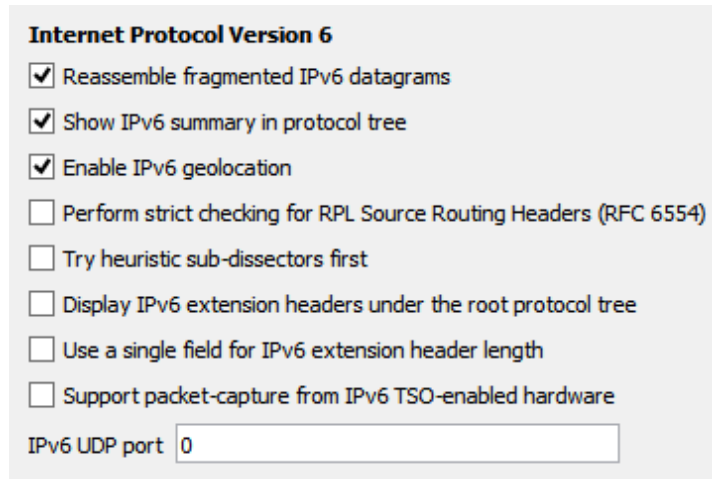


Figure 11.12 – IPv6 preferences

Once there, you can modify any of the options as described here:

- **Reassemble fragmented IPv6 datagrams:** When enabled, this will reassemble fragmented IPv6 datagrams.
- **Show IPv6 summary in protocol tree:** When enabled, this summarizes the header contents.
- **Enable IPv6 geolocation:** Wireshark uses the IP addresses to identify packet origin using the [GeoIP] databases. Select if you want to use this option.
- **Perform strict checking for RPL Source Routing Headers (RFC 6554):** If enabled, this will aid in troubleshooting streams using **Routing Protocol for Low-Power and Lossy Networks (RPL)**.
- **Try heuristic sub-dissectors first:** Wireshark will attempt to identify what type of application is used by using the port number to properly dissect the packet.
- **Display IPv6 extension headers under the root protocol tree:** IPv6 has several extension headers, such as the routing header and the fragment header. Enabling this option will display the headers under the root protocol tree.

- **Use a single field for IPv6 extension header length:** Enabling this will display a single field for the IPv6 extension header length (if any). If this is not enabled, the field will appear on two lines as follows:

```
Length: 0 (8 bytes)
[Length: 8 bytes]
```

- **Support packet-capture from IPv6 TSO-enabled hardware:** TSO is a performance-boosting technique used in a virtualized environment. When used, the packet length may be inaccurate. Enabling this option will attempt to correct any errors.
- **IPv6 UDP port: 0...:** Use this option if you want to change the protocol behavior.

The migration from IPv4 to IPv6 has been tepid, as many network administrators continue to use IPv4 on the LAN, mainly because of the flexibility of using private IP addresses. The following section outlines how the two protocols can coexist with one another on the same network, using various tunneling protocols.

## Discovering tunneling protocols

Some organizations have decided to make the switch to a dedicated IPv6 networked environment. However, many are running a dual-stack environment, where hosts that are using both IPv4 and IPv6 must be able to communicate with one another.

As evidenced, an IPv4 header is completely different to an IPv6 header. In order to have traffic pass from an IPv4 network through an IPv6 network and vice versa, the traffic must use a tunneling protocol. A discussion of the various ways to transport an IPv6 packet through an IPv4 network is outlined in RFC 7059, found at <https://tools.ietf.org/html/rfc7059>. The following diagram shows the proper format for encapsulation of an IPv6 packet within an IPv4 packet:

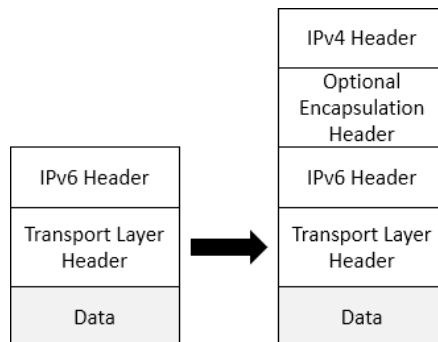
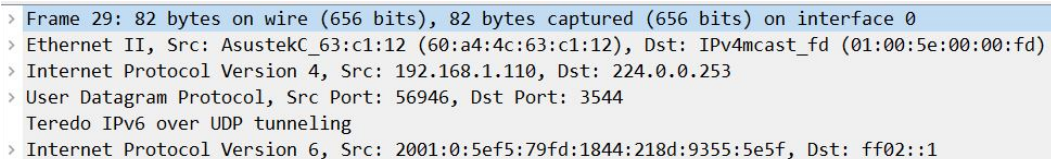


Figure 11.13 – Encapsulation of an IPv6 packet within an IPv4 packet

Some of the tunneling protocols that enable an IPv6 packet to travel over an IPv4 network include the following:

- **Intra-Site Automatic Tunnel Addressing Protocol (ISATAP):** Transmits data to and from hosts that use IPv6 through an IPv4 network
- **Generic Routing Encapsulation (GRE):** Creates a point-to-point IPv6 connection within an IPV4 network
- **Teredo:** Generated in a Windows OS to allow IPv4 hosts to connect to an IPv6 network when **network address translation (NAT)** is in place.

One of the tunneling protocols, Teredo, wraps (or encapsulates) an IPv6 packet with an IPv4 header so that the packets can travel over an IPv4 environment using NAT. To see an example of Teredo tunneling, go to <https://www.cloudshark.org/captures/c0b7d1a1d1ec?filter=frame%20and%20eth%20and%20ip%20and%20udp%20and%20teredo>, and open in Wireshark. Go to **Frame 29**, where we see the IPv6 packet encapsulated in an IPv4 header using UDP as the transport protocol, as shown here:



```
> Frame 29: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0
> Ethernet II, Src: AsustekC_63:c1:12 (60:a4:4c:63:c1:12), Dst: IPv4mcast_fd (01:00:5e:00:00:fd)
> Internet Protocol Version 4, Src: 192.168.1.110, Dst: 224.0.0.253
> User Datagram Protocol, Src Port: 56946, Dst Port: 3544
  Teredo IPv6 over UDP tunneling
> Internet Protocol Version 6, Src: 2001:0:5ef5:79fd:1844:218d:9355:5e5f, Dst: ff02::1
```

Figure 11.14 – IPv6 packet encapsulated in an IPv4 header

Although there are several tunneling protocols, they all do essentially the same thing: encapsulate one header by using another header, so that data can travel through the network. Because of this, there is additional overhead in creating the tunnel, as well as adding the additional headers.

Because of our complex network environment, you will most likely run into tunneling protocols at some point while troubleshooting your network.

## Summary

In this chapter, we started by covering a brief history of IP. We learned how both versions of IP can do the job of routing and addressing; however, there are several differences between the IPv4 and IPv6 headers. We examined and explained each of the field values of both IPv4 and IPv6. Additionally, to give you a better understanding of the two protocols, we compared some of the similarities along with the differences between IPv4 and IPv6.

To help strengthen your knowledge of addressing, we briefly covered the classes of IPv4 addresses, along with reviewing the different types of IPv6 addresses. We then looked at how you can personalize the settings for IPv4 and IPv6 by modifying the protocol preferences. Finally, because of the need for both IP versions to coexist on today's networks, we compared the different types of tunneling protocols in use today.

In the next chapter, we will learn about ICMP, the companion protocol to IP, which works in the network layer of the OSI model. We will evaluate both ICMP, which is used for IPv4, and ICMPv6, which is used with IPv6. We'll then take a deep dive into how ICMP works in both versions, and you will have a better understanding of the two types of messages: error reporting and queries. At the end of the chapter, you'll see how ICMP is the scout for IP and how its use is essential in delivering data.

## Questions

Now it's time to check your knowledge. Select the best response, and then check your answers, which can be found in the *Assessments* appendix:

1. Class Selector 6 in the `DiffServ` field is used with \_\_\_\_\_.
  - A. Signaling
  - B. Broadcast video
  - C. Network control
  - D. Realtime
2. The `172.18.23.119` IP address is a:
  - A. Class C IPv4 address
  - B. Class B private IPv4 address
  - C. Class E IPv4 address
  - D. Class D private IPv6 address
3. An IPv6 address has \_\_\_\_\_ bits.
  - A. 32
  - B. 48
  - C. 64
  - D. 128



4. In IPv4, we use a Time to Live value that indicates the number of hops it can take when traveling through the network. In IPv6, this field value is called the \_\_\_\_\_.
  - A. Router pass
  - B. Class stop
  - C. TTL
  - D. Hop count
5. An IPv4 header has \_\_\_\_\_ flag(s).
  - A. 1
  - B. 2
  - C. 3
  - D. 4
6. The following figure shows an IPv6 header. What is the value of the flow label?

```
Internet Protocol Version 6
  0110 .... = Version: 6
  > .... 0000 0000 .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
  .... 1111 1011 1011 0111 0100 = Flow Label: 0xfbb74
  Payload Length: 136
  Next Header: Routing Header for IPv6 (43)
  Hop Limit: 63
  Source Address: fc00:42:0:1::2
  Destination Address: fc00:2:0:5::1
  > Routing Header for IPv6 (Segment Routing)
```

Figure 11.15 – An IPv6 header

- A. 0X00
  - B. fc00
  - C. 0xfbb74
  - D. 136
7. In either an IPv4 or IPv6 header, the payload length cannot exceed \_\_\_\_\_ bytes.
  - A. 65,535
  - B. 1,111
  - C. 256
  - D. 128

## Further reading

Please refer to the following links for more information:

- In 2021, IANA updated the list for the next header field in an IP header. The list can be found at <https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>.
- Learn more about RFC 3168 by visiting <https://tools.ietf.org/html/rfc3168>.
- The TTL value varies as it is OS-dependent. To see the TTL values of various OSs, go to <https://subinsb.com/default-device-ttl-values/>.
- To view a list of IPv4 options, visit <https://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml#ip-parameters-1>, which was updated on May 3, 2018.
- A detailed explanation of the IPv6 header can be found at <https://www.microsoftpressstore.com/articles/article.aspx?p=2225063&seqNum=3>.
- IPv4 has several ranges of special and private IPv4 addresses. To see a complete list, visit [https://en.wikipedia.org/wiki/Reserved\\_IP\\_addresses](https://en.wikipedia.org/wiki/Reserved_IP_addresses).



# 12

# Discovering ICMP

Everyone is familiar with the **Internet Protocol (IP)**, which is responsible for routing and addressing traffic. However, many are not as familiar with the **Internet Control Message Protocol (ICMP)**, a powerful protocol that plays a major role in delivering data. In this chapter, we'll learn about ICMP, the companion protocol to IP, which works in the network layer of the OSI model. We'll begin with an overview so that you have a general understanding of the main functions of ICMP.

We will then evaluate both ICMP (used with IPv4) and **ICMPv6** (used with IPv6) so that you can compare some of the main differences. In addition, you'll get a better understanding of the two types of messages: **error reporting** and **queries**. We'll then look at common **type** and **code** values, along with a discussion on which types of ICMP messages are no longer used. We'll finish with a discussion on how malicious actors can use ICMP to launch an attack. Then, we'll outline some basic firewall guidelines in terms of what types of ICMP messages to allow on your network.

This chapter will address all of this by covering the following:

- Understanding the purpose of ICMP
- Dissecting ICMP and ICMPv6
- Sending ICMP messages
- Evaluating type and code values
- Configuring firewall rules

## Understanding the purpose of ICMP

Early in the 1980s, scientists developed protocols that drove internet traffic. In addition, they identified potential issues that might prevent traffic from reaching its destination, especially when using IP. This is mainly because IP doesn't guarantee delivery and has no way of communicating network problems with end devices. ICMP overcomes the deficiencies of IP by sending query messages and generating error reports on possible issues that may require attention.

As shown in the following table, the network layer is responsible for addressing and routing:

### OSI Model

Layer	Name	Role	Protocols	PDU	Address
7	Application	Initiate contact with the network	HTTP, FTP, SMTP	Data	
6	Presentation	Formats data, optional compression and encryption		Data	
5	Session	Initiates, maintains, and tears down a session		Data	
4	Transport	Transports data	TCP, UDP	Segment	Port
3	Network	Addressing, routing	IP, ICMP, ARP	Packet	IP
2	Data Link	Frame formation	Ethernet II	Frame	MAC
1	Physical	Data is transmitted on the media		Bits	

Figure 12.1 – The OSI model network layer

The network layer has three main protocols, which are essential in delivering data. In addition to ICMP, the other protocols are as follows:

- **IP** is a best-effort, connectionless protocol that routes packets from source to destination using a logical IP address.
- **Address Resolution Protocol (ARP)** resolves an IPv4 address (network layer) to a **Media Access Control (MAC)** (data link layer) address on a **local area network (LAN)** so the frame can be delivered to the appropriate host.

As we learned in *Chapter 11, Analyzing IPv4 and IPv6*, the network layer is responsible for addressing and routing traffic. Because IP is a best-effort, unreliable protocol, ICMP is essential for data delivery and must be implemented by every IP module.

ICMP has two main tasks:

- **Communicate errors** that prevent data delivery. Common issues include the network or port being unreachable.
- **Issue queries** such as an echo request/reply, which is used in the ping network utility, or router solicitation, which provides a way to solicit and receive router information.

As there are two IP versions, there are also two versions of ICMP. These have roles that are specific to their respective IP version:

- **IPv4** uses ICMP.
- **IPv6** uses ICMPv6.

Because of its role in assisting IP in delivering data, the use of ICMP is made clear in the **Request for Comments (RFC)** for both versions of ICMP as follows:

- As stated in RFC 792, *"ICMP is actually an integral part of IP, and must be implemented by every IP module."*
- As stated in RFC 4443, *"ICMPv6 is an integral part of IPv6, the base protocol must be fully implemented by every IP version six node."*

To communicate either queries or errors, an ICMP message must provide information within the header. In the next section, we'll see how an ICMP packet follows the IP header, along with the three fields that are present in both versions.

## Understanding the ICMP header

When communicating information, both ICMP and ICMPv6 packets will follow an IP packet, as shown in the following diagram:

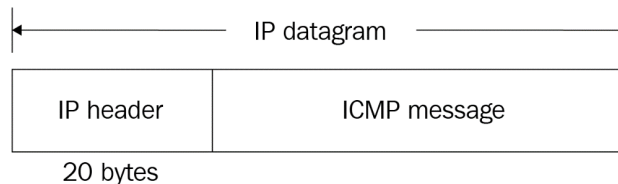


Figure 12.2 – ICMP message following an IP header

All ICMP messages have a common structure that begins with the **type**, **code**, and **checksum**, as shown along the top of the following diagram:

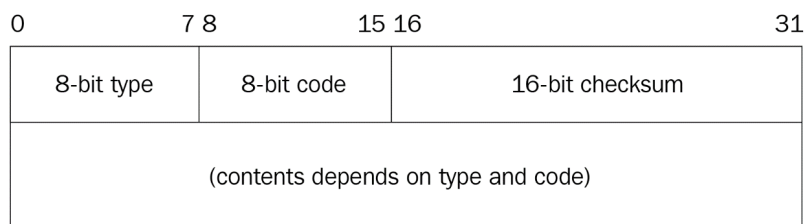


Figure 12.3 – ICMP message

The three fields consistent in an ICMP header are described as follows:

- **Type (8-bit):** This field indicates the type, such as Type: 3 (Destination unreachable).
- **Code (8-bit):** The code field *further defines* the type field. For example, Code: 1 (Host unreachable).
- **Checksum (16-bit):** This field holds a numeric value used for error detection.

Following the type, code, and checksum field values, you'll find the data payload. The contents will depend on the ICMP type and code, which can be either an ICMP error report or query message.

The entire payload is encapsulated in a frame, as shown in the following diagram:

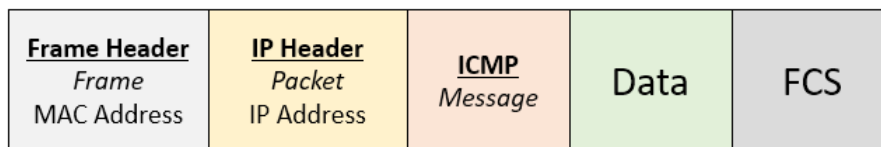


Figure 12.4 – ICMP message in an Ethernet II frame

Here we see the various headers, which include the frame header, IP header, ICMP message, data, and **Frame Check Sequence (FCS)**.

**Note**

ICMP *does not* have a transport layer header, as it does not exchange or transport data. Its primary role is to test for reachability and report transmission errors.

To examine several examples of ICMP in detail, we will use the `bigFlows.pcap` packet capture found at <http://tcpreplay.appneta.com/wiki/captures.html#bigflows-pcap>. Download the file and open it in Wireshark. Once open, apply the `icmp` filter. Select any of the ICMP packets and you will see the Type, Code, and Checksum fields. As shown here, we see the specifics of **Frame 202**:

```

> Frame 202: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
> Ethernet II, Src: 00:90:7f:3e:02:d0, Dst: 30:e4:db:b1:58:60
> Internet Protocol Version 4, Src: 172.16.128.254, Dst: 172.16.133.233
v Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0x6598 [correct]
  [Checksum Status: Good]
  Identifier (BE): 1894 (0x0766)
  Identifier (LE): 26119 (0x6607)
  Sequence Number (BE): 4 (0x0004)
  Sequence Number (LE): 1024 (0x0400)
  [Request frame: 38]
  [Response time: 98.640 ms]
> Data (36 bytes)

```

Figure 12.5 – Frame 202 - ICMP message

In the frame details, we see the following:

- Ethernet II: Frame header
- Internet Protocol Version 6: IP header
- Internet Control Message Protocol: Message
- Data: Specific to an ICMP echo reply

As shown in *Figure 12.5*, the details for this type of ICMP message include fields for identifiers and sequence numbers that help to match corresponding echoes and replies. This information is used by the host to reconstruct the process that sent the original datagram.

After the Type, Code, and Checksum fields, there is a *data portion* within the ICMP message. The following section explains what you might find in the data payload.



## Investigating the data payload

In an ICMP datagram, the payload is dependent on the type of message. In a standard ICMP request/reply, the data payload is meaningless and will have either ASCII characters or NULL values, depending on the **Operating System (OS)**.

In this section, we'll explore the different payloads you might encounter in an ICMP packet. In addition to data, an ICMP payload can include a watermark, or even evidence of concealed data, such as what you might see in a Loki attack.

Let's start with reviewing what you might see in an ICMP echo request.

### Viewing an echo request/reply

To see an example of an echo request, open `bigFlows.pcap`, and then go to **Frame 38**, which shows a Type 8, Code 0 message. Expand the ICMP header, as shown in the following screenshot:

```
> Frame 38: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
> Ethernet II, Src: 30:e4:db:b1:58:60, Dst: 00:90:7f:3e:02:d0
> Internet Protocol Version 4, Src: 172.16.133.233, Dst: 172.16.128.254
< Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x5d98 [correct]
  [Checksum Status: Good]
  Identifier (BE): 1894 (0x0766)
  Identifier (LE): 26119 (0x6607)
  Sequence Number (BE): 4 (0x0004)
  Sequence Number (LE): 1024 (0x0400)
  [Response frame: 202]
< Data (36 bytes)
  Data: 00000000138a1a34abcdabcdabcdabcdabcdabcdabcdabcdabcdabcd
  [Length: 36]
```

Figure 12.6 – ICMP echo request details

For example, in the echo reply shown in **Frame 38** the data portion is a string of characters as follows:

```
00000000138a1a34abcdabcdabcdabcdabcdabcdabcdabcdabcdabcd
cdabcdabcd
```

In addition, you should see a similar payload in a standard ICMP reply, which you can see in **Frame 74**.

While in an echo request/reply you will see a string of characters, an error message has a different format for the payload. Let's take a look.

## Reporting an error

Whenever ICMP encounters an error, ICMP must return the IP header, *plus at least the first 8 bytes (or 64 bits) of the original datagram*, to the sender.

### Note

The whole length of an ICMP error message cannot exceed 576 bytes.

To see an example, open `bigFlows.pcap` and use the `icmp.type == 3` filter, which will display all ICMP packets with a Type 3: Destination unreachable error message.

Select **Frame 4794** and expand the ICMP header where you will see the ICMP Type: 3 (Destination unreachable) Code: 3 (Port unreachable) fields, as shown here in the first section:

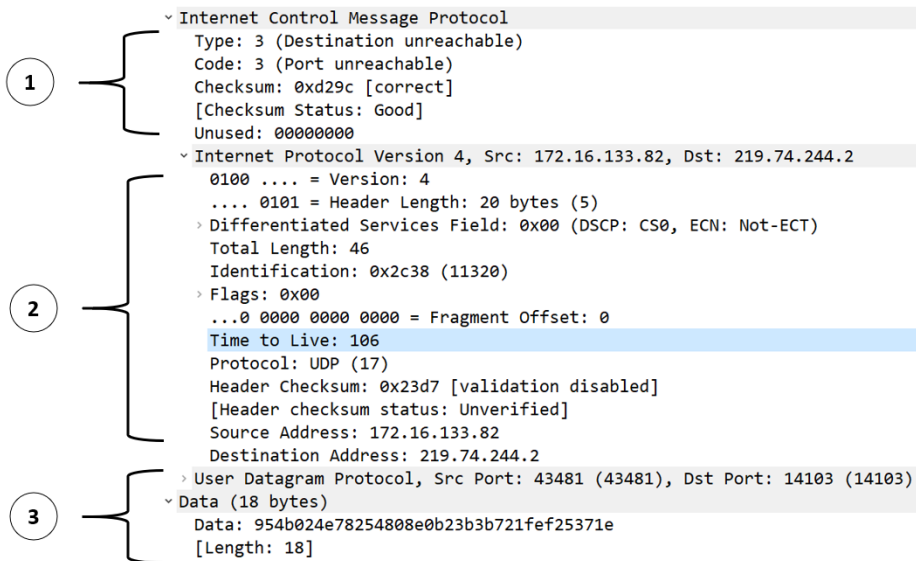


Figure 12.7 – ICMP echo request details

Because this is an error, ICMP has returned the IP header and the original datagram. As you can see, there are several parts to the ICMP packet, as follows:

- The first section (1) is the ICMP header indicating an error.
- The second section (2) is the original IP header.
- The third section (3) is the original **User Datagram Protocol (UDP)** header followed by the data.

In most cases, the only ICMP payload you might encounter is an echo request/reply along with error messages. However, there are other instances in which you might see a payload. Let's explore when this might happen.

## Including other data

ICMP was designed to issue queries and communicate errors on a network. However, over time, ICMP has been used for other purposes, as the data portion in an ICMP request can be modified. One way is by using a non-malicious watermark to identify a company while monitoring the network. Another way is the malicious intent to exfiltrate data by using an ICMP tunnel.

Let's start by seeing how a watermark is used.

## Creating a watermark

Paessler, a network monitoring company (<https://www.paessler.com/ping-monitoring>), uses ping monitoring to assess the health of the devices on the network. When gathering packets while in Wireshark, you'll see that Paessler's packets have a watermark, as shown in the following screenshot:

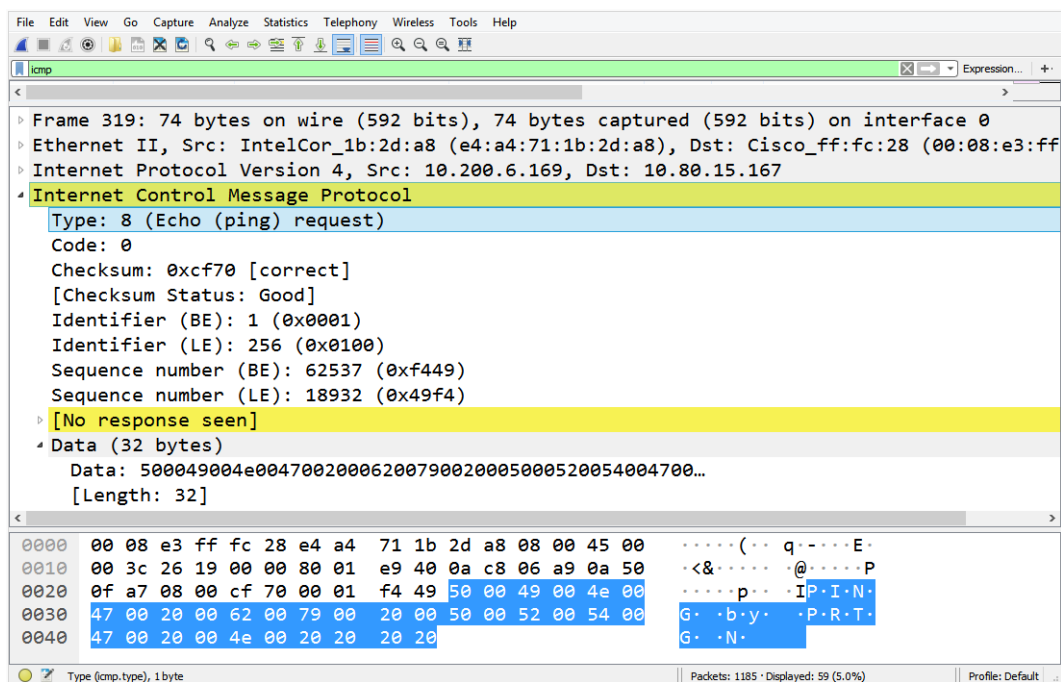


Figure 12.8 – Ping request with a watermark

In the **Packet Bytes** panel, you can see in the lower right-hand side an area that is highlighted. The data is as follows: P•I•N•G• •b•y• •P•R•T•G• •N•.

In this case, the watermark is not malicious. However, an ICMP packet can be modified to send data covertly, as we'll see next.

### Sending commands covertly

Malicious actors have found numerous ways to conceal their activity. One way to send commands through a network is by using a **Loki** tool to execute a covert channel attack, which works in the following manner:

1. Data is embedded within an ICMP echo packet.
2. The data is then covertly sent through the network.

Instead of the watermark as described in the previous section, this type of activity poses a security risk. As a result, the network administrator should tune devices to enable the inspection of ICMP data. If found, the device should send an alert if the payload contains an unrecognized data pattern, as this may be an indication of a covert ICMP tunnel.

We can now see that ICMP is an essential network layer protocol that is used alongside both IPv4 and IPv6 to provide error reporting and informational messages. Next, let's compare the two versions, ICMP and ICMPv6.

## Dissecting ICMP and ICMPv6

Although IPv4 and IPv6 are both responsible for routing and addressing data, the two protocols have a number of differences. As a result, there are two versions of ICMP.

In the next section, we'll explore ICMP alongside ICMPv6 so that you understand some of the basic roles and functions in reporting network issues.

Let's start with ICMP, which is used with IPv4.

### Reviewing ICMP

ICMP is used alongside IPv4 to communicate network issues that prevent data from being delivered. ICMP error and query messages can alert end systems when there are issues with connectivity and can also obtain diagnostic information from intermediary systems, such as the round-trip time.

As powerful as ICMP is, it cannot make IP a reliable protocol; it only assists in data delivery by providing error messages and information. There are times when the causes of delays in data transmission are outside of the messages that ICMP can send and report. In that case, it's up to the **Transmission Control Protocol (TCP)**, or other higher-layer protocol, to notify the host of transmission errors during delivery.

Next, let's take a look at ICMPv6, which has many of the same functions but also provides additional roles to support IPv6.

## Outlining ICMPv6

While IPv4 and IPv6 are similar in terms of their overall functions, IPv6 has many additional benefits, which include the following:

- Options and extensions
- Improved multicast routing
- **StateLess Address AutoConfiguration (SLAAC)**

As a result, ICMPv6 was developed for IPv6 and is used to communicate updates or error messages. An ICMPv6 message contains the **Type**, **Code**, and **Checksum** details, followed by the contents, which will depend on the type and the code. However, ICMPv6 has options related to IPv6 traffic.

One example is an ICMPv6 **router solicitation** message, which alerts other routers on the network of their presence. To see an example of ICMPv6 messages communicating to other devices on the network, go to CloudShark: <https://www.cloudshark.org/captures/0f90f2c2de86?filter=frame%20and%20eth%20and%20ipv6%20and%20icmpv6>.

Download and open `test55.pcap` in Wireshark. In the display filter, enter `icmpv6` and press *Enter* to run the filter. In the following screenshot, **Frame 63** has an ICMPv6 router solicitation message:

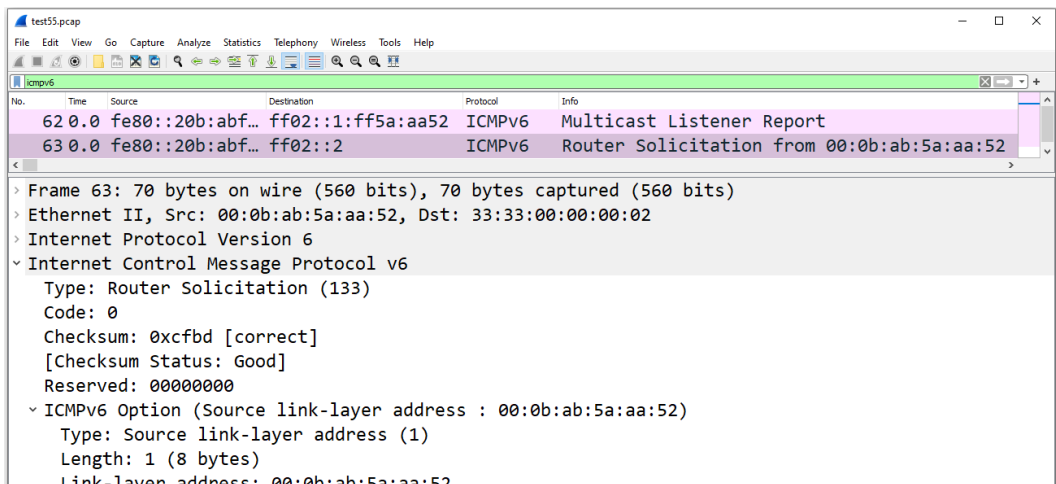


Figure 12.9 — ICMPv6 router solicitation

**Note**

The IPv6 `ff02::2` destination address is a multicast address, which are delivered to all nodes on a network using a single address.

Both ICMP and ICMPv6 can provide insight into network activity. The next section explores the two main functions of ICMP: reporting errors and queries.

## Sending ICMP messages

ICMP messages are grouped into two categories: error reporting and queries. Some messages are specific to each version; however, a few are common to both versions, as shown here:

ICMP Messages				
Error Reporting			Queries	
Type	Message		Type	Message
3	Destination unreachable		8/0	Echo Request/Reply
11	Time exceeded		9	Router Advertisement
5	Parameter problem			

Figure 12.10 – ICMP messages

For both categories, each ICMP packet has `type`, `code`, and `checksum` fields. The payload for queries is different from error messages, as each has a different purpose, as we'll see in the following sections.

Let's start with a review of how ICMP reports errors.

## Reporting errors on the network

ICMP error messages report on network issues that prevent data from being delivered. Commonly sent error messages are grouped into categories that have a specific purpose, and include the following:

- **Destination unreachable** is where a router informs the host that the requested destination address can't be reached.
- **Time exceeded** is sent when either the IPv4 **Time to Live (TTL)** value or IPv6 hop limit reaches zero.
- **Parameter problems** can be reported when there is an issue in determining a field value in the header or extension header.

Next, let's take a look at examples of some of the errors that you might see in your network.

### Viewing examples

We have already seen an example of a `Destination Unreachable` message in the *Investigating the data payload section*. Another error is `Time Exceeded`. To see an example of this error, open `bigFlows.pcap`, and then go to **Frame 7217**, which shows a `Type 11, Code 0` message. Expand the ICMP header, as shown in the following screenshot:

```

  ▾ Internet Control Message Protocol
    Type: 11 (Time-to-live exceeded)
    Code: 0 (Time to live exceeded in transit)
    Checksum: 0xf4df [correct]
    [Checksum Status: Good]
    Unused: 00
    Length: 32
    [Length of original datagram: 128]
    Unused: 0000
  ▾ Internet Protocol Version 4, Src: 172.16.133.109, Dst: 64.30.236.34
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x20 (DSCP: CS1, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x0000 (0)
    > Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    > Time to Live: 1
    Protocol: ICMP (1)
    Header Checksum: 0x1bcb [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.16.133.109
    Destination Address: 64.30.236.34

```

Figure 12.11 – ICMP Time Exceeded error message

In this case, once the packet hits the router, the TTL will have reached zero, and that will trigger the ICMP error.

Another example is a Parameter Problem error message. We can find this by going to CloudShark: <https://www.cloudshark.org/captures/bed61f75bde3>. Download and open alive6-1.pcap in Wireshark. Go to **Frame 3** and expand the ICMPv6 header, as shown:

```

  ▾ Internet Control Message Protocol v6
    Type: Parameter Problem (4)
    Code: 2 (unrecognized IPv6 option encountered)
    Checksum: 0x2def [correct]
    [Checksum Status: Good]
    Pointer: 42
  ▾ Internet Protocol Version 6
    0110 .... = Version: 6
    > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 0000 0000 0000 0000 0000 = Flow Label: 0x000000
    Payload Length: 24
    Next Header: Destination Options for IPv6 (60)
    Hop Limit: 255
    Source Address: 2001:470:cbf7:1ab:20c:29ff:feb7:8eeb
    Destination Address: ff02::1
    [Source SLAAC MAC: 00:0c:29:b7:8e:eb]

```

Figure 12.12 – ICMPv6 Parameter Problem error message



In this case, the error message is Type: Parameter Problem (4) Code: 2 (unrecognized IPv6 option encountered).

As discussed, ICMPv6 has many of the same functions as ICMP but also provides additional reporting to support IPv6. One of the ICMPv6-specific messages is `Packet Too Big`. Let's take a look.

## Monitoring the packet size

In addition to the three error messages reported in IPv4, ICMPv6 includes another error, `Packet Too Big`, as shown in this diagram listing some of the ICMPv6 error messages:

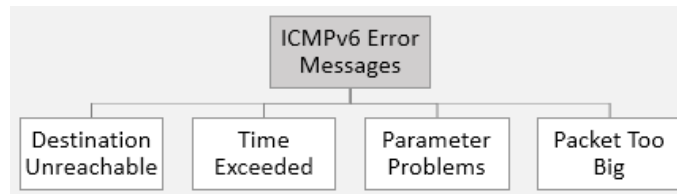


Figure 12.13 – ICMPv6 error messages

As we learned in *Chapter 11, Analyzing IPv4 and IPv6*, the payload length is a field value in IPv6. The packet size is monitored as the data travels through the network. If the packet is too large, ICMPv6 will report the `Packet Too Big` error. This error is sent when a device cannot send the data, as the packet is larger than the **Maximum Transmission Unit (MTU)** of the outgoing link.

To see an example of an ICMPv6 `Packet Too Big` message, go to CloudShark: <https://www.cloudshark.org/captures/7dd0b50eb768>. Download and open `packet too big.pcap` in Wireshark. Once open, go to **Frame 3** and expand the ICMPv6 header, as shown here:

```

v Internet Control Message Protocol v6
  Type: Packet Too Big (2)
  Code: 0
  Checksum: 0x2e57 [correct]
  [Checksum Status: Good]
  MTU: 1300
v Internet Protocol Version 6
  0110 .... = Version: 6
  > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
  .... 1010 1000 1001 1111 1000 = Flow Label: 0xa89f8
  Payload Length: 1456
  Next Header: Fragment Header for IPv6 (44)
  Hop Limit: 63
  Source Address: 2001:db8:1::1
  Destination Address: 2001:db8:2::2
  > Fragment Header for IPv6
  
```

Figure 12.14 – ICMPv6 Packet Too Big error

Because of the complexity of the network, sometimes it's hard to avoid this error. Let's see what options we have when the payload is too big for the MTU.

## Managing a Packet Too Big error

When sending data, most OSs employ **Path MTU Discovery (PMTUD)**, a process used to determine what size packet is allowed to travel on the network segment. If the packet is too big, the router will send an ICMPv6 `Packet Too Big` message back to the host. In that case, one of two things must be done:

- The packet must be discarded.
- The packet must be fragmented if allowed.

Fragmenting the packet will solve the problem of the packet being too big, as it can then be sent out onto the network. As we know, an IPv6 header *does not* have a fragmentation field and seeks not to fragment packets; however, it *can be* fragmented by using an extension header. You can see an example of the `Fragment Header for IPv6` fragment extension header in the last line of *Figure 12.14*.

### Note

Many devices on the network will block a fragmented packet, and for good reason. Fragmentation puts additional strain on a device and can pose a security risk.

ICMP error messages provide additional information so that the host can see exactly what happened. Errors are received and acted upon by TCP, IP, or user applications. However, in some cases, ICMP messages are ignored.

In addition to sending error messages, ICMP can also request and provide information, as discussed in the following section.

## Issuing query messages

An ICMP query has two messages (a request and a reply) that work together and have a specific purpose: to provide status updates and information.

Two examples of requests and replies are as follows:

- **Echo request/reply:** The ICMP packet works as a probe to test for the reachability of a remote host.
- **Router solicitation/advertisement:** Provides a way to solicit and receive router information that provides the IP addresses of the interface of an available router.

On a network that is primarily using IPv4, most ICMP messages provide enough information. However, to assist IPv6 in delivering data, ICMPv6 needs to provide information specific to IPv6, as we'll see next.

## Providing information using ICMPv6

Although in many ways, ICMP and ICMPv6 are similar, ICMPv6 has more responsibilities. The reason for this is that IPv6 no longer uses ARP broadcasts or **Internet Group Message Protocol (IGMP)**. As a result, ICMPv6 provides additional services to communicate issues on the network, as shown in the following diagram:

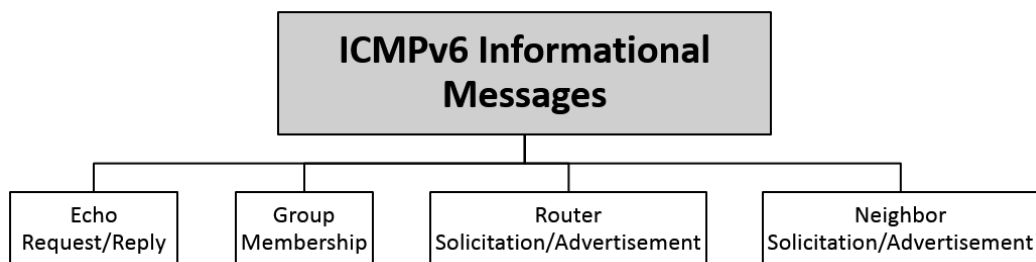


Figure 12.15 – ICMPv6 informational messages

Some of the messages unique to ICMPv6 are as follows:

- **Neighbor solicitation/advertisement:** These types are used for the **Neighbor Discovery Protocol (NDP)** to provide a method for hosts to share their existence on the network.
- **Multicast listener query/report:** This is used to exchange group multicast information to routers and hosts.
- **Group membership:** This is used to alert neighboring routers on hosts included in multicast group membership.

To see an example of the many ICMPv6 messages communicating to other devices on the network, open `test55.pcap` in Wireshark. In the display filter, enter `icmpv6` and press *Enter* to run the filter.

Create a flow graph by completing the following steps:

3. Go to **Statistics**, and then **Flow Graph**.
4. Once open, go to the lower left-hand corner and select **Limit to display filter**.

The results are as shown in the following screenshot:

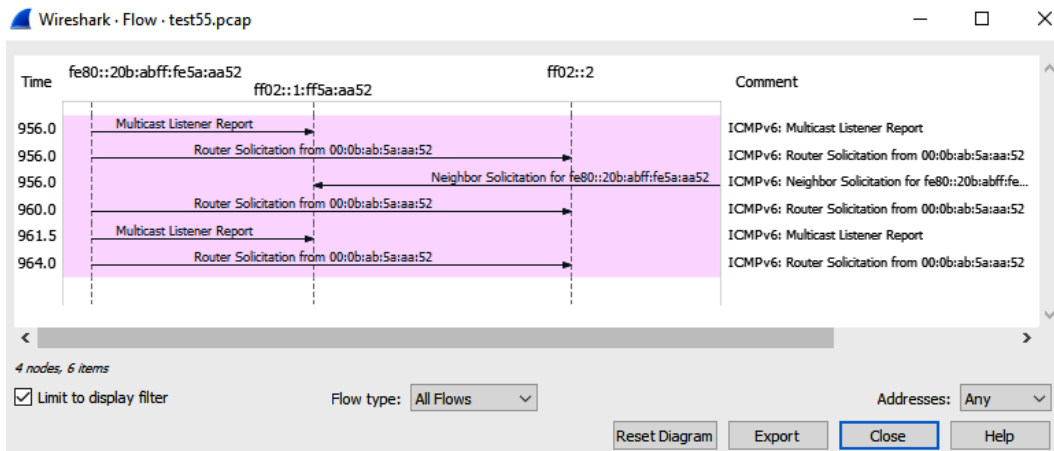


Figure 12.16 – ICMPv6 flow graph

Within the flow graph, you can see **Multicast Listener Report** and **Router Solicitation** ICMPv6 packets.

Some of the ICMPv6 reports have additional details. For example, in a separate capture file we can see the details provided in a single report as shown here:

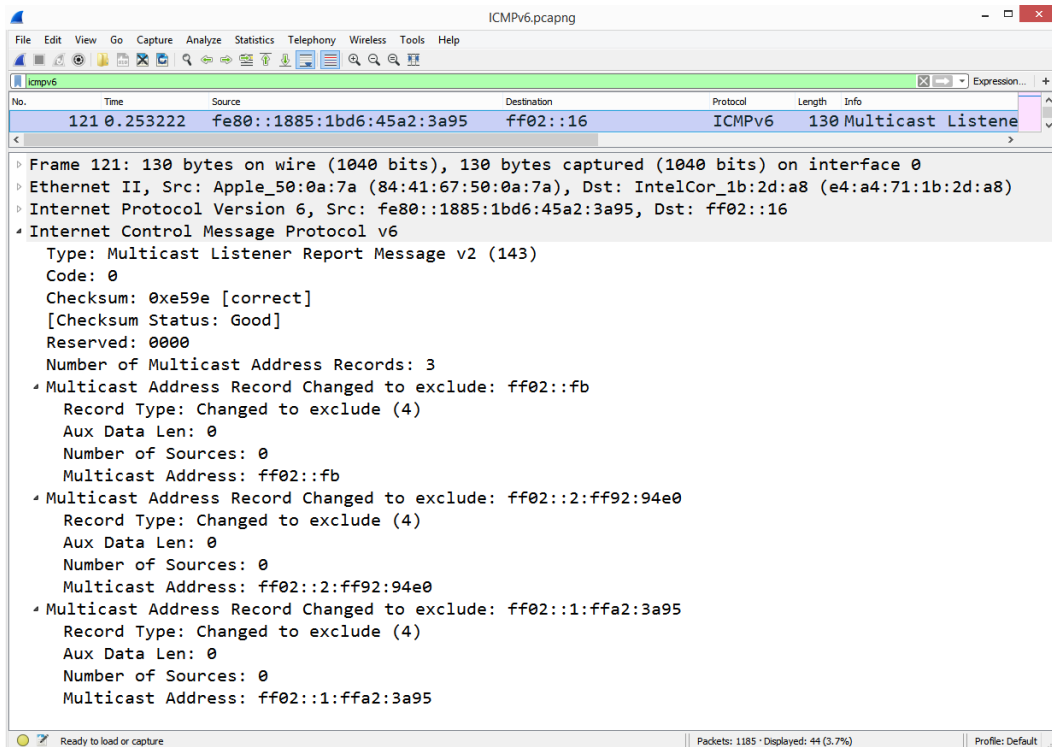


Figure 12.17 – ICMPv6 Multicast Listener Report

Another example of how ICMPv6 communicates on the network is the use of neighbor solicitation. Here is a host that has lost the connection to the gateway. To establish a connection, the host in **Frame 99** sends out multicast **Neighbor Solicitation** messages on the network, as shown in the following screenshot:

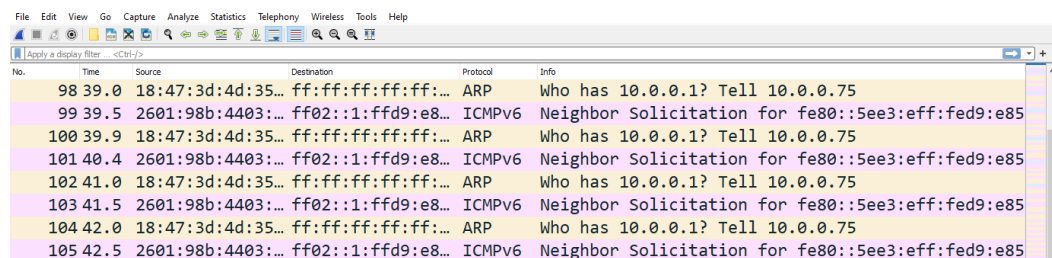


Figure 12.18 – ICMPv6 neighbor discovery

The reason you see multiple ICMPv6 packets is that the host is unable to reach the gateway. The **Neighbor Solicitation** messages will continue until the connection is restored.

As you can see, ICMPv6 is a powerful protocol. In addition to error and information messages, ICMPv6 provides additional information and works with IPv6 to maintain connectivity.

As discussed, ICMP headers hold a value for type and code. Let's take a look at these two fields in order to help us understand what ICMP is trying to tell us.

## Evaluating type and code values

The original objective of ICMP was to provide updates on network status and other informational messages. In this section, we'll review the type and code values for ICMP and ICMPv6. Let's start with ICMP.

### Reviewing ICMP type and code values

ICMP has been used for IPv4 for many years. There are many different types of ICMP messages, some of which should look familiar, such as these:

- **Type 0:** Echo reply
- **Type 3:** Destination unreachable
- **Type 5:** Redirect
- **Type 8:** Echo request
- **Type 9:** Router advertisement

Some, but not all, ICMP types have a corresponding set of code values that further define the ICMP message. For example, **Type 3** and **Type 9** both have a set of code values.

**Type 3** (destination unreachable) has many code values and includes the following:

- **Code 0:** Net (network) unreachable
- **Code 1:** Host unreachable
- **Code 2:** Protocol unreachable
- **Code 3:** Port unreachable
- **Code 4:** Fragmentation needed and the **Don't Fragment** bit was set

**Type 9** (router advertisement) only has two code values:

- **Code 0:** Normal router advertisement
- **Code 16:** Does not route common traffic

The type and code are the first two fields in an ICMP message and are used to convey information. For example, in **Frame 543**, ICMP has returned an ICMP Type: 3 and Code: 13, as shown in the following screenshot:

```

▶ Frame 543: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interf
▶ Ethernet II, Src: 88:75:56:3d:5e:00, Dst: e4:a4:71:1b:2d:a8
▶ Internet Protocol Version 4, Src: 10.19.28.1, Dst: 10.22.5.223
▶ Internet Control Message Protocol
  Type: 3 (Destination unreachable)
  Code: 13 (Communication administratively filtered)
  Checksum: 0x1cef [correct]
  [Checksum Status: Good]
  Unused: 00000000
▶ Internet Protocol Version 4, Src: 10.22.5.223, Dst: 10.80.15.169
▶ Transmission Control Protocol, Src Port: 64599 (64599), Dst Port: ms-wbt-ser
  Source Port: 64599 (64599)
  Destination Port: ms-wbt-server (3389)
  Sequence number: 3981044004

```

Figure 12.19 – ICMP Type 3 and Code 13

When ICMP returns Type: 3 Destination Unreachable Code: 13 (Communication Administratively Prohibited), this means a firewall is blocking the request.

As we have learned, ICMP headers hold a value for type and code to convey information on what is happening on the network. However, some of the ICMP types are no longer used and are deprecated. Let's explore this.

## Discouraging the use of outdated type values

While ICMP lists multiple type values, many are no longer in use because over time they have been found to be ineffective. The outdated and ineffective types are considered deprecated. The following are some of the deprecated ICMP types:

- **Type 4:** Source Quench
- **Type 33:** IPv6 Where-are-you
- **Type 34:** IPv6 I-am-here

- **Type 35:** Mobile Registration Request
- **Type 37:** Domain Name Request

Using deprecated ICMP type values is discouraged. To read more on best practices when dealing with ICMP type and code values, visit RFC 7279, found at <https://datatracker.ietf.org/doc/html/rfc7279/>.

Along with ICMP for IPv4, ICMPv6 is used to communicate updates or error messages and has its own set of type and code values. In the next section, we'll review some of the values for ICMPv6.

## Defining ICMPv6 type and code values

Because ICMPv6 provides additional data on IPv6 router and host configuration, you'll find specific type values that help provide this information.

A short list of ICMPv6 type values includes the following:

- **Type 1:** Destination unreachable
- **Type 2:** Packet too big
- **Type 3:** Time exceeded
- **Type 4:** Parameter problem
- **Type 130:** Multicast listener query
- **Type 131:** Multicast listener report

In some cases, the type will have a corresponding code value to further define the message, similar to IPv4. If the type does not have a corresponding code value, the field value will be set to 0, as shown in *Figure 12.17*.

The following are examples of the various type and corresponding code values for ICMPv6.

**Type 1** (destination unreachable) has several code values. Some of them are as follows:

- **Code 0:** No route to destination
- **Code 1:** Communication with destination administratively prohibited
- **Code 2:** Beyond the scope of the source address
- **Code 3:** Address unreachable



**Type 3** (time exceeded) has two codes, as follows:

- **Code 0:** Hop limit exceeded in transit
- **Code 1:** Fragment reassembly time exceeded

As we have learned, ICMP can provide a great deal of information on a network, therefore, it's important to understand that this protocol can be used in malicious ways. As a result, the firewall rules should be tuned to prevent malicious activity, as outlined in the next section.

## Configuring firewall rules

ICMP supports IP to help ensure data delivery; however, it can also be used in malicious ways. For example, ICMP can be used to conduct reconnaissance as a precursor to an attack, or even to help evade firewall rules. In this section, we'll provide an example of how ICMP can be used to obtain information on the network or to redirect traffic. Then, we'll evaluate some of the firewall rules used to limit the effectiveness of an attack.

Let's start with a brief discussion on a few attacks using ICMP.

### Acting maliciously

ICMP can determine a great deal of information about a network. As a result, it can be used as an effective scanning tool. In addition, if security devices aren't tuned to disallow certain types of ICMP packets, you may be the victim of an attack, such as a **Distributed Denial of Service (DDoS)** or redirect attack.

First, let's start with an overview of a **ping sweep**, a technique used to see which network hosts might be awake and responding.

### Sending malicious ping sweeps

Malicious actors use various techniques to scan a network for vulnerable hosts prior to an attack. One way is by using ICMP ping packets to determine which hosts are alive and responding. A ping sweep (or ping scan) uses a series of ICMP echo request packets on a local area network to see what hosts are alive and responding. As shown in the following diagram, the attacker sends a series of ping packets on the network. Once sent, the attacker then waits to hear a response from a host, in the form of an echo/reply, indicating it is alive:

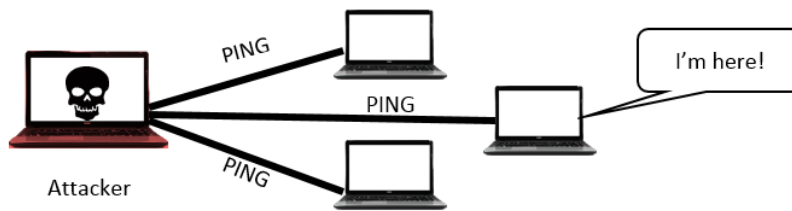


Figure 12.20 – ICMP ping sweep

Once a responding host is identified, the attacker will send more advanced probes to obtain additional information about the host.

Along with using a series of echo requests/replies, there are several ICMP queries that malicious actors can use to scout information before launching an attack. To see an example of using ICMP packets to probe a network, go to CloudShark: <https://www.cloudshark.org/captures/51eabf15169e>.

Download and open `sniffer_cybercop_scan_1-4223.cap` in Wireshark. In the display filter, enter `((ip.src == 192.168.10.33) && (icmp)) && !(ip.src == 192.168.10.138)` and press *Enter* to run the filter, which will then display 12 packets. Create a flow graph by completing the following steps:

1. Go to **Statistics**, and then **Flow Graph**.
2. Once open, go to the lower left-hand corner and select **Limit to display filter**.

The results are as shown in the following screenshot:

Time	192.168.10.33	192.168.10.138	Comment
4.1	Echo (ping) request id=0x2900, seq=9...		ICMP: Echo (ping) request id=0x2900, seq=9032/...
9.6	Echo (ping) request id=0x115c, seq=0/...		ICMP: Echo (ping) request id=0x115c, seq=0/0, ttl...
9.6	Echo (ping) request id=0x0e58, seq=0/...		ICMP: Echo (ping) request id=0x0e58, seq=0/0, tt...
9.6	Echo (ping) request id=0x0000, seq=0/...		ICMP: Echo (ping) request id=0x0000, seq=0/0, tt...
9.6	Echo (ping) request id=0x6418, seq=3...		ICMP: Echo (ping) request id=0x6418, seq=33435...
16.7	Echo (ping) request id=0x0100, seq=2...		ICMP: Echo (ping) request id=0x0100, seq=256/1,...
20.4	Echo (ping) request id=0x6c0c, seq=33...		ICMP: Echo (ping) request id=0x6c0c, seq=33435/...
21.3	Address mask request id=0x0100, seq=...		ICMP: Address mask request id=0x0100, seq=256...
21.3	Timestamp request id=0x0100, seq=...		ICMP: Timestamp request id=0x0100, seq=256/...
21.4	Address mask request id=0x0100, seq=...		ICMP: Address mask request id=0x0100, seq=256...
22.1	Address mask request id=0x0100, seq=...		ICMP: Address mask request id=0x0100, seq=256...
23.3	Address mask request id=0x0100, seq=...		ICMP: Address mask request id=0x0100, seq=256...

Figure 12.21 – Sending ICMP packets during reconnaissance

In this flow graph, we see how the malicious actor at 192.168.10.33 sends various ICMP messages to the 192.168.10.138 host. In addition to ICMP request packets, the malicious actor also uses timestamp and address mask requests. This type of probe sends various ICMP packets with the hope of getting a reply to help the scanning software rule out different OSs along with other information on the host.

Next, we'll take a look at another type of attack that uses an ICMP redirect message to redirect traffic.

## Redirecting traffic

While ICMP is designed to help data move on the network, malicious actors have found ways to use certain types of messages to launch an attack.

ICMP and ICMPv6 both have redirect messages:

- **ICMP** uses Type 5 - Redirect
- **ICMPv6** uses Type 137 - Redirect Message

When used to launch an attack, the malicious actor crafts specially designed redirect messages that can be used to modify the routing tables. The message contains information indicating there is a more optimal route to send traffic. Once the attack is set, the malicious actor poses as a router and sends the specially crafted ICMP message out on the network.

To see an example of using an ICMP redirect packet, go to <https://github.com/bro/bro/blob/master/testing/btest/Traces/icmp/icmp6-redirect.pcap>. In the middle of the page, select **View Raw**, and then open `icmp6-redirect.pcap` in Wireshark. There is only one packet. Expand both the IP and ICMP headers, as shown in this screenshot:

```

v Internet Protocol Version 6
  0110 .... = Version: 6
  > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
  .... 0000 0000 0000 0000 = Flow Label: 0x000000
  Payload Length: 40
  Next Header: ICMPv6 (58)
  Hop Limit: 255
  Source Address: fe80::dead
  Destination Address: fe80::beef
v Internet Control Message Protocol v6
  Type: Redirect (137)
  Code: 0
  Checksum: 0x593e [correct]
  [Checksum Status: Good]
  Reserved: 00000000
  Target Address: fe80::cafe
  Destination Address: fe80::babe

```

Figure 12.22 – An ICMPv6 redirect packet

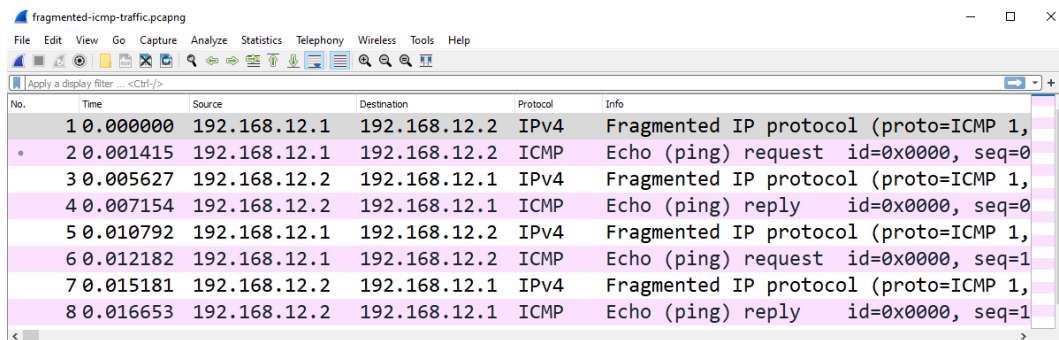
Once the host receives the redirect message, the traffic will be redirected to the malicious actor. The malicious actor can then intercept traffic to obtain sensitive information, redirect traffic to another server, or prepare for a more advanced attack.

The last attack we'll cover is an IP/ICMP fragmentation DDoS.

## Denying service

One of the ways to disrupt a business is by launching a DDoS. When launched on an internal network, one type of attack is a **volumetric DDoS**. A volumetric DDoS seeks to flood a device or network with so much traffic that it consumes all the resources, and legitimate traffic is locked out.

One example is a fragmented IP/ICMP volumetric DDoS attack. To look at an example of what you might see during this type of attack, go to <https://www.cloudshark.org/captures/962444a14a56>. Download and open `fragmented-icmp-traffic.pcapng` in Wireshark. Once open you will see the following:



No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.12.1	192.168.12.2	IPv4	Fragmented IP protocol (proto=ICMP 1,
2	0.001415	192.168.12.1	192.168.12.2	ICMP	Echo (ping) request id=0x0000, seq=0
3	0.005627	192.168.12.2	192.168.12.1	IPv4	Fragmented IP protocol (proto=ICMP 1,
4	0.007154	192.168.12.2	192.168.12.1	ICMP	Echo (ping) reply id=0x0000, seq=0
5	0.010792	192.168.12.1	192.168.12.2	IPv4	Fragmented IP protocol (proto=ICMP 1,
6	0.012182	192.168.12.1	192.168.12.2	ICMP	Echo (ping) request id=0x0000, seq=1
7	0.015181	192.168.12.2	192.168.12.1	IPv4	Fragmented IP protocol (proto=ICMP 1,
8	0.016653	192.168.12.2	192.168.12.1	ICMP	Echo (ping) reply id=0x0000, seq=1

Figure 12.23 – `fragmented-icmp-traffic.pcapng`

Within this capture, you will see the following:

1. The host at `192.168.12.1` sends Fragmented IP protocol in **Frame 1**.
2. This is followed by Echo (ping) request in **Frame 2**.
3. The host at `192.168.12.1` then repeats this pattern after every Echo (ping) reply from the host at `192.168.12.2`.

This means the host at `192.168.12.2` must continually reassemble the fragmented packets, which will require additional resources.

If you expand the IP header in **Frame 1**, you will see the following:

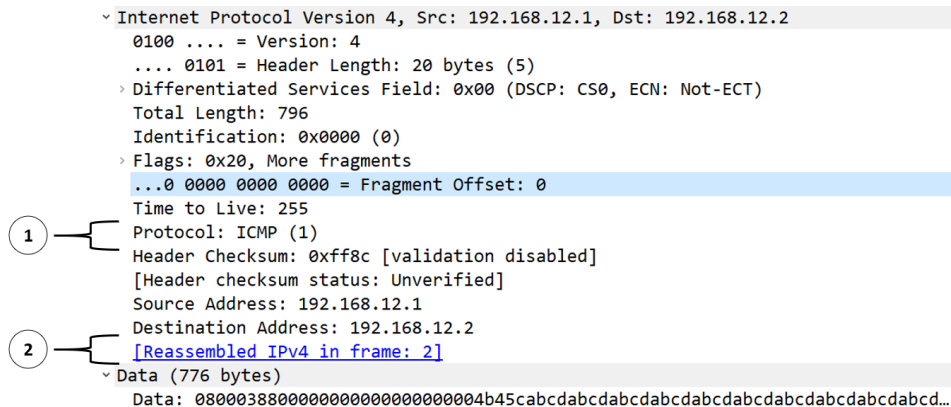


Figure 12.24 – A fragmented packet

As illustrated by (1), the protocol that follows the IP header is `Protocol: ICMP (1)`. We also see by (2) that the packet is `[Reassembled IPv4 in frame: 2]`.

The `fragmented-icmp-traffic.pcapng` capture only has 20 packets; however, if the attack were to continue, this could result in the device being unable to respond to legitimate hosts.

As evidenced, ICMP can be used to obtain information on hosts and act in a malicious manner. As a result, it's best to be aware of the various types and only allow ICMP packets that are absolutely necessary, as we'll see in the next section.

## Allowing only necessary types

Because ICMP can affect the operation of important system functions and obtain configuration information, hackers use ICMP messages while conducting reconnaissance on a network or during an active attack. As a result, best practices can include the following:

- Blocking certain ICMP messages with an **Access Control List (ACL)** on a firewall, especially at border routers
- Configuring an **Intrusion Detection System (IDS)** to either block or alert the administrator if an ICMP redirect message is detected

While ICMP can be used maliciously, we must remember that diagnostic utilities, such as Ping and Tracert, require ICMP. As a result, a network administrator must decide what types of ICMP packets should be allowed on a network. When setting up your firewall, keep in mind the *only essential* ICMP traffic is `Destination Unreachable (Type 3 for ICMP and Type 1 for ICMPv6)`, along with the corresponding codes.

All other ICMP types are optional, depending on whether you would like to allow them on your network. Depending on your organization, other types that are allowed can include the following:

- **Type 8/0:** Echo request/reply
- **Type 11:** Time exceeded

ICMP helps to ensure that data gets delivered; however, it can be used in malicious ways, therefore, you need to make sure that firewalls are properly tuned.

## Summary

By now, you can see the many aspects of ICMP, which is a significant protocol in the TCP/IP suite. We looked at the purpose of ICMP, a method to communicate issues that prevent data delivery. We compared ICMP and ICMPv6, which have similar functions, but we now understand that ICMPv6 has a bigger role. So that you can use this protocol while troubleshooting, we discovered ICMP messages that communicate with hosts to report on transmission errors. In addition, we saw how query messages can be used to attempt to obtain information from a host.

To better understand the ICMP type and code values, we examined how they work when communicating information. In addition, we learned that there are some ICMP types that you will rarely see as they are now deprecated and/or not supported. By now, you should recognize that ICMP is a powerful protocol that helps to move traffic on a network, but can also be used in malicious ways. As a result, you should configure firewall rules that allow or deny specific ICMP types in order to reduce the threat of malicious ICMP traffic on the LAN.

In the next chapter, we'll take a closer look at examining **Domain Name System (DNS)** traffic with Wireshark. Using practical examples, you'll be able to understand how DNS works when resolving a hostname to an IP address. You'll discover what the field values are in a DNS header, along with stepping through a DNS request/response transaction. In addition, because DNS problems are common on many networks, we'll review tools that can help you calculate DNS response time using Wireshark, along with methods to secure DNS.

## Questions

Now, it's time to check your knowledge. Select the best response, and then check your answers with those in the *Assessments* appendix:

1. ICMP communicates issues that prevent data delivery. Common issues include the network or port being \_\_\_\_\_.
  - A. Stateful
  - B. Inspected
  - C. Unreachable
  - D. ARP enabled

2. Some of the ICMP types are no longer used since, over time, they have been found to be ineffective and are considered \_\_\_\_\_.
  - A. Quenched
  - B. Unreachable
  - C. Digital
  - D. Deprecated
3. In ICMPv6, a \_\_\_\_\_ message can be sent when there is an issue determining a field value in the IPv6 header or the IPv6 extension header.
  - A. Time exceeded
  - B. Parameter problems
  - C. Packet too big
  - D. Source quench
4. When setting firewall rules, the only essential ICMP traffic is \_\_\_\_\_. The others are optional.
  - A. Type 3 for ICMP and Type 1 for ICMPv.
  - B. Type 123 for ICMP and Type 17 for ICMPv.
  - C. Type 1 for ICMP and Type 3 for ICMPv.
  - D. Type 9 for ICMP and Type 104 for ICMPv.
5. ICMP can be used in a malicious way. One way is a \_\_\_\_\_ scan that uses a series of ICMP echo request packets on a network to see what hosts are alive and responding.
  - A. Payload
  - B. Ping
  - C. Port
  - D. Checksum
6. In ICMPv6, a \_\_\_\_\_ message can be sent when a device cannot send the data as the packet is larger than the MTU of the outgoing link.
  - A. Time exceeded
  - B. Parameter problems
  - C. Packet too big
  - D. Source quench



7. Some, but not all, ICMP types have a corresponding set of code values that further define the ICMP message. For example, \_\_\_\_\_ both have a set of code values.
- A. Type 0 and Type 7
  - B. Type 1 and Type 2
  - C. Type 1 and Type 6
  - D. Type 3 and Type 9

## Further reading

Please refer to the following links for more information:

- To read more on RFC 792, *Internet Control Message Protocol*, visit <https://datatracker.ietf.org/doc/html/rfc792>.
- Learn more about ICMPv6 from RFC 4443, *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, at <https://datatracker.ietf.org/doc/html/rfc4443>.
- To view a discussion on various attacks, including a Loki ICMP attack, visit <https://resources.infosecinstitute.com/topic/icmp-attacks/>.
- For a discussion on three different methods to extract data using a covert channel, go to <https://www.giac.org/paper/gsec/2601/radar-covert-communications-channels/104464>.
- At some point, you may need to reference an ICMP type or code value. To see a summary of the most up-to-date values, visit the following:
  - For ICMP, visit <https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml>.
  - For ICMPv6, visit <https://www.iana.org/assignments/icmpv6-parameters/icmpv6-parameters.xhtml>.

# Part 4

# Deep Packet

# Analysis of Common

# Protocols

Once you understand the basic protocols of the TCP/IP suite, it's time to conduct a deep packet analysis of commonly used protocols – HTTP, DNS, DHCP, and ARP. We'll examine the header formats and field values of each protocol in detail, along with common options that can be used to facilitate the exchange of data.

The following chapters will be covered under this section:

- *Chapter 13, Diving into DNS*
- *Chapter 14, Examining DHCP*
- *Chapter 15, Decoding HTTP*
- *Chapter 16, Understanding ARP*



# 13

# Diving into DNS

The **Domain Name System (DNS)** converts a human-readable hostname into an **Internet Protocol (IP)** address. It is one of the most common application layer protocols in use today and is essential to any network. In this chapter, we'll review the purpose of DNS along with a brief history so that you can better understand how it all began. We'll then cover the different types of servers, such as root, authoritative, and recursive. Additionally, we'll compare how DNS records are transported when resolving an IP address versus updating the zone file.

To get a better understanding of this protocol, we'll review some of the common types and classes of **Resource Records (RRs)**. We'll also examine the DNS packet structure by drilling down into the header and query sections using Wireshark. You'll be able to recognize key field values such as the transaction **Identification (ID)** and header flags. In addition, you'll be able to break down how DNS presents the answers to a query. We'll then step through the process of resolving a hostname to an IP address. You'll learn how the **Operating System (OS)** stores a resolved address and how long it can remain in cache. We'll also cover how Wireshark can help you calculate DNS response times while troubleshooting, and finish with a discussion on ways to secure DNS.

This chapter will cover the following:

- Recognizing the purpose of DNS
- Comparing the different types of records
- Reviewing the DNS packet
- Evaluating queries and responses

## Recognizing the purpose of DNS

Anyone who has even had a basic networking class will be familiar with DNS. This well-known protocol provides an essential service of translating a human-readable hostname into a machine-readable numeric IP address.

In this section, we'll cover the concept of how DNS works, along with a brief history of how it all began. We'll then move on to a review of the different types of servers involved in resolving an IP address and finish up with comparing the ways DNS is transported.

Let's start with a discussion of why DNS is so essential.

## Mapping an IP address

DNS resolves addresses for hosts and services, along with any object that is connected to the internet. In addition, it is also required to deliver email, defend against spam, or initiate a **Voice over Internet Protocol (VoIP)** conversation.

Whenever a host makes a request to access a resource, the process moves through a series of servers to make the name resolution. The OS will search the local cache first, and if no answer is found, it will then extend the search to other servers, until a resolution is made.

But where did it all begin? In the next section, let's review the history of DNS.

## Reflecting on the start of DNS

For most of us, it seems that DNS has been around forever. But there was a time, when the internet was young, that all that was used to locate hosts was a plain text file. From the early days of the **Advanced Research Projects Agency Network (ARPANET)** to 1987, there were several significant advances in the development of DNS, as shown in the graphic:

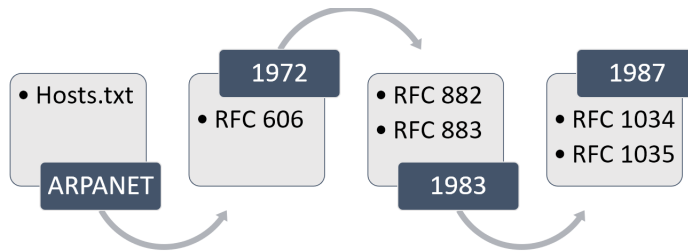


Figure 13.1 – Key moments in DNS history

The ARPANET began in 1969, and throughout the 1970s, the tiny network began to expand and add hosts daily. Similar to the way we identify hosts today, the ARPANET infrastructure used a numeric address. To locate another host, scientists devised a file called `hosts.txt`, which mapped hostnames to their network addresses.

The Host Status file, found at <https://datatracker.ietf.org/doc/html/rfc597>, included entries as follows:

101	65	UCLA-CCn	IBM 360/91	Server
204	132	UTAH-TIP		TIP

In the early days of ARPANET, each host kept a list of systems that were on the network. In addition to their network address, they assigned them human-readable names so that they could be more easily identified.

The file was then manually maintained and circulated to all members. Over the next two decades, significant changes occurred in the way the mappings were handled:

- **1973:** As ARPANET expanded, scientists recognized the limitations of a centrally managed file and began exploring alternatives to sharing the host information.
- **1983: Request for Comment (RFC) 882 and RFC 883** outlined the beginnings of DNS acting as a distributed (instead of centralized) system.
- **1984: Berkeley Internet Name Domain (BIND)** was released and provided a way to store and retrieve information about objects and resources. BIND remains the most widely used method to manage hosts in the world.
- **1987:** RFC 1034 and RFC 1035 outlined the framework for DNS as we know it today.

Although DNS works in a distributed manner, there is a structured hierarchy, using zones. A zone represents an area where an administrator can provide granular control of DNS objects.

Searching for an IP address can be complex; however, one starting point for all is DNS root zone. Let's explore this concept.

## Starting at the root

Root servers are name servers that respond to queries housed within the root zone. The servers are an essential component of resolving addresses so that data can move throughout the internet. The structure of a root server has the root server at the top, followed by the **Top-Level Domain (TLD)** servers, as shown in the graphic:

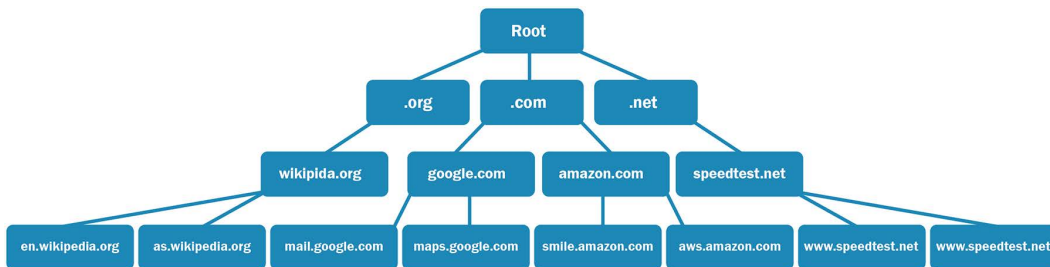


Figure 13.2 – The DNS root server hierarchy

The DNS root zone is comprised of 13 root servers along with approximately 600 redundant root servers around the globe. The TLD servers represent the extension of the domain name and provide a way to organize the data so that it's easier to resolve a query. Off of the TLD, you'll see the servers for the particular domain, followed by their respective subdomains, if any.

To give an example, if we start at the TLD.org, we see the following:

- The server, `wikipedia.org`, for the domain
- Followed by the respective subdomains, `en.wikipedia.org` and `as.wikipedia.org`.

Although the root server has a significant role, there are other servers involved when making a DNS request. Let's investigate this concept next.

## Types of DNS servers

When requesting an IP address, there are two main types of servers that are involved in a transaction, **authoritative** and **recursive**.

First, let's see how the authoritative server plays a significant role when resolving an IP address.

### Acting as an authority

Within any DNS structure, there must be an authoritative DNS server. The authoritative server holds a master set of records for the domain and is updated whenever there is a change.

Within each zone, there is an authoritative server. In most cases, there is more than one for redundancy. The different types are as follows:

- A primary server will house all reliable domain records, which includes the IP address and administrator name, along with domain-specific RRs.
- A secondary server will house a read-only copy of the zone file. Periodically, it will obtain an updated version of the file by requesting a zone transfer.
- A stub server contains only **name server (NS)** data, as its primary role is to send requests to a recursive server.

A primary authoritative server is required. A secondary authoritative server is optional. However, it is good practice to have a secondary server in place to provide redundancy and reduce dependency by sharing the load during a high-request volume.

Another type of DNS server is a recursive server.

### Using a recursive server

A recursive server is a non-authoritative server that holds cached copies of normal back-and-forth user queries.

When a client requests an IP address, the query can be either of the following:

- **Recursive** is when the DNS server will do the work of obtaining the response.
- **Iterative** is when the client must do the work of obtaining the response.



In most cases, when a client makes a DNS request, it will be directed to a recursive server. If the server does not have the requested IP address, it will continue to search for a response, on the client's behalf. The requests from the recursive server are done in an iterative fashion until they reach the authoritative DNS name server for the requested domain, as shown in the graphic:

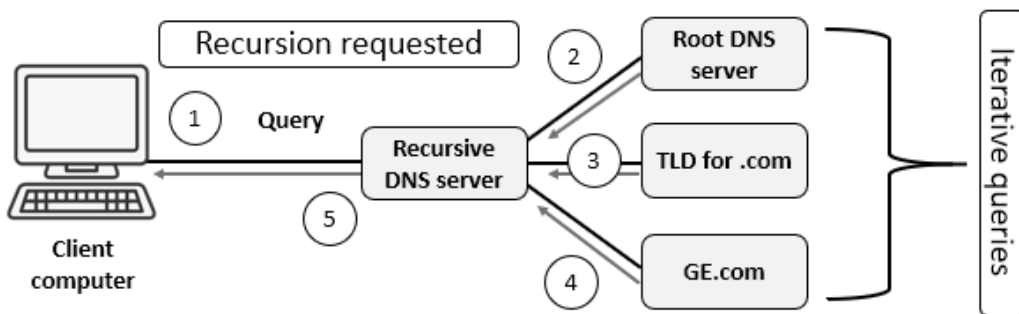


Figure 13.3 – A DNS query using a recursive server

With a recursive query, the resolution process for the IP address of `GE.com` is as follows:

1. The client requests an IP address for `GE.com`.
2. If the recursive server does not have the answer in the cache, it will need to query the root DNS server.
3. The root server will direct the recursive server to the TLD for `.com`.
4. The TLD server will direct the recursive server to query the authoritative nameservers for the `GE.com` domain.
5. The recursive server then returns the result to the client.

As evidenced, the DNS process involves many components to resolve an IP address. But how does DNS communicate with other hosts? As we know, many protocols use a dedicated transport method. For example, **Dynamic Host Configuration Protocol (DHCP)** uses **User Datagram Protocol (UDP)** as its transport protocol. However, in the next section, we'll learn how DNS can use both **Transmission Control Protocol (TCP)** and **UDP** using port 53, depending on the task.

## Transporting DNS

The two predominant transport layer protocols are TCP and UDP. The difference is as follows:

- UDP is a connectionless lightweight protocol used when data transport needs to be fast.
- TCP is a connection-oriented protocol that has the ability to ensure the complete delivery of data while monitoring for congestion and providing flow control.

When transporting data, DNS can use either TCP or UDP. First, let's investigate why UDP is the predominantly used protocol when making requests.

### Providing a speedy resolution

In most cases, DNS uses UDP for either primary or reverse DNS requests. One key reason is that when a host sends a query, the goal is to provide a speedy resolution. UDP doesn't go through a handshake process; it simply requests the resolution and waits for a response.

In addition, UDP was designed as a lightweight protocol to transport small bits of data using packets that are under 512 bytes. Most of the time, a DNS query or response is well under that limit.

Using UDP is an optimal choice for a transport protocol, as a DNS server must respond quickly and be available to respond to incoming queries, which wouldn't be possible if the server had to maintain a TCP connection.

Although UDP is the predominant transport protocol when using DNS, there are a few reasons why DNS will use TCP.

### Ensuring a complete transfer

When using DNS, there are two reasons TCP might be used as the transport protocol:

- When completing a zone transfer, as accuracy is more important than speed. Using a connection-oriented process will ensure the transfer is complete.
- Any time the payload is greater than 512 bytes, the application must use TCP.

All DNS records are not the same. In the next section, let's compare some of the DNS RRs.

## Comparing types and classes of RRs

At the heart of DNS are RRs, which include information on the name, type, **Time to Live (TTL)**, and IP address of the requested resource. In this section, we'll review some of the different types of RRs, along with examining the structure of one.

Let's start by reviewing some of the different types of RRs.

### Breaking down DNS types

Whenever you send a request to a DNS server, the request will include the type of record to return. Although there are many types of DNS RR, we'll take a look at some common types, as shown in the following table:

Value	Type	Description	Meaning
1	A	Address	A 32-bit IPv4 address
2	NS	Nameserver	Identifies the authoritative server for the zone
6	SOA	Start of authority	Marks the beginning of a zone
12	PTR	Pointer	Converts an IP address to a domain name
15	MX	Mail exchange	Redirects to a mail exchange (or server) that accepts email messages for a domain
28	AAAA	Address	A 128-bit IPv6 address
33	SRV	Service locator	Defines available services such as Lightweight Directory Access Protocol (LDAP) and Session Initiation Protocol (SIP).
252	AXFR	Zone transfer	Request for a zone transfer
255	*	Any	Request for all records

Table 13.1 – The types of DNS RRs

As shown, these are common records; however, there are more. For a comprehensive list of DNS types, visit <https://phoenixnap.com/kb/dns-record-types>.

**Note**

An AAAA-type RR, which is an IPv6 address, is also called a **Quad A**. This type uses four As because an IPv6 address is four times the length of an IPv4 address.

Now that you understand some of the RR types, let's take a look at the structure of an RR.

## Examining the RR structure

The structure of all RRs contains elements that describe the RR, as shown in the following table:

Identifier	Meaning
NAME	The name of the RR.
TYPE	The type of RR.
CLASS	The class of RR. In most cases, the class will be listed as IN (Internet).
TTL	The TTL value reflects how long the record can live in the cache.
RDLENGTH	Lists the length (in bytes) of the RDATA field.
RDATA	A string of bytes that describes the resource.

Table 13.2 – The elements in a DNS RR

Next, we'll take a look at an example of an RR answer in a DNS response packet. So that you can follow along, we will get a packet capture from CloudShark (<https://www.cloudshark.org/captures/13833cdd14ba>). Download the file, `DNS Question & Answer.pcapng`, and open it in Wireshark. Select **Frame 2** and expand the DNS header. Then, expand the first RR in the `Queries` section, which is shown as follows:

```

  v Answers
    v google.com: type A, class IN, addr 74.125.236.35
      Name: google.com
      Type: A (Host Address) (1)
      Class: IN (0x0001)
      Time to live: 4 (4 seconds)
      Data length: 4
      Address: 74.125.236.35

```

Figure 13.4 – A DNS RR answer

In any transaction, you'll find a great deal of information within the structure of a DNS packet. Let's investigate this next.

## Reviewing the DNS packet

DNS is a client-server model for resolving a hostname to an IP address. To see a DNS packet in its entirety, go to `DNS Question & Answer.pcapng` and expand the `Domain Name System (response)` caret in **Frame 2**, which is shown as follows:

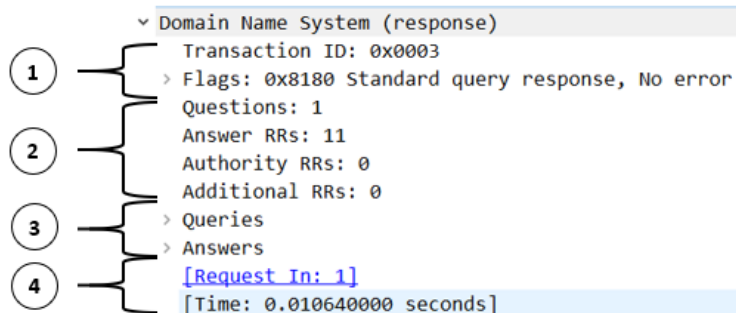


Figure 13.5 – Expanded DNS packet

As you can see, there are several parts to the DNS packet, as follows:

- The first section (1) is the DNS header.
- The second section (2) summarizes the contents.
- The third section (3) is the query section.
- The fourth section (4) are Wireshark-specific references.

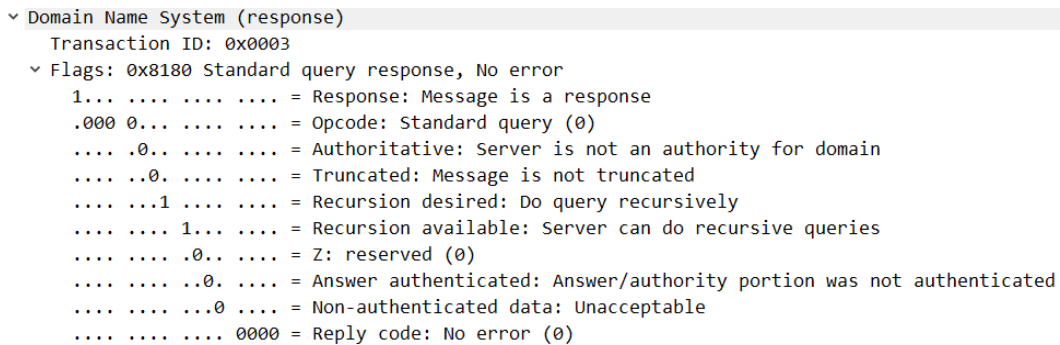
In this section, we'll review the field values in a DNS header, review the structure of a DNS packet, and then compare the DNS question and answer section.

Let's start with examining the DNS header elements.

## Examining the header

The header *structure* for either the client or the server is the same. What will be different are the *field values* for each header, which will identify whether the message is a query or response and include other parameters of the transaction.

Once in DNS Question & Answer.pcapng, expand the Flags caret in **Frame 2**, as shown in the following screenshot:



```

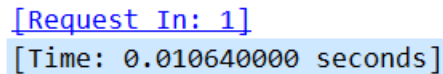
v Domain Name System (response)
  Transaction ID: 0x0003
  v Flags: 0x8180 Standard query response, No error
    1... .. = Response: Message is a response
    .000 0... .. = Opcode: Standard query (0)
    .... .0.. .. = Authoritative: Server is not an authority for domain
    .... ..0. .... = Truncated: Message is not truncated
    .... ...1 .... = Recursion desired: Do query recursively
    .... .... 1... .. = Recursion available: Server can do recursive queries
    .... .... .0.. .. = Z: reserved (0)
    .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated
    .... .... ...0 .... = Non-authenticated data: Unacceptable
    .... .... .... 0000 = Reply code: No error (0)
  
```

Figure 13.6 – The expanded DNS header

When examining the header fields, you will see two main sections:

- Transaction ID
- Flags

`Transaction ID` is an essential element of a DNS packet, as it identifies the specific DNS transaction during the exchange of the query and response packets. Wireshark keeps track of `Transaction ID` so that you can easily reference parts of the transaction. For example, at the bottom of **Frame 2**, you will see the following indicators:



The image shows two indicators from Wireshark. The first is a blue hyperlink text "[Request In: 1]". The second is a light blue highlighted text "[Time: 0.010640000 seconds]".

Figure 13.7 – Wireshark-created indicators

`Request In: 1` is a hyperlink, which will take you to the request packet for `Transaction ID: 0x0003`.

**Note**

The `Time` section lists the time that transpired between the query and the resultant response.

After `Transaction ID`, you will see the `Flags` section, which provides the details of the transaction. You will notice, however, that the flags for the client are different than the flags for the server. Let's explore this next.

## Comparing the flags

When comparing the flags in a DNS header, you will see a distinct difference between the packet coming from the client and the packet coming from the server.

The flags from the *client* are as follows:

- **Response:** Set at 0, which indicates that the message is a query.
- **OpCode:** For most requests, the **Operation code (OpCode)** is 0, which indicates this is a standard query.
- **Truncated:** When set at 0, this means the message is not truncated.
- **Recursion desired:** When set at 1, this will indicate that the client requests recursion.
- **Z:** A deprecated flag that was used in outdated DNS implementations.
- **Non-authenticated data:** Set at 0, as this flag is used only in a response packet.

The flags from the *server* are as follows:

- **Response:** Set at 1, which indicates that the message is a response.
- **OpCode:** For most responses, the OpCode is 0, which indicates this is a standard query.
- **Authoritative:** Indicates whether the server is an authority for the domain. When set at 0, this indicates the server is not an authority for the domain.
- **Truncated:** When set at 0, this means the message is not truncated.
- **Recursion desired:** When set at 1, this will indicate that the client requests recursion.
- **Recursion available:** When set at 1, this will indicate that the server can do recursive queries.
- **Z:** A deprecated flag that was used in outdated DNS implementations.
- **Answer authenticated:** Will indicate whether the answer/authority portion was authenticated by the server.
- **Non-authenticated data:** Used in a response packet. If the value is set to 1, this will indicate that all data included has been authenticated by the DNS server.
- **Reply code:** 0 will indicate that there is no error.

**Note**

When set, the truncated flag will indicate whether the message is shortened because it exceeds the limit of 512 bytes for a UDP packet.

The OpCode flag will indicate the type of message that is being sent, as it issues a command to the DNS server to perform some action. Let's take a look.



## Defining the opcode

In most cases when examining the OpCode value, the value will be set at 0, indicating that this is a standard query. However, there may be other values set, as follows:

OpCode	Description
0	Standard query
1	Inverse query (obsolete)
2	Request for server status
3	Unassigned
4	Notify
5	Update
6	DNS Stateful Operations (DSOs)
7–15	Unassigned

Table 13.3 – DNS OpCode

Some OpCodes, such as 3 and 7–15, are currently unassigned. However, that can change – for example, OpCode 6, DSO, is a newer opcode, used when there is an active state, such as during a TLS session.

The one flag that is most likely set at 1 in a query is the recursion flag. Let's discuss the significance of this flag.

## Requesting recursion

There are two types of queries in DNS – **recursive** and **iterative**. Whenever the client submits a DNS query, it's common to see the following flag set – `Recursion desired: Do query recursively`.

A recursive query is the preferred option, as when available, the DNS server will do the work of obtaining the response. In contrast, an iterative query is for when the client must do the work of obtaining the response.

When the server responds, it will reply with either of the following:

- `Recursion Available` where the flag is set at 1, which means the server will complete a recursive query
- `Recursion Not Available` where the flag is set at 0, which means the server will not complete a recursive query, as this type of query is not supported

When looking at a DNS header for either a request or reply, there are four sections that follow the message header. Let's explore these next.

## Dissecting the packet structure

The four sections that follow the DNS header, for either a query or response, are as follows:

- `Questions`: The requested resolution
- `Answers RRs`: The number of RRs that provide the answer(s)
- `Authority RRs`: The number of RRs that provide the IP address of an authoritative NS
- `Additional RRs`: The number of RRs that contain additional information

To give an example, as shown in the following graphic in the content summary (as shown in the section 2), we can see that there was **1** question and that it returned **11** RRs:

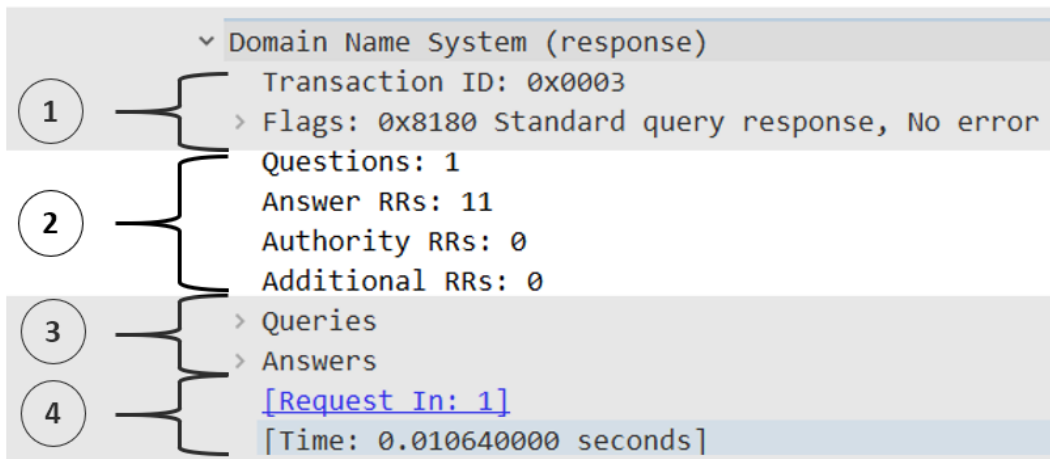


Figure 13.8 – The DNS summary section

Following the packet structure, you'll find the query section, which is used to send the question or provide the response.

## Outlining the query section

When looking at DNS header details, the format for the DNS question is different than the DNS answer, as shown here:

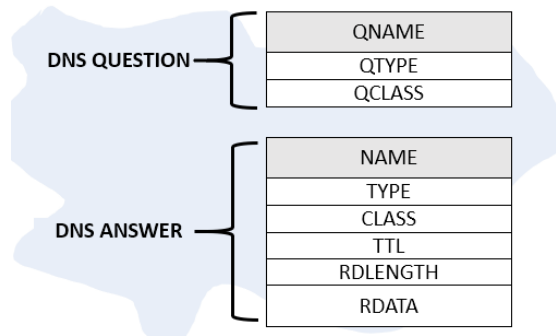


Figure 13.9 – DNS questions and answers

When a client makes a request to a DNS server, it is asking for a resolution. If we go to `DNS Question & Answer.pcapng` and expand the `Queries` caret in the request in Frame 1, from the client, we will see the following information:

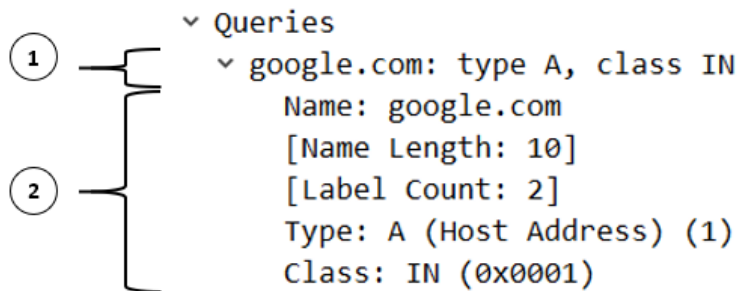


Figure 13.10 – The DNS client query

The first section (1) is Wireshark's summary of the query section. The second section (2) is the DNS question section, which contains the following:

- **Name:** The requested resolution. In this case, the client would like a resolution of `Google.com`.
- **Type:** The type of RR. In this case, the client is requesting an A type or a Type A (IPv4) address.
- **Class:** The class of RR requested. In this case, the class is listed as `IN`, which is the most common QClass request.

When a server replies to a DNS request, it will return the client's query and (in most cases) return answers. If we go to `DNS Question & Answer.pcapng` and select the `Answers` caret in the response in Frame 2, from the server, we will see the following information:

```
▼ Answers
  > google.com: type A, class IN, addr 74.125.236.35
  > google.com: type A, class IN, addr 74.125.236.37
  > google.com: type A, class IN, addr 74.125.236.39
  > google.com: type A, class IN, addr 74.125.236.32
  > google.com: type A, class IN, addr 74.125.236.40
  > google.com: type A, class IN, addr 74.125.236.33
  > google.com: type A, class IN, addr 74.125.236.41
  > google.com: type A, class IN, addr 74.125.236.34
  > google.com: type A, class IN, addr 74.125.236.36
  > google.com: type A, class IN, addr 74.125.236.46
  > google.com: type A, class IN, addr 74.125.236.38
```

Figure 13.11 – The DNS server answers

Of course, answers from the server can vary. However, in this case, the server has returned 11 RRs.

Now that you understand the elements of a DNS packet, let's take a look at some other considerations when DNS makes a resolution.

## Evaluating queries and responses

DNS queries and responses are pretty straightforward. A client sends a query to a DNS server for an IP address and the server responds with the information. In this section, we'll take a look at some of the behavior during a transaction, such as caching responses, along with monitoring average response times during a transaction using Wireshark.

We'll then evaluate what happens when we need to troubleshoot DNS, and how `nslookup` helps to check and verify the response. Finally, we'll take a look at spoofing DNS, and how we can secure the process.

Let's start with learning how caching plays a part in the DNS process.

## Caching a response

Anything on the network has a time limit. DNS is no exception. When a server returns a response, there are several elements within the answer. Within that response is the TTL value, which reflects how long the record can live in the cache before disappearing.

The TTL value can vary by system, however, often the value is set by the domain name owner.

In *Figure 13.4*, we can see `Time to live: 4 (4 seconds)`, which means the response can live in the cache for 4 seconds.

### Note

Keep in mind that a TTL in a DNS RR is different than the TTL value in an IPv4 header. The TTL in a DNS RR is how long the response can live in the cache. A TTL in an IP header represents how many hops a packet can take before being dropped by a router.

It's common to see a wide range of TTL values for DNS RRs. To check your own cache on a Windows machine, open a Command Prompt and type `ipconfig /displaydns`. This will display a list of your own DNS RRs held in the cache – for example, after running the command, I selected one of the records, as shown here:

```
api.wildtangent.com
-----
Record Name . . . . . : api.wildtangent.com
Record Type . . . . . : 1
Time To Live . . . . . : 17
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . : 34.210.94.123
```

Within the response, we can see that the RR answer for `api.wildtangent.com` has a `Time To Live` value of 17 (seconds).

If, after 17 seconds, the OS needs the IP address for `api.wildtangent.com` and the DNS information is not in the cache, it must issue another DNS query.

Most DNS queries and responses are under 120 milliseconds (ms or msec) or 0.12 seconds; however, some responses can take longer. Next, let's see how we can use Wireshark to calculate response times.

## Calculating response times

DNS response times can vary; however, there are times when a response seems longer than usual. To test response times, the network administrator has a variety of tools. For example, to test the speed of your DNS provider, go to <https://www.dnsperf.com/dns-speed-benchmark>. Once there, enter `example.com` in the domain name field and then select **RUN TEST**. The site will display response times for servers around the world. Scroll down, where you will see results such as the following:

- 36 ms [United States] United States, Dallas (DC139)
- 14 ms [United States] United States, Asheville (DC82)
- 126 ms [Egypt] Egypt, Cairo (DC225)

You can also view response times using Wireshark-generated statistics. Let's take a look.

## Viewing DNS response times

To show you an example of visualizing longer DNS response times, we will use `bigFlows.pcap`. You can find Bigflows at <http://tcpreplay.appneta.com/wiki/captures.html#bigflows-pcap>.

Once you open `bigFlows.pcap`, apply the following filter:

```
(((((dns) && (dns.flags.response == 1)) && !(sflow)) &&
!(icmp)) && !(dns.flags.rcode == 2)) && (dns.time > 0.2)
```

The filter will do the following:

- Display all DNS traffic using the `dns` filter
- Show *only* DNS responses using the `dns.flags.response == 1` filter
- Omit any DNS server errors using: `!(dns.flags.rcode == 2)`
- Remove all management traffic such as **Internet Control Message Protocol (ICMP)** and **Sampled Flow (sFlow)** traffic using the `!(sflow) && !(icmp)` filter
- Narrow the search to show only packets with a response time greater than 0.2 seconds by using the `dns.time > 0.2` filter

Once you run the filter, you can add a column header to show response times.

To add a column header, first expand the DNS header to view the details. Next, locate the Wireshark-generated value, `Time`, as shown here:

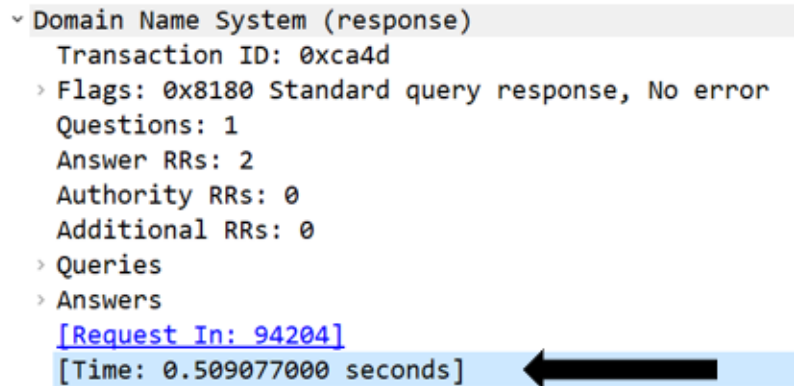


Figure 13.10 – The DNS server answers

Right-click on the value and select **Apply as Column**. To adjust and edit the columns so that you can better visualize the data, right-click on any column header and select **Column Preferences...**

Once there, you can move the columns by clicking and dragging the column title to the desired location. To display the response times, I have renamed the shortcut DNS Response time and disabled some of the column headers, as shown here:

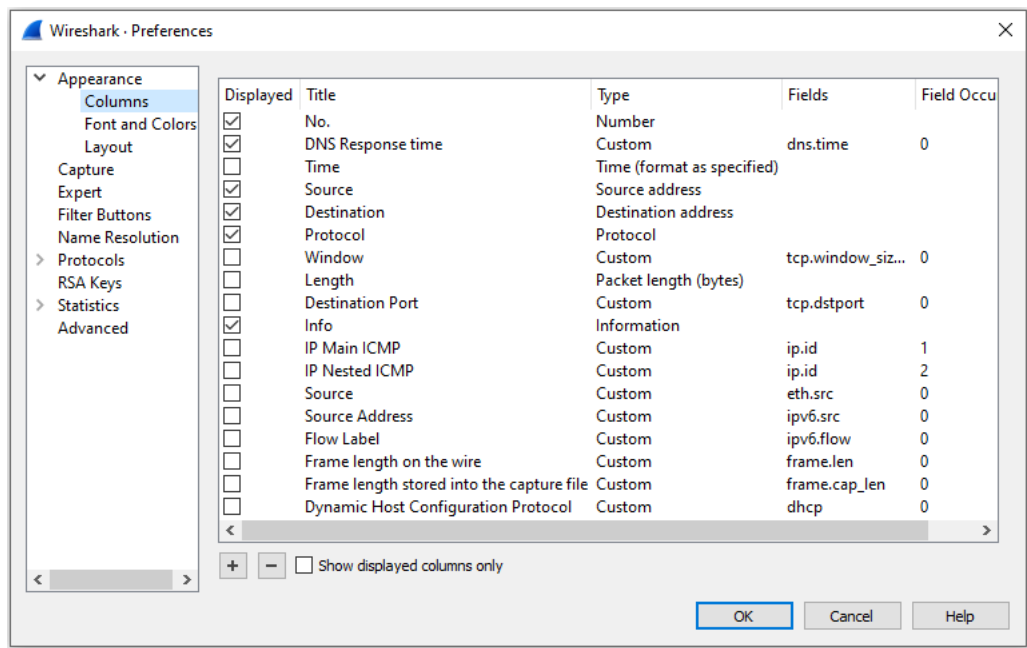


Figure 13.11 – Column preferences

After you have adjusted the columns' headers, you can sort the DNS response time to show the longest time value by clicking once on the **DNS Response time** header. The results are shown here:

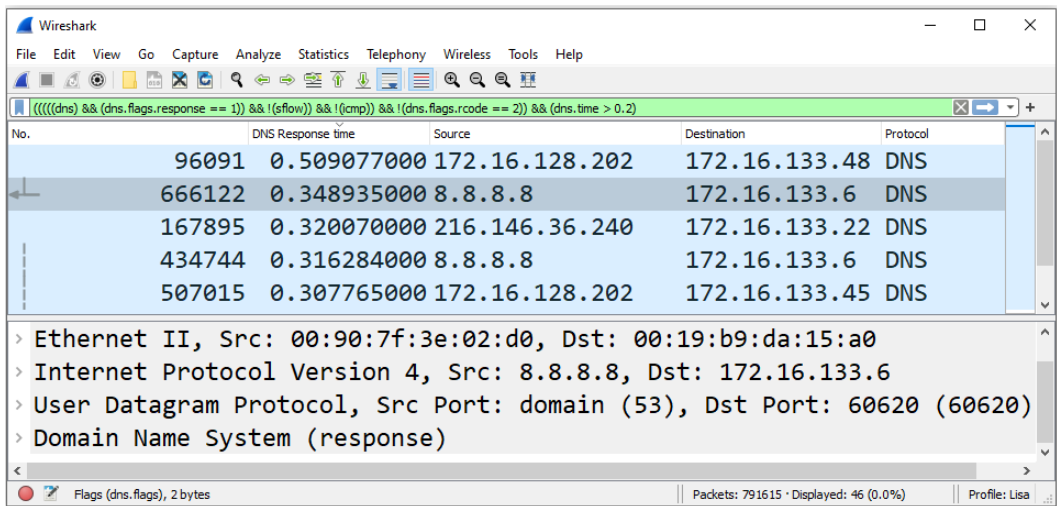


Figure 13.12 – BigFlows showing DNS response times



As shown, DNS response times can vary. While this is expected, another way to view the health of a DNS service is to check the statistics. To achieve this, go to **Statistics | DNS**, which will open the DNS statistics window. I applied the `!(dns.flags.rcode == 2)` filter, which then removed the server failure errors, as shown here:

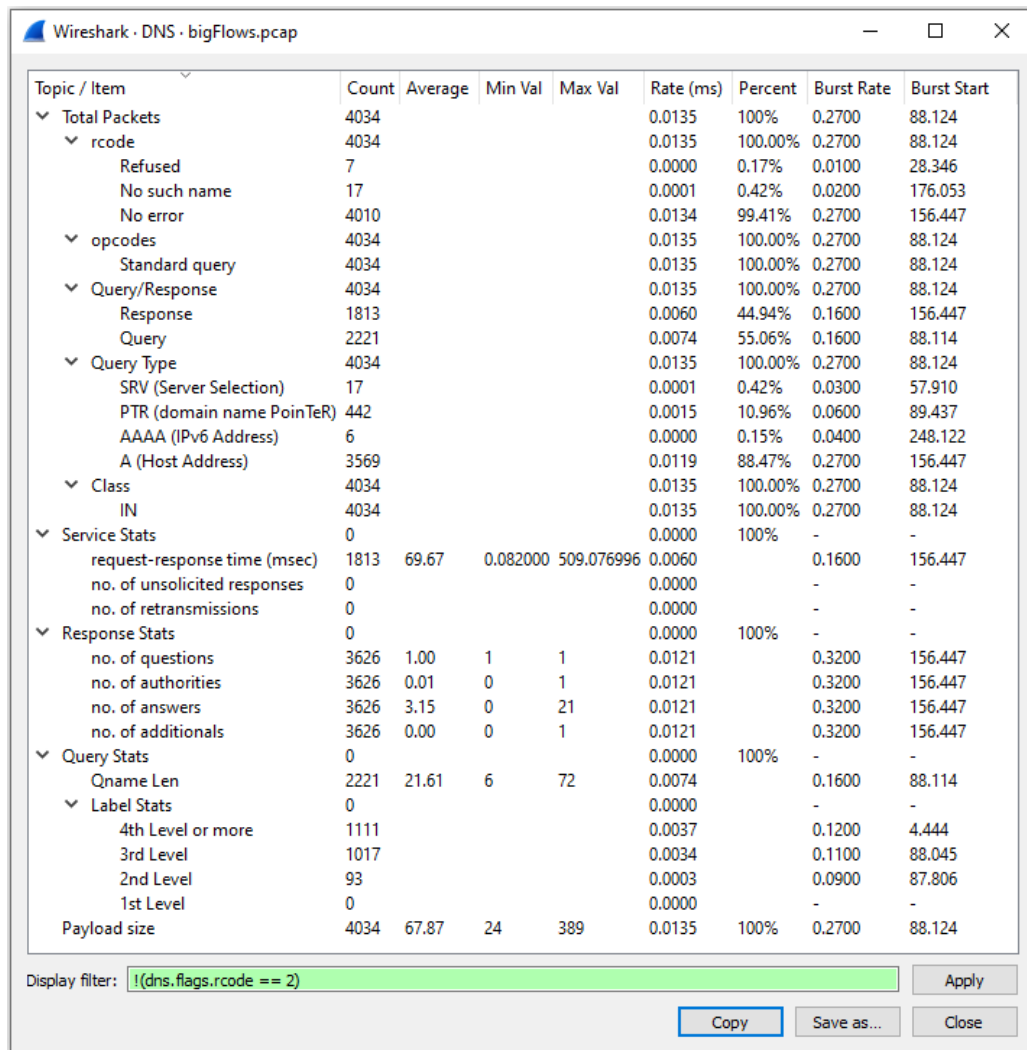


Figure 13.13 – BigFlows DNS statistics

Within the statistics window, you can see a variety of calculated values. In the center of the window are the service stats. We can see for this (filtered) capture that the request-response time in millisecond stats are as follows:

- Average request-response time – **69.67** milliseconds
- Minimum request-response time – **0.082000** milliseconds
- Maximum request-response time – **509.076996** milliseconds

**Note**

In most cases, you will want to see the server failure errors. However, in this capture, there were so many errors it skewed the response time stats.

DNS statistics can be a valuable tool to use during troubleshooting. For example, it will be worth investigating the following issues:

- Queries with a lengthy request-response time or failed responses might indicate that the DNS server is having issues, such as a bottleneck or a failing **Network Interface Card (NIC)**.
- If you see an unusually large number of requests and responses, this may indicate DNS tunneling.
- Numerous requests and responses can also be indicative of a host communicating with a **Command and Control (C&C)** server, which is used to remotely control an infected host.
- Experiencing a disproportionately large amount of DNS responses can indicate a **DNS Denial of Service (DoS)** may be at play.

Next, let's take a look at `nslookup`, a command-line tool you can use while troubleshooting DNS.

## Testing using nslookup

Setting up and maintaining a DNS server can be challenging. The DNS service is essential, and if improperly configured, this can cause serious issues in the ability of hosts to communicate with one another. In addition, if the DNS cache has been poisoned, this also will prevent hosts from communicating and can misdirect clients to malicious sites.

One way to do a quick test on a Windows machine is to use `nslookup`, a command-line tool primarily used to troubleshoot DNS issues.

To use `nslookup`, go to *Start* and then type *Run*. Once in the app, type `cmd`, as shown in the following screenshot:

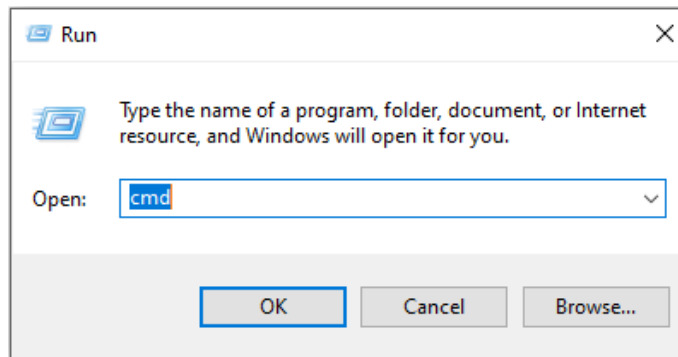


Figure 13.14 – Using Run

Once in the command-line interface, type `nslookup example.com`, and then press *Enter*. Once run, my output was as follows:

```
C:\>nslookup example.com
Server:  cdns01.comcast.net
Address: 2001:558:feed::1

Non-authoritative answer:
Name:    example.com
Addresses: 2606:2800:220:1:248:1893:25c8:1946
          93.184.216.34
```

The request first went to my **Internet Service Provider (ISP)**, and then `cdns01.comcast.net` resolved the IP address.

**Note**

When on a macOS and/or Linux, using the `dig` command will provide similar results.

DNS can suffer from malicious activity. In the next section, let's discuss some of the security issues related to DNS.

## Securing DNS

DNS has been used to resolve addresses for many years. In the early days of the internet, there wasn't any consideration of securing the protocol. As a result, it can suffer from attacks such as cache poisoning and spoofing.

First, let's outline what can happen if a malicious actor poisons the cache.

### Poisoning the cache

DNS is essential to any network. When clients request an IP address, they are trusting the server to provide an accurate address. If the address is incorrect, the client might be redirected to a bogus site.

DNS RRs have a TTL value that represents how long they can live in the cache, which is a temporary holding area for DNS records. The cache is an important concept, as it provides a way to send a quicker response to a query.

Cache poisoning injects spoofed information into a DNS server's cache. This attack is used by malicious actors to misdirect clients requesting a DNS resolution. The results of the attack will depend on what server was poisoned. On a **Local Area Network (LAN)**, all local users will be affected. However, poisoning the cache of an ISP server will have widespread effects.

It's essential to protect against attacks that disrupt DNS. Let's examine ways to achieve this.

### Defending DNS

DNS servers can represent a vulnerable target. To protect the server, the network administrator can employ a few techniques:

- Use cache locking, which controls how and when the cache can be overwritten.
- Restrict zone transfers to only respond to trusted servers.
- Use **DNS Security Extensions (DNSSEC)**, which provides data authentication and integrity.

Additionally, all other standard good practice techniques to secure DNS servers should be used. Techniques include keeping the servers updated and patched and using firewalls that restrict access to only authorized entities.

## Summary

In this chapter, we outlined the purpose of DNS, reviewed the types of servers, and then compared the way DNS is transported. We then evaluated the types and classes of RRs and then analyzed the structure and information contained within an RR. Next, we examined a DNS packet and drilled down into the header fields and flags, analyzed the packet structure, and reviewed the query section.

By now, you should have a solid understanding of the DNS query and response process and can recognize the TTL value, which indicates how long the value can remain in the cache. In addition, we saw how to examine DNS statistics in Wireshark and how we can test DNS by using tools such as `nslookup` and `dig`. We then summarized by discussing potential threats along with general advice on how to secure DNS.

In the next chapter, we will begin by explaining the need for **Dynamic Host Configuration Protocol (DHCP)** and review the purpose of this essential protocol. So that you understand how an IP address is obtained from the DHCP server, I'll outline the **Discover Offer Request Acknowledge (DORA)** process. I'll then dissect a DHCP header and review all the field value flags and port numbers, and then finish with a DHCP example.

## Questions

Now, it's time to check your knowledge. Select the best response, and then check your answers, which can be found in the *Assessment* appendix:

1. \_\_\_\_\_ servers are name servers that respond to queries housed within the root zone and are an essential component of resolving addresses.
  - A. Recursive
  - B. Root
  - C. Redundant
  - D. Caching
2. Within any DNS structure, there must be a(n) \_\_\_\_\_ DNS server that holds a master set of records for the domain and is updated whenever there is a change.
  - A. recursive
  - B. redundant
  - C. authoritative
  - D. caching

3. There are many types of DNS RR. A type 12 \_\_\_\_\_ converts an IP address to a domain name.
  - A. AXFR
  - B. MX
  - C. AAAA
  - D. PTR
4. When set, the \_\_\_\_\_ flag will indicate whether the message is shortened because it exceeds the limit of 512 bytes for a UDP packet.
  - A. recursive
  - B. caching
  - C. blunted
  - D. truncated
5. Experiencing a disproportionately large amount of DNS responses can indicate that a DNS \_\_\_\_\_ attack may be at play.
  - A. spoofing
  - B. DoS
  - C. truncating
  - D. cache poison
6. Within the DNS question section, the \_\_\_\_\_ is the requested resolution.
  - A. name
  - B. type
  - C. class
  - D. cache
7. One way to troubleshoot DNS issues on a Windows machine is to use the \_\_\_\_\_ command-line tool.
  - A. baseline
  - B. boost
  - C. nslookup
  - D. dig

## Further reading

Please refer to the following links for more information:

- Visit <https://www.cloudflare.com/learning/dns/glossary/dns-root-server/> to learn more on DNS root servers.
- Verisign has an insightful discussion on how DNS works. Learn more by visiting [https://www.verisign.com/en\\_US/website-presence/online/how-dns-works/index.xhtml](https://www.verisign.com/en_US/website-presence/online/how-dns-works/index.xhtml).
- Visit <https://www.cloudns.net/blog/dns-history-creation-first/> for a brief history of DNS.
- To view a detailed infographic on all the RFCs involved in creating and refining the DNS protocols we use today, visit: <https://emaillab.jp/dns/dns-rfc/>.
- To see an example of a simple zone file, visit [https://docs.fedoraproject.org/en-US/Fedora/12/html/Deployment\\_Guide/s2-bind-zone-examples.html](https://docs.fedoraproject.org/en-US/Fedora/12/html/Deployment_Guide/s2-bind-zone-examples.html).
- For more information on DNS statistics in Wireshark, visit [https://www.wireshark.org/docs/wsug\\_html\\_chunked/ChStatDNS.html](https://www.wireshark.org/docs/wsug_html_chunked/ChStatDNS.html).
- Visit <https://datatracker.ietf.org/doc/html/rfc1035> for details on domain names and how to properly implement and configure a DNS ecosystem.
- For an insightful primer on DNS, visit <https://www2.cs.duke.edu/courses/fall16/compsci356/DNS/DNS-primer.pdf>.
- For a comprehensive review of the field values in a DNS header, visit <https://isc.sans.edu/forums/diary/When+attackers+use+your+DNS+to+check+for+the+sites+you+are+visiting/16955/>.
- To learn more about how the DNS truncated flag is used, visit <https://support.f5.com/csp/article/K91537308>.
- To learn more about DNS response time, visit <https://www.keycdn.com/support/reduce-dns-lookups>.

- Discover ways to keep your DNS server safe by visiting: <https://securitytrails.com/blog/8-tips-to-prevent-dns-attacks>.
- To see a list of DNS parameters, visit <https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-5>.
- Visit <https://www.dnsfilter.com/blog/c2-server-command-and-control-attack> to learn how a C&C server is used to communicate with an infected host.





# 14

## Examining DHCP

Whenever you first boot up your machine to start your day, your **operating system (OS)** moves through a series of events. One of those events is obtaining an **internet protocol (IP)** address. The protocol that's used to obtain your IP address is known as **Dynamic Host Configuration Protocol (DHCP)**. In this chapter, we'll begin by reviewing the purpose of DHCP. Then, we step through the DHCP process, which has four steps: **discover**, **offer**, **request**, and **acknowledgment**, also called the **DORA** process.

To familiarize ourselves with the elements that make the DHCP process possible, we'll take a closer look at the field values in a DHCP header. In addition, you'll get a better understanding of the role of the client and server during this process. We'll also take a look at DHCPv6, which is used to provide an IP address to the growing number of networks that are moving toward employing IPv6. We'll also review some of the security aspects of this essential protocol and then provide an example of a DHCP transaction.

In this chapter, we will cover the following topics:

- Recognizing the purpose of DHCP
- Stepping through the DORA process
- Dissecting a DHCP header
- Following a DHCP example

## Recognizing the purpose of DHCP

Every host on the network must have a unique IP address to communicate with other network hosts. Some hosts, such as servers and printers, have a hardcoded IP address since they don't want the IP address to change. However, the vast majority of hosts have a dynamically assigned IP address.

DHCP is a method that's used on a network to provide configuration information to hosts on a network. In addition to providing an IP address, most responses provide additional information. This information can include the **Domain Name Service (DNS)** server's IP address and **Network Time Protocol (NTP)**, which provides the time service for the network.

DHCP lets you conserve IP addresses on a busy network. It also removes the need for the tedious and unnecessary hardcoding of individual IP addresses.

DHCP is a client-server model, where hosts request an IP address, and the server responds with an IP address that is *leased* for a period. The lease time can vary as it is generally up to the network administrator to configure the different parameters of a DHCP server.

When you're requesting an IP address, a client will receive at least one, but in many cases, two are offered since many networks use multiple servers for redundancy.

On an enterprise network, DHCP will use the following elements:

- **DHCP clients:** These are hosts that have been configured to obtain an IP address automatically in their network bindings.
- **DHCP servers:** These issue IP addresses to clients on the network from a pool of usable addresses, along with configuration data for specific periods.
- **DHCP relay agent:** This is optionally used when the DHCP server is on a different subnetwork than the DHCP client. The relay agent receives requests and directs the (unicast) message to the DHCP server(s) on another subnetwork on behalf of the client.

Whenever a host joins the network or wakes up, it will begin the DHCP process to obtain an IP address and begin communicating. To make this possible, the host must be configured to obtain an IP address automatically.

## Configuring the client's IP address

In most cases, DHCP is automatically configured on a host computer – for example, go to **Network Connections** and select the properties of your network interface card, **Internet Protocol Version 4 (TCP/IPv4) Properties**. Once there, you will likely see the **Obtain an IP address automatically** setting, as shown in the following screenshot:

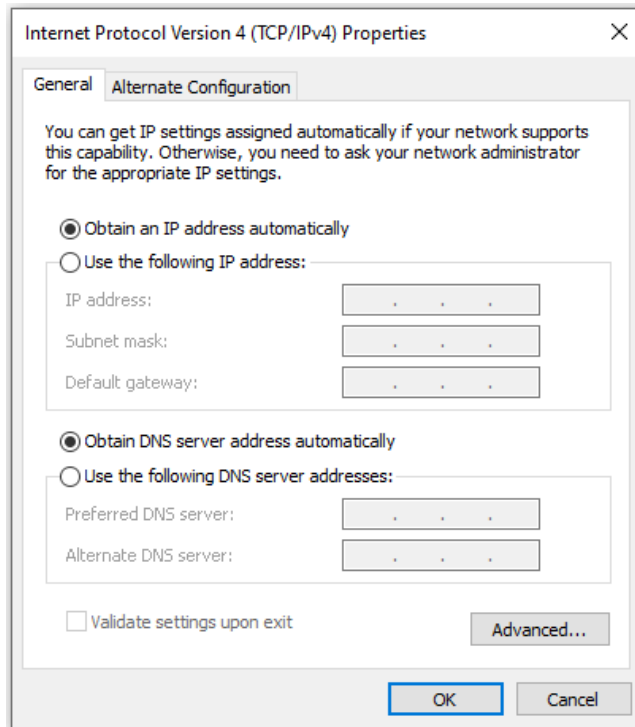


Figure 14.1 – Internet Protocol Version 4 (TCP/IPv4) Properties

Most of the hosts on a **Local Area Network (LAN)** are configured to obtain an IP address automatically. Whenever a client boots up, it will begin obtaining an IP address by attempting to reach a DHCP server. However, there will be times when the DHCP server won't be on the same subnetwork as the client. In that case, a relay agent must be used.

## Using a DHCP relay agent

On an IPv4 subnetwork, clients that rely on obtaining an IP address using DHCP begin the process using a broadcast out on the network. If there isn't a DHCP server on the subnetwork, the client will need a way to reach the server. Most routers do not forward broadcasts, so if a DHCP server is on the other side of the router, the client will need to use a relay agent to request an IP address on their behalf.

The relay agent is configured with the IP address of the DHCP server. The agent resides on the interface of the router where the DHCP clients reside so that it can respond to DHCP broadcasts. Once a discover packet has been accepted, the relay agent will communicate directly with the DHCP server using unicast packets.

The relay acts as a proxy and provides a bridge between the DHCP client and server, as shown in the following diagram:

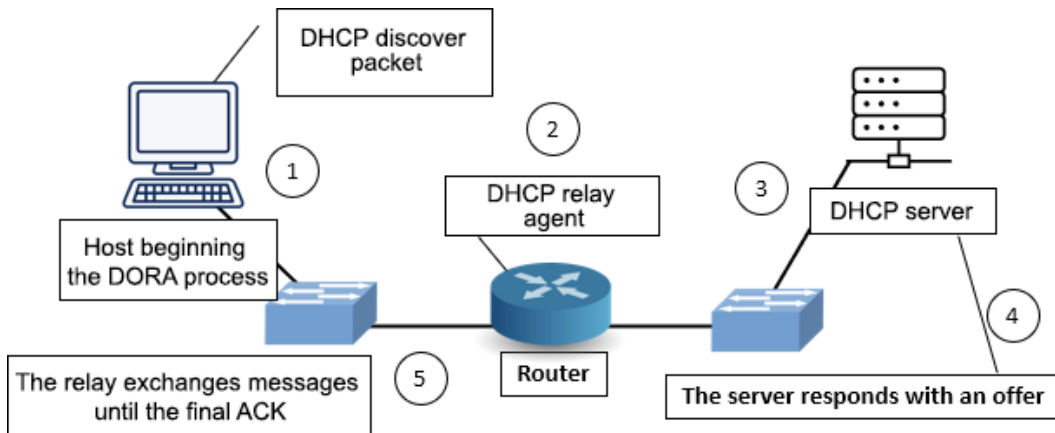


Figure 14.2 – Using a DHCP relay agent on a router

On a subnetwork that uses a relay agent, the process for obtaining an IP address using DHCP works in the following manner:

1. The client sends a discovery broadcast out on the subnetwork.
2. The relay agent collects the request on behalf of the client.
3. The request is repackaged and then sent via unicast directly to the DHCP server.
4. The server responds to the relay agent with an offer that contains an IP address, along with any optional information that's been requested from the client.
5. The relay agent forwards the offer to the client and will continue to exchange messages between the client and server, until the final acknowledgment.

DHCP provides a way for an IPv4 host to obtain an IP address. In the next section, we'll outline the ways that clients on a network that are using IPv6 can obtain an IP address.

## Working with IPv6 addresses

While many LANs today primarily use IPv4, we are starting to see a small percentage of organizations adopting IPv6. When using IPv6, there are three ways to configure an IP address on a host:

- Manually configure an address on hosts that require a static IP address, such as servers and printers.
- Use **StateLess Address AutoConfiguration (SLAAC)** to automatically assign an IP address.
- Use DHCPv6 to issue an address from the preconfigured pool of addresses.

While a few hosts on a LAN will have a manually assigned IPv6 address, the vast majority will need to obtain a network address in some other way. Let's compare ways an IPv6 client can obtain an IP address. We'll start by discussing how SLAAC is used.

### Employing SLAAC

On an IPv6 network, SLAAC is the preferred way to assign an IP address. When you're using SLAAC, the network administrator will need to configure a router to publish its prefix. The prefix is then used so that each client can generate their own IPv6 address.

SLAAC creates an IPv6 address by doing the following:

1. First, it takes a 12-digit (48-bit) **Media Access Control (MAC)** address from the client's **Network Interface Card (NIC)**.
2. Then, it separates the **Organizationally Unique Identifier (OUI)** and the NIC serial number.
3. Finally, it inserts the 16-bit identifier, 0xFFFE, between the OUI and the NIC serial number.

Once complete, it will appear as follows:

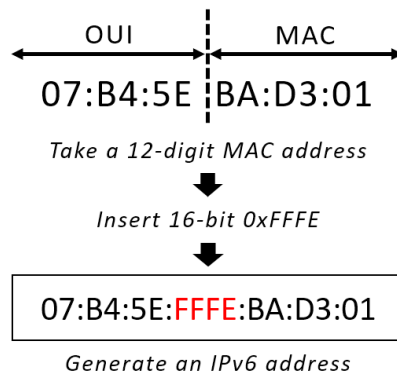


Figure 14.3 – Generating an IPv6 using SLAAC

When you're using SLAAC, generating an IPv6 address for network hosts is done automatically. However, DHCPv6 can still be used as it can offer additional configuration options for hosts.

## Providing information with DHCPv6

While SLAAC can provide an efficient way to automatically create an IPv6 address, the host may need additional information, which can be provided by using DHCPv6.

Some of the configuration data a DHCPv6 server can supply to clients on the network includes the following, along with the corresponding **Request for Comment (RFC)**, where you can find more information:

Option	Description	Reference
21	The Session Initiation Protocol (SIP) server's domain name list	RFC 3319
23	Domain Name Server (DNS) Recursive Name Server	RFC 3646
31	Simple Network Time Protocol (SNTP) server list	RFC 4075

Table 14.1 – DHCPv6 options

For example, if a client has obtained an IP address using SLAAC, they can send an `Information-Request` packet message with an **Option Request** outlining the requested configuration parameters. The following is an example:

```
~ Option Request
  Option: Option Request (6)
  Length: 4
  Requested Option code: DNS recursive name server (23)
  Requested Option code: Domain Search List (24)
```

Figure 14.4 – DHCPv6 Option Request

DHCP is a widely used protocol that enables hosts to automatically obtain an IP address. However, it's important to address possible security risks when employing DHCP.

## Addressing security issues

DHCP became formalized as a protocol in 1993. During that time, little thought was put into providing methods to secure DHCP and DHCP servers from malicious activity. However, there is a chance that a rogue device may launch an attack, such as a DHCP starvation attack.

One primary reason this is possible is that when using DHCP, there is no native authentication or authorization between the client and the server. Because of this, the network administrator should take steps to ensure that a DHCP server interacts only with a legitimate client on the network, before any DHCP interaction.

One way to do this is to use DHCP snooping. When configured on a layer 2 device, such as a switch, DHCP snooping will monitor the traffic and do the following:

- Allow valid DHCP client messages to communicate with a DHCP server.
- Allow valid DHCP server messages to communicate with a DHCP client.
- Deny and drop traffic from *untrusted* sources.

In addition, in a Windows Server environment, DHCP servers must be authorized in Active Directory before they can start leasing addresses. This is a validation check to ensure clients do not receive addresses from rogue DHCP servers.

In the next section, we will describe the process of obtaining an IP address using the DORA process.



## Stepping through the DORA process

The DHCP process is an efficient way to provide addresses to clients. Because we want our IP address quickly, it uses **User Datagram Protocol (UDP)** as the transport layer protocol.

DHCP uses the following UDP ports:

- Clients use port 68.
- Servers and relay agents use port 67.

DHCPv6 uses the following UDP ports:

- Clients use port 546.
- Servers and relay agents use port 547.

You will see these ports in use when we examine DHCP traffic.

The DHCP process progresses through various states during a transaction, as outlined in the next section.

## Moving through DHCP states

During the process of a DHCP transaction, the client can be in one of six states. Understanding these states and what transpires will help during troubleshooting. These DHCP states are as follows:

- **Initiate:** The client begins the process by sending a broadcast on the network in search of a DHCP server.
- **Select:** The client will select an offer that's been sent by the server(s).
- **Request:** The client will formally request an offer from a server.
- **Bound:** Once the server sends an IP address, the client will remain in a bound state until the lease is exhausted.
- **Renew:** During this state, the client will request a renewal of the lease. At this point, one of two things can happen:
  - If an ACK is sent by the server, the lease is renewed, and the client reverts to the *bound* stage.
  - If an ACK is not sent by the server and 87.5% of the time has passed, the client moves to the *rebind* stage.

- **Rebind:** During this state, the client will attempt to rebind. At this point, one of two things can happen:
  - If an ACK is sent by the server, the lease is renewed, and the client reverts to the *bound* stage.
  - If the lease expires or the server sends a **Negative Acknowledgment (NAK)**, the client will revert to the *initiate* stage.

**Note**

When you're working with a packet capture, you may see Wireshark identify the application as bootstrap protocol (`bootstrap`) since that is the original name of the protocol, as outlined in RFC 951. However, to view only DHCP traffic, use the `dhcp` (lowercase) filter in the display filter toolbar.

When you're moving through the various stages of the DHCP process, there must be a way to keep the transactions together. Let's see what role the **Transaction Identifier (XID)** plays in this process.

## Defining the transaction ID

Throughout the DORA process, you will see an XID, which is randomly generated by the client. The server will adopt the same XID and both the client and the server will be able to recognize which transactions belong together.

In rare cases, the client will fail to provide an XID. In those cases, the server will use the **Client Hardware Address (chaddr)** to identify the client. Using `chaddr` is not normally advised as it may cause undesirable results. In addition, a malicious actor could attempt a starvation attack by flooding the DHCP server with multiple requests using a forged `chaddr`. This could prevent legitimate users from obtaining an IP address.

Whenever a client needs an IP address, it begins to move through the four-step DORA process. Let's dissect this procedure.

## Obtaining an IP address

The DORA process moves through four phases, as follows:

- **DHCP Discover:** The process begins with the client sending a broadcast out on the network, asking for an IP address.
- **DHCP Offer:** The server responds by offering an IP address. Within the reply, there are generally many parameters.

- **DHCP Request:** The client formally requests an IP address.
- **DHCP ACK:** The server offers an IP address.

**Note**

The DHCP process is commonly described using four steps. However, there are actually *two* operations – the client asking for an IP address and the client formally requesting the IP address. As a result, the XID may be different for the first two and the last two packets.

In this section, we'll outline what takes place in each of the steps, starting with the client attempting to locate a server.

## Searching for a DHCP server

When a host needs an IP address, it begins the DORA process by issuing a DHCP `Discover` broadcast message out onto the subnetwork. The `chaddr` is listed, along with a client-selected XID.

Within the `Discover` packet, there may be DHCP options requesting additional configuration parameters, along with possible suggestions for the IP address.

If there isn't a DHCP server on the same subnet as the DHCP client, a relay agent will need to request an IP address from a DHCP server on another subnetwork.

Once the broadcast has been sent out on the LAN, one or more DHCP servers will respond with an offer.

## Offering an address

Once the server(s) accepts the `Discover` packet, the next step is to check the available IP addresses in the DHCP pool. If an address is available, the server(s) will record the XID and return a DHCP offer.

The offer will have a network address for the client to use, along with the lease time parameters. In addition, there may be responses to the requested options that have been sent by the client in the DHCP `Discover` packet.

**Note**

Although the client may have requested a specific IP address (for example, one that was previously used), the server does not have to honor the request. However, it's best practice to offer the client a requested IP address to avoid any conflicts.

At this point, the client will receive an offer from one or more DHCP servers.

## Requesting an address

Once the client receives an offer, the next step is to formally request an IP address. If the DHCP client receives multiple offers, it will check each one and either accept, drop, or reject the offer, as follows:

- If the client accepts the DHCP offer from the server, a DHCP request message is sent as a *broadcast* that must include the **server identifier** option, along with the IP address, as supplied by the DHCP server.
- When the client receives the DHCP offer message, it will check to make sure the DHCP offer XID matches the XID of the DHCP discover message. If they do not match, the client will discard the DHCP offer.
- If the client receives more than one offer, it will reject the offer by sending a DHCP decline message to one of the servers.

The last step is for the server to respond to the client with either an IP address or **DHCP NAK**, as we'll see next.

## Confirming the parameters

After sending an offer for an IP address, the server waits for a response from the client. Once the server receives the broadcast, it will send the **DHCP Acknowledgment (DHCP ACK)** to the client to indicate the completion of the four-packet DHCP Discovery process.

The acknowledgment packet contains an IP address, along with any configuration options requested by the client in the DHCP Request packet.

However, there are times when the server won't be able to supply the client with an IP address. If that happens, the server will send a DHCP NAK to the client and the client must start the DORA process again.

DHCP NAK messages are rare, but if you are seeing multiple DHCP NAK messages on your network, you will need to assess the root cause as to why clients aren't getting an IP address. Some reasons for this are as follows:

- A misconfigured DHCP pool.
- The DHCP pool is exhausted (it has run out of IP addresses).
- A DHCP server is the victim of a starvation attack.
- A misconfigured DHCP relay agent.

The good news is that in most cases, the client will receive an IP address. After this, the client will check whether any other hosts are using the same IP address.

## Checking for duplicate IP addresses

Once the client has obtained an IP address, it will go through the process of making sure the newly assigned address is not already in use.

To check for a duplicate IP address, the following methods are used:

- On an IPv4 network, the client will broadcast an **Address Resolution Protocol (ARP)** request out onto the network and wait to see whether another host responds.
- On an IPv6 network, the client will send an **Internet Control Message Protocol (ICMPv6)** neighbor solicitation message.

If an IP address is already in use, the client must send a DHCP DECLINE to the server and begin the DORA process all over again. However, in most cases, the IP address won't have been assigned already, so the client can accept the IP address.

Once the client accepts an IP address and the server sends a final DHCP ACK, the client moves into the bound state until the lease time is exhausted. Next, let's see what's involved when a client leases an IP address from a server.

## Leasing an IP address

The idea behind the lease is that a network address is only needed for a short period and that at some point, the host will no longer need the IP address. This allows for a more fluid environment and efficient sharing of a limited amount of IP addresses.

The lease time defines how long the client can use the IP address, which can vary, depending on the environment.

## Defining the lease time

The DHCP lease time defaults at 24 hours (1,440 minutes). However, often, this value is set by the network administrator and can vary, as follows:

- In a highly volatile environment, where clients stay for a short period, such as at a gym or hair salon, setting the lease for 60-90 minutes would be appropriate.
- In a transient environment, where the clients come and stay for short periods, setting the lease time for 1 to 3 days would be appropriate.
- In a stable environment, where the hosts seldom move and clients stay for a lengthy period, setting the lease time for 1 to 3 weeks would be appropriate.

When the timer begins, the lease time moves through various states, as shown in the following diagram:

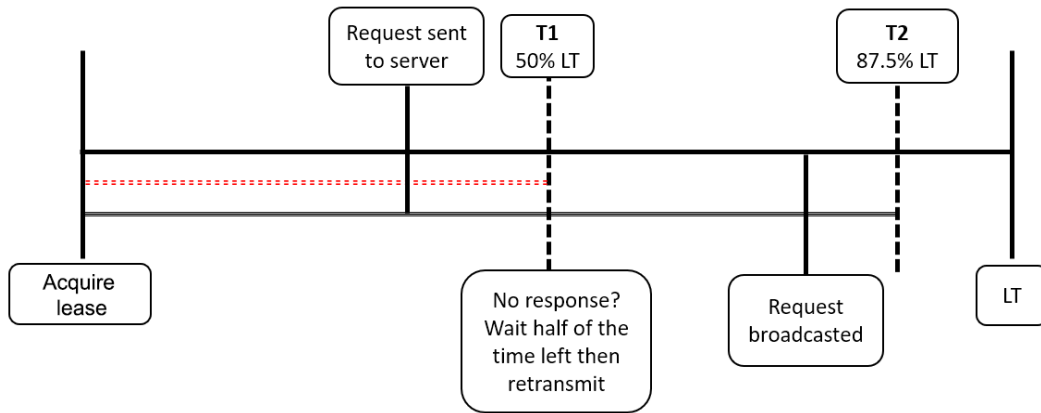


Figure 14.5 – DHCP lease time process

Once the final acknowledgment is sent from the server, the client begins a timer to monitor the lease time. The values for **Timer 1 (T1)** to **Timer 2 (T2)** are as follows:

- Rebind (T1) represents 50% of the lease time.
- Renew (T2) represents 87.5% of the lease time.

As the lease time moves from T1 to T2, various events take place. Let's outline this process, starting with events that occur before T1.

### Closing in on the renewal phase

Once the client acquires the lease, the timer begins. When the timer inches toward T1, the client may (or may not) choose to extend the lease. If the client would like to renew the IP address, a renewal packet is sent directly to the DHCP server. At this point, if the server sends a DHCP ACK, the lease is renewed, and the client reverts to the bound stage.

However, if an ACK is not sent by the server and 87.5% of the time has passed, the client moves to the rebind stage.

### Moving toward the rebind stage

If the server does not send a DHCP ACK in a timely manner, the client will attempt to rebind by broadcasting a DHCP Request message out on the network. If an ACK is sent by the server, the lease is renewed, and the client reverts to the **bound** stage.

However, if the lease expires or the server sends a NAK, the client must revert to the **initiate** stage to begin the process again.

Now that you understand the DORA process, let's take a look at the DHCP header and review the field values, flags, and port numbers.

## Dissecting a DHCP header

When reviewing a DHCP packet, you will see many fields in the header, as shown here:

UDP Header			
Opcode	Hardware Type	Hardware Length	Hops
Transaction ID Number			
Seconds since Boot		Flags	
Client IP Address			
Your (Client) IP Address			
Server IP Address			
Gateway IP Address			
Client Hardware Address			
Server Hostname			
Boot File			
Options			

Figure 14.6 – DHCP packet structure

To follow along, obtain a copy of `DHCP.cap` from [https://wiki.wireshark.org/uploads/\\_\\_moin\\_import\\_\\_/attachments/SampleCaptures/dhcp.pcap](https://wiki.wireshark.org/uploads/__moin_import__/attachments/SampleCaptures/dhcp.pcap) and open it in Wireshark. Expand the DHCP header in **Frame 1**, as follows:

```

~ Dynamic Host Configuration Protocol (Request)
  Message type: Boot Request (1)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x00003d1e
  Seconds elapsed: 0
  > Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0
  Your (client) IP address: 0.0.0.0
  Next server IP address: 0.0.0.0
  Relay agent IP address: 0.0.0.0
  Client MAC address: 00:0b:82:01:fc:42
  Client hardware address padding: 00000000000000000000
  Server host name not given
  Boot file name not given

```

Figure 14.7 – A DHCP header

As we can see, each DHCP header contains key fields and identifiers. Let's look at these in more detail.

## Examining DHCP field values

Within the header, you will see key fields that help the DORA process move through the various states. The field values are as follows:

- **Operation Code (op):** This indicates the message type; for example, 1 = BOOTREQUEST or 2 = BOOTREPLY. In Wireshark, this is displayed as Message type, as shown in the preceding screenshot.
- **Hardware Type (htype):** This defines the type of connection for the session. Commonly, this field is listed as Ethernet (0X01) because often, the message is coming from either an Ethernet or 802.11 client.
- **Hardware Length (hlen):** This defines the length of the hardware address in bytes. For example, in **Frame 1**, the length is 6 (bytes), which is a common size for a MAC address.
- **Hops:** This field is optionally used by a relay agent. Generally, this is set to 0.
- **XID:** This is selected by the client to keep track of messages between the client and the server.



- **Seconds (secs):** This specifies the time that has elapsed since beginning the DORA or renewal process. This value is set by the client.
- **Flags:** There are two flags, as follows:
  - **Broadcast flag:** This option is set by the client to indicate how the server should send the message back to the client.
  - **Reserved flag:** Not used.
- **Client IP Address (ciaddr):** This is the current IP address of the client, which will only have a value if the client is in one of three states: bound, renew, or rebinding.
- **Your IP Address (yiaddr):** The IP address, as supplied by the DHCP server.
- **Server IP Address (siaddr):** The IP address of the DHCP server responding to the requests.
- **Relay Agent IP Address (giaddr):** The IP address of the relay agent or gateway.
- **chaddr:** The client's MAC address.
- **Server Host Name (sname):** This is an optional value that contains the name of the server.
- **Boot file:** This is an optional file that's used in a **Preboot Execution Environment (PXE)**, such as a thin client, where the device is booted via the network.
- **Options:** DHCP options are listed after the main header.

**Note**

At the end of the header fields, you will see a field called `Magic cookie:` DHCP, which alerts network hosts that the options are DHCP and not the legacy Bootstrap Protocol (BOOTP).

When you're examining DHCP traffic, it's common to see messages such as DHCPDISCOVER and DHCPOFFER. However, there are several other messages that DHCP uses to convey information.

## Understanding DHCP messages

DHCP messages are exchanged between the client and the server to request information such as configuration data or provide status updates. The various types of messages, along with the purpose, are outlined in the following table:

DHCP Message	Sending Body	Purpose
DHCPDISCOVER (Discover)	Client	A broadcast that is sent out on a subnet to discover available DHCP servers.
DHCPOFFER (Offer)	Server	A reply to the client offering an IP address and additional configuration details.
DHCPREQUEST (Request)	Client	<p>This is sent to DHCP servers for one of the following reasons:</p> <ul style="list-style-type: none"> <li>• Formerly, to request configuration details from a server and reject all other offers</li> <li>• To confirm the accuracy of a previously allocated address after a reboot</li> <li>• To request an extension on an assigned IP address lease</li> </ul>
DHCPACK (Acknowledgement)	Server	An acknowledgment to the client confirming an IP address, along with any additional configuration details.
DHCPNAK (Negative Acknowledgment)	Server	Sent to the client denying a request for a specific IPv4 address because it is already in use or is requesting an address that is on a different subnet network.
DHCPDECLINE (Decline)	Client	Sent to the server surrendering their IPv4 address and canceling any remaining lease time.
DHCPINFORM (Inform)	Client	Sent to the server while requesting <i>only</i> configuration details, not an IPv4 address.

Table 14.2 – DHCP messages

After the DHCP header, several options might be listed. In the next few sections, we'll take a brief look at some of the DHCP options that we may encounter.

## Comparing DHCP options

In the DHCP header in **Frame 1** of `DHCP.cap`, immediately after the `Magic cookie` identifier, we can see a list of options, as shown here:

```
> Option: (53) DHCP Message Type (Request)
> Option: (61) Client identifier
> Option: (50) Requested IP Address (192.168.0.10)
> Option: (54) DHCP Server Identifier (192.168.0.1)
> Option: (55) Parameter Request List
> Option: (255) End
```

Figure 14.8 – DHCP header

DHCP can have multiple options, as outlined here: <https://www.iana.org/assignments/bootp-dhcp-parameters/bootp-dhcp-parameters.xhtml>.

Some of these options are as follows:

- **DHCP Option 3:** Router (or gateway) address
- **DHCP Option 6:** DNS server address(es)
- **DHCP Option 50:** Requested IP address
- **DHCP Option 51:** Lease time for this IP address
- **DHCP Option 53:** Type of message

Within each option, various parameters will be listed. For example, in **Frame 1**, we can see `Option: (61) Client identifier`, which lists the client's MAC address, as shown here:

```
~ Option: (61) Client identifier
  Length: 7
  Hardware type: Ethernet (0x01)
  Client MAC address: 00:0b:82:01:fc:42
```

Figure 14.9 – DHCP Option 61

While there are several options, the most common option is `Option: (53)`, which describes the type of message. For example, in the following screenshot, I have expanded the `Option` field, where you see the type of message is a DHCP Release:

```

  ▾ Option: (53) DHCP Message Type (Release)
    Length: 1
    DHCP: Release (7)

```

Figure 14.10 – DHCP Release option

Now that we understand the general role and purpose of DHCP, along with the header field values and options, let's step through an example of the DORA process.

## Following a DHCP example

When a host boots up and attempts to join the network, one of the first things they must do is obtain an IP address.

In this section, we'll look at what you may see when a client issues a DHCP Release. Then, we'll step through the DORA process and outline the details of each of the steps that are taken when a client obtains an IP address.

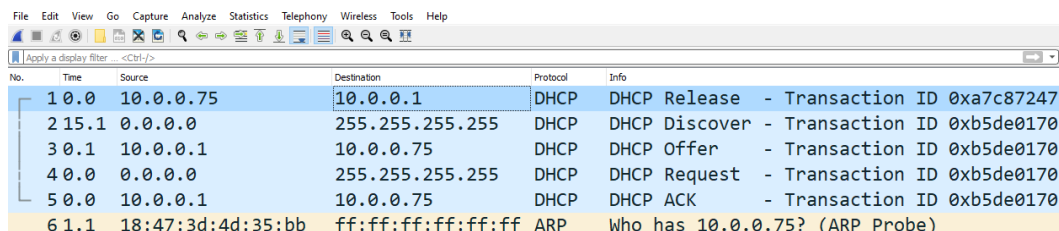
Let's start by seeing what happens when a client releases their network address.

## Releasing an IP address

There are times when you may need to release and renew an IP address, such as during a troubleshooting exercise.

One way to release an IP address on a Windows machine is by opening a Command Prompt and typing `ipconfig /release`. To renew the IP address, type `ipconfig /renew`.

I did a manual release and then renewed while capturing traffic with Wireshark. The results are shown here:



The screenshot shows the Wireshark network protocol analyzer interface. The packet list pane on the left shows a sequence of packets: a DHCP Release (No. 1), followed by a DHCP Discover (No. 2), a DHCP Offer (No. 3), a DHCP Request (No. 4), a DHCP ACK (No. 5), and an ARP probe (No. 6). The packet details pane on the right shows the expanded details of the selected DHCP Release packet (No. 1), indicating it is a Release message with Transaction ID 0xa7c87247. The packet bytes pane at the bottom shows the raw data of the selected packet.

No.	Time	Source	Destination	Protocol	Info
1	0.0	10.0.0.75	10.0.0.1	DHCP	DHCP Release - Transaction ID 0xa7c87247
2	15.1	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xb5de0170
3	0.1	10.0.0.1	10.0.0.75	DHCP	DHCP Offer - Transaction ID 0xb5de0170
4	0.0	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xb5de0170
5	0.0	10.0.0.1	10.0.0.75	DHCP	DHCP ACK - Transaction ID 0xb5de0170
6	1.1	18:47:3d:4d:35:bb	ff:ff:ff:ff:ff:ff	ARP	Who has 10.0.0.75? (ARP Probe)

Figure 14.11 – DHCP Release and Renew

Finally, in **Frame 6**, once I accepted the IP address, I sent an ARP probe to check for a duplicate IP to make sure my IP address is not in use by another host.

Now, let's move through the DORA process together. To follow along, open `DHCP.cap` in Wireshark so that you can see the four steps of the DORA process in the first four packets, as shown here:

No.	Time	Source	Destination	Protocol	Info
1	0.0	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x3d1d
2	0.0	192.168.0.1	192.168.0.10	DHCP	DHCP Offer - Transaction ID 0x3d1d
3	0.0	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x3d1e
4	0.0	192.168.0.1	192.168.0.10	DHCP	DHCP ACK - Transaction ID 0x3d1e

Figure 14.13 – The four-packet DORA process

#### Note

Although various options have been listed, we'll mainly focus on the details of the transaction.

The process begins with the client sending a broadcast message out on the network.

## Broadcasting a discover packet

In **Frame 1**, we can see the DHCP `Discover` message, as shown here:

```

Dynamic Host Configuration Protocol (Discover)
  Message type: Boot Request (1)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0xb5de0170
  Seconds elapsed: 0
  Bootp flags: 0x0000 (Unicast)
    0... .... = Broadcast flag: Unicast
    .000 0000 0000 0000 = Reserved flags: 0x0000
  Client IP address: 0.0.0.0
  Your (client) IP address: 0.0.0.0
  Next server IP address: 0.0.0.0
  Relay agent IP address: 0.0.0.0
  Client MAC address: 18:47:3d:4d:35:bb
  Client hardware address padding: 00000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP

```

Figure 14.14 – DHCP Discover packet

Within the `Discover` message, you can see the following key values:

- `Message type: Boot Request (1)`
- `Transaction ID: 0xb5de0170`
- `Your (client) IP address: 0.0.0.0`
- `Client MAC address: 18:47:3d:4d:35:bb`

At this point, the client's IP address is `0.0.0.0`. `Transaction ID` is `0x00003d1d`, which both the client and server will use to keep track of the transaction. Within the options of **Frame 1**, we don't see a request for a specific IP address. However, if requested, the server would typically offer the last address that was used by the client.

Once the broadcast is sent out on the LAN, one or more DHCP servers will respond with an offer.

## Delivering an offer

After the server(s) accepts the `Discover` packet, the next step is to check the available IP addresses in the DHCP pool. If an address is available, the server(s) will record the `XID` and return a DHCP Offer. The next step is to send an offer directly to the DHCP client. Keep in mind that there may be more than one server sending an offer to the client.





At this point, the client IP address is still 0.0.0.0, and Transaction ID: 0x00003d1d is the same as it was for the Discover message.

During the DORA process, the DHCP client may receive multiple offers. If this happens, the client will check each one and either accept, drop, or reject the offer. If the client chooses to accept the offer, the next step is to send a formal request to the server.

## Requesting an IP address

Once an offer has been accepted, the client sends a DHCP request as a *broadcast* out on the network.

As shown in **Frame 3**, the request will include Option: (54) DHCP Server Identifier (192.168.0.1), along with Option: (50) Requested IP Address (192.168.0.10), as supplied by the DHCP server:

```

  ▾ Dynamic Host Configuration Protocol (Request)
    Message type: Boot Request (1)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x00003d1e
    Seconds elapsed: 0
  ▾ Bootp flags: 0x0000 (Unicast)
    0... .... = Broadcast flag: Unicast
    .000 0000 0000 0000 = Reserved flags: 0x0000
    Client IP address: 0.0.0.0
    Your (client) IP address: 0.0.0.0
    Next server IP address: 0.0.0.0
    Relay agent IP address: 0.0.0.0
    Client MAC address: 00:0b:82:01:fc:42
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  ▸ Option: (53) DHCP Message Type (Request)
  ▸ Option: (61) Client identifier
  ▸ Option: (50) Requested IP Address (192.168.0.10)
  ▸ Option: (54) DHCP Server Identifier (192.168.0.1)
  ▸ Option: (55) Parameter Request List
  ▸ Option: (255) End
    Padding: 00
```

Figure 14.16 – DHCP Request packet



Within the ACK message, you will see the following key values:

- Message type: Boot Reply (2)
- Transaction ID: 0x00003d1e
- Your (client) IP address: 192.168.0.10
- Client MAC address: 00:0b:82:01:fc:42

The client's IP address is now 192.168.0.10 and Transaction ID is 0x00003d1e.

Within these options, we can see details of the lease time and other parameters, as follows:

```
  v Option: (58) Renewal Time Value
    Length: 4
    Renewal Time Value: (1800s) 30 minutes
  v Option: (59) Rebinding Time Value
    Length: 4
    Rebinding Time Value: (3150s) 52 minutes, 30 seconds
  v Option: (51) IP Address Lease Time
    Length: 4
    IP Address Lease Time: (3600s) 1 hour
  v Option: (54) DHCP Server Identifier (192.168.0.1)
    Length: 4
    DHCP Server Identifier: 192.168.0.1
  v Option: (1) Subnet Mask (255.255.255.0)
    Length: 4
    Subnet Mask: 255.255.255.0
```

Figure 14.18 – DHCP acknowledgment options

At this point, the client has an IP address, and the lease timer begins counting down. The client can now begin transacting with other hosts on the network.

## Summary

DHCP is an essential protocol that's used on nearly all LANs. When it's configured properly, DHCP is a seamless way to assign IP addresses, with little or no intervention. In this chapter, we covered the purpose of DHCP and learned about its impressive role in supplying a host with an IP address, along with configuration details. We discovered the role of a relay agent, outlined the various ways IPv6 obtains an IP address, and briefly discussed security issues when working with DHCP.

Then, we looked at the **DORA** process – **discover**, **offer**, **request**, and **acknowledgment** – and examined the mechanics of the DHCP lease time. You should now have a better understanding of the DHCP header field values, along with the types of messages. In addition, you should be able to recognize the various DHCP options. We summarized this by bringing everything together and stepping through a DHCP example. We moved from the initial broadcast on the network, to the offer and formal requests, to the host obtaining an IP address so that they can communicate on the network.

In the next chapter, we'll examine **Hypertext Transfer Protocol (HTTP)**, an application layer protocol that's used to browse the web. We'll learn about HTTP, including available versions and connection methods, and dissect the headers and fields for both the client and the server. After that, we'll compare the request and response messages and see what we can learn when we follow an HTTP stream.

## Questions

Now, it's time for you to check your knowledge. Select the best response, and then check your answers, which can be found in the *Assessment* appendix:

1. The DHCP process begins with a(n) \_\_\_\_\_ packet sending a broadcast out on the network asking for an IP address.
  - A. Offer
  - B. Request
  - C. Discover
  - D. Appeal
2. The DHCP \_\_\_\_\_ packet is sent by the client and formally requests an IP address from the server.
  - A. Offer
  - B. Request
  - C. Petition
  - D. Appeal

3. One way to obtain an IPv6 address is to use \_\_\_\_\_ to automatically assign an IP address.
  - A. Auto Request
  - B. Ready Option
  - C. Pine Option
  - D. SLAAC
4. DHCPv6 clients use UDP port 546. Servers and relay agents use port \_\_\_\_\_.
  - A. 546
  - B. 547
  - C. 68
  - D. 67
5. In a DHCP header, the \_\_\_\_\_ defines the length of the hardware address in bytes.
  - A. htype
  - B. XID
  - C. ciaddr
  - D. hlen
6. DHCP can have several options. Option 50 represents \_\_\_\_\_.
  - A. The DNS server address(es)
  - B. The requested IP address
  - C. The lease time for this IP address
  - D. The type of message
7. When you're sending the DHCP Release message, the way the server recognizes the client is by using both the ciaddr and the \_\_\_\_\_.
  - A. ciaddr
  - B. flags
  - C. giaddr
  - D. htype

## Further reading

Please refer to the following links for more information regarding the topics that were covered in this chapter:

- Visit <https://www.computernetworkingnotes.com/ccna-study-guide/dhcp-configuration-parameters-and-settings-explained.html> to learn more about DHCP configuration settings.
- To learn more about SLAAC, visit <https://www.networkacademy.io/ccna/ipv6/stateless-address-autoconfiguration-slaac>.
- Visit <https://www.networkworld.com/article/3297800/why-dhcp-days-might-be-numbered.html> to learn more about DHCP and DHCPv6.
- DHCPv6 has several options. To see a comprehensive list, go to <http://www.networksorcery.com/enp/protocol/dhcpv6.htm>.
- To see what's involved when a client only requires configuration information when using DHCPv6, visit [https://techhub.hp.com/eginfolib/networking/docs/switches/5130ei/5200-3942\\_l3-ip-svcs\\_cg/content/483572567.htm](https://techhub.hp.com/eginfolib/networking/docs/switches/5130ei/5200-3942_l3-ip-svcs_cg/content/483572567.htm).
- Learn more about DHCP snooping at <https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SXF/native/configuration/guide/swcg/snoodhcp.pdf>.
- Defend against DHCP starvation attacks. Visit [https://techhub.hp.com/eginfolib/networking/docs/switches/5940/5200-1022b\\_l3-ip-svcs\\_cg/content/491966649.htm](https://techhub.hp.com/eginfolib/networking/docs/switches/5940/5200-1022b_l3-ip-svcs_cg/content/491966649.htm) to learn more.
- For an extensive list of DHCP and DHCPv6 options, go to <https://www.incognito.com/tutorials/dhcp-options-in-plain-english/>.
- To learn more about how to use ARP after obtaining an IP address, go to <https://www.netmanias.com/en/post/techdocs/5999/dhcp-network-protocol/understanding-the-detailed-operations-of-dhcp>.



# 15

# Decoding HTTP

At some point, most of us have accessed a web page to download information. But just what's involved when retrieving a web page? In this chapter, we'll take a closer look at the **HyperText Transfer Protocol (HTTP)**. We'll start with an overview and review some of the objects and elements that we can obtain when requesting content. We'll then compare the available HTTP versions, 1.0, 1.1, and 2.0, along with the methods used, such as GET, POST, and HEAD. HTTP has three versions, each with a default method to establish and maintain a connection. So that you understand the mechanics of the different methods, we'll compare the differences between a non-persistent and persistent connection.

HTTP is a stateless protocol. You'll learn how HTTP uses cookies to maintain state by keeping track of the details of each transaction. To help you troubleshoot a web connection, we'll review what takes place during a transaction and examine the general format of request and response messages. We'll dissect the HTTP header and fields for both the client and the server. We'll then summarize this by following an HTTP stream, and then break down each of the elements when examining a transaction.

This chapter will address all of these by covering the following topics:

- Describing HTTP
- Keeping track of the connection
- Comparing request and response messages
- Following an HTTP stream



## Describing HTTP

HTTP is an application-layer protocol responsible for exchanging data between a client browser and a web server. HTTP allows us to gather, distribute, collaborate, and disseminate a wide range of data. Although a stateless protocol, HTTP has several request methods, error codes, and headers that allow us to access resources from many different types of applications.

### Note

HTTP is used to transport data across the network. Two other protocols that transport data include **File Transfer Protocol (FTP)** and **Simple Mail Transfer Protocol (SMTP)**. All are essential in providing an efficient way to exchange data.

In this section, we'll begin by outlining the definition of a web page and describe HTTP's function in retrieving objects from a website. We'll compare the role of client and server, outline the versions in use today, and then summarize with a review of the different types of HTTP methods.

Let's start by reviewing the elements of a web page.

## Dissecting a web page

Using a browser, such as Firefox or Chrome, a client requests a page by clicking on a hyperlink. The link directs the client to a website, which is a set of linked web pages, and the server sends the requested data to the client.

The first page we hit is called *index* or *default*, and that's where most of us start when searching for content. Once at the website, each page contains objects, such as a **HyperText Markup Language (HTML)** file, **Joint Photographic Experts Group (JPEG)** images, text, applications, and/or JavaScript. Most websites have many linked pages that are hyperlinked to other content within the site, along with a variety of information and objects that can be extracted.

At a high level, HTTP will retrieve a web page that can contain elements, as shown in the following graphic:

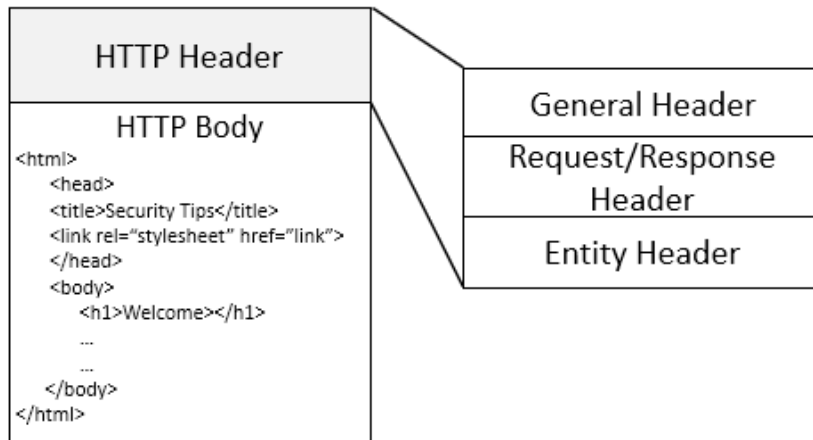


Figure 15.1 – A standard web page

Prior to downloading any objects, the client must first access the web page. Next, let's review how resources are located when requesting data from a website.

## Finding the target

When a client requests a web page, the user will click on a link to request an object using a **Uniform Resource Identifier (URI)**, which is used to identify an object (or resource) on the internet. Two *subsets* of a URI are a **Uniform Resource Locator (URL)** and a **Uniform Resource Name (URN)**.

In some cases, the client will request a specific page from the server. For example, if requesting `https://www.nist.gov/blogs/cybersecurity-insights`, the URL will break down in the following manner:

- **Hostname:** The name of the host is `www.nist.gov`.
- **Path:** The path to the specific resource is `/blogs/cybersecurity-insights`.

Once the request reaches the target, the next step is to interact with the server using either a standard or encrypted connection.

## Making the connection

When a client initiates a connection with a web server, they will use either **Transmission Control Protocol (TCP)** port 80 or port 443, depending on the type of connection, as described here:

- TCP port 80 is used for a standard, unencrypted connection.
- TCP port 443 is used for a secure, encrypted connection.

### Note

In some cases, the server may be reached using an alternate port, such as `http-alt 8080` or `http-alt 8008`.

Using a secure connection is common today, as most websites use encryption to protect data transactions. When accessing a secure site, the URL will be identified using the **HTTP Secure (HTTPS)** preface. HTTPS uses **Transport Layer Security (TLS)** to secure all transactions between the client and the server. Even if someone were able to obtain the data stream, they would not be able to read the contents without the appropriate key to decrypt the data.

Once the target is located, the client will make a request to the server. Each party has a specific role in a web transaction, as discussed next.

## Comparing client and server roles

HTTP is a client-server model, whereby the client makes a request to the server, and the server responds to the client, as shown in the following illustration:

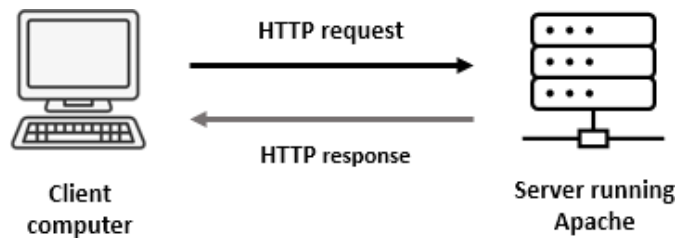


Figure 15.2 – HTTP request and response

A **client** is a host that initiates each session using a browser to interface with a server and retrieve objects.

A **server** is an always-on host with a fixed **Internet Protocol (IP)** address that uses dedicated web server software. Although there are several web server applications, the top three are listed as follows:

- **Apache** is one of the most popular open source web server applications in use today. Apache has a full library of modules that enable a rich set of features that can power even the largest sites.
- **nginx** (or engine X) is the second most popular open source web server application and is robust, scalable, and easy to configure.
- **Cloudflare** is a popular option used to host websites and help prevent malicious attacks such as **distributed denial-of-service (DDoS)** attacks.

Whenever requesting a web page, the version will be either HTTP version 1.0, 1.1, or 2.0, as discussed next.

## Understanding HTTP versions

HTTP has three versions in use today. Each has an associated **Request for Comments (RFC)**, as follows:

- **HTTP 1.0**, outlined in *RFC 1945*, was written in 1996. This was the original version, developed when we were just beginning to see the benefits the internet could potentially offer. This stripped-down version offered limited functionality, only had 16 response codes, and didn't have a secure method to authenticate users when interacting with the site.
- **HTTP 1.1**, outlined in *RFC 2068*, was written in 1996, shortly after version 1.0. This version had numerous improvements, including authentication, improved performance, the ability to reuse sessions, and expanded status codes, along with granular error reporting.
- **HTTP 2.0**, outlined in *RFC 7540*, was written in 2015 and evolved from an experimental protocol developed by Google called **SPeeDY (SPDY)**. Version 2 reduces latency and improves efficiency by compressing the HTTP headers to reduce overhead, multiplexes requests and responses, and proactively pushes content to the client.

All three versions are in use; however, you will most likely see versions 1.1 and 2.0 for most of today's web traffic, as these versions offer improved performance.

When doing an analysis, you will see a method listed in the HTTP header that will indicate the type of action that is requested.

## Recognizing HTTP methods

HTTP is primarily used to obtain objects and interact with a web server using various methods to exchange information. Some of the most common methods are described in the following table:

Method	Description
GET	Used by the client to obtain information from the server
POST	Used to submit information and files, such as form data
PUT	Used to submit information and files; however, replaces any existing data
DELETE	Deletes any existing data

Table 15.1 – Common HTTP methods

While there are several HTTP request methods, the two most commonly used are GET and POST.

In the next section, we'll compare connection methods and cover how cookies are used to keep track of transactions.

## Keeping track of the connection

Every version of HTTP has evolved in methods to transport and process data, with techniques such as using persistent connections with a pipelining goal to optimize the connection. In addition, because HTTP is a stateless protocol, cookies are used to maintain client information about the connection, such as shopping cart elements and pages the client has visited.

In this segment, we'll compare the different methods of making and maintaining a connection, along with how cookies are used to preserve state information. Let's start by comparing connection types.

## Evaluating connection types

When HTTP version 1.0 was developed, it had minimal functionality and used a non-persistent connection. With the introduction of version 1.1, several enhancements were added. One of the improvements was the ability to keep the data moving by using a persistent connection.

Let's start by understanding the mechanics of a non-persistent connection.

### Using a non-persistent connection

A non-persistent connection is one of the original ways to retrieve content from a web page. This method works in the following manner:

1. The client requests content from a web server.
2. The server returns the content to a client.
3. The connection is closed.

If the client wants to obtain any more content from the web server, a new connection must be made, resulting in one round trip per request and response. As you can imagine, all of the round trips and connections can add to latency.

There are two approaches in a non-persistent connection, as outlined here:

- A non-parallel (or serial) connection uses one connection at a time.
- A parallel connection uses multiple concurrent connections.

If possible, a parallel connection is used, as this will improve the overall efficiency of obtaining objects.

When using HTTP version 1.0, the default is to use a non-persistent connection. HTTP version 1.1 improved the method to connect by offering a persistent connection, as outlined next.

### Offering a persistent connection

To optimize a connection, HTTP version 1.1 introduced the idea of using a persistent connection to keep a session alive until all objects are retrieved. This type of connection is also referred to as an HTTP keep-alive, as it prevents having to reestablish a new connection every time a request is made.

**Note**

Although the idea of a keep-alive was made official in version 1.1, version 1.0 adopted an unofficial method to keep the session alive as well. That is why you might see HTTP 1.0 traffic using a keep-alive during a transaction.

When used, the HTTP header can set a timeout and define the maximum number of requests the client can make.

Keep-alive packets are sent between the client and the server to keep the session active and to verify that both sides are still responding. A keep-alive packet doesn't have any data; it has the **acknowledgment (ACK)** flag set and the sequence number is set to one less than the *current* sequence number.

HTTP version 2.0 improves this option and permits multiple concurrent transactions to be combined in a single connection.

HTTP 1.1 made persistent connections the default mode. Next, let's see how a persistent connection can benefit from using a concept called pipelining.

## Pipelining the data transfer

To optimize a data transfer using a persistent connection, HTTP can employ pipelining, which maintains the connection to the server until all objects are obtained. Pipelining improves the efficiency of a data transaction, as the client can continue to request objects immediately without waiting for the server to send previous requests.

Instead of closing each connection after a request-response transaction, the server will keep the connection open and wait for further requests. A persistent connection using pipelining is the default connection method in HTTP 1.1. This type of connection requires a bit more overhead in terms of setting up the connection. In addition, because the server is waiting for multiple requests, it might be idle for short segments of time. However, this type of connection is preferred as it improves overall efficiency when obtaining objects from a web server.

Because HTTP is a stateless protocol, cookies are used to keep track of the details of each transaction.

## Maintaining state with cookies

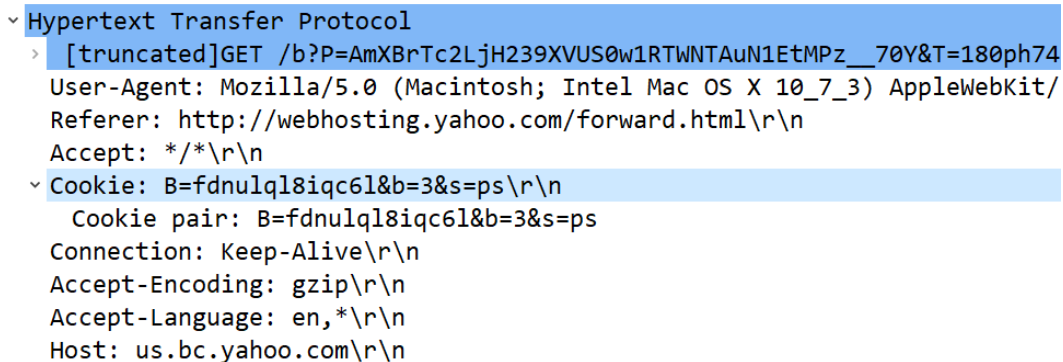
When using HTTP, neither the server nor the client natively maintains information on the state of the transaction. To overcome this, HTTP uses the concept of a cookie.

A cookie is a string of characters created by the server that can be placed on a user's system and then managed by the user's browser to interact with the server. Once the user accepts a cookie, the information contained in the cookie can be used for the following reasons:

- **Authentication:** After a user provides login information, the data can be used as a form of **single sign-on (SSO)** to interact with the website.
- **Personalize the user experience (UX):** A cookie can collect data on shopping patterns. This will then improve the shopping experience by retaining previously accessed objects and periodically presenting the information to the client.

For a closer look at a cookie being used in an HTTP conversation, go to <http://tcpreplay.appneta.com/wiki/captures.html#bigflows-pcap>. Once there, download `bigFlows.pcap` so that you can follow along.

Go to **Frame 912** and then expand the HTTP header. Once there, you will see the cookie and cookie pair, as shown in the following screenshot:



```

Hypertext Transfer Protocol
> [truncated]GET /b?P=AmXBrTc2LjH239XVUS0w1RTWNTAuN1EtMPz__70Y&T=180ph74
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/
Referer: http://webhosting.yahoo.com/forward.html\r\n
Accept: */*\r\n
Cookie: B=fdnulql8iqc6l&b=3&s=ps\r\n
    Cookie pair: B=fdnulql8iqc6l&b=3&s=ps
Connection: Keep-Alive\r\n
Accept-Encoding: gzip\r\n
Accept-Language: en,*\r\n
Host: us.bc.yahoo.com\r\n
  
```

Figure 15.3 – Viewing cookies in an HTTP header

In most cases, cookies are designed to be helpful; however, they can pose a privacy risk as they can gather marketing statistics and personal information. That is why, in most cases, a user will have the ability to *opt out* of allowing cookies on their system.

In some cases, a cookie is sent to the server to be maintained and retrieved for future visits by the client. However, if a cookie is not sent to the server, it is said to be non-persistent, and one of the following scenarios will take place:

- The cookie will expire.
- The cookie will be removed when the user shuts down the browser.
- The user will remove (or delete) the cookie.



During an HTTP transaction, there are a series of requests and responses between the client and the server. In the next section, we'll examine the general format of request and response messages.

## Comparing request and response messages

When interacting with a web page, a client will request objects from a server. In most cases, a standard transaction involves a request from the client and then a response from the server. Because the client and the server both convey a different message, the headers are slightly different. In this segment, we'll take a high-level view of the HTTP header and fields for both the client and the server.

Let's start by examining the elements of an HTTP request.

### Viewing an HTTP request

When viewing a client request, you will most likely see a request line followed by header lines, as shown in the following diagram:

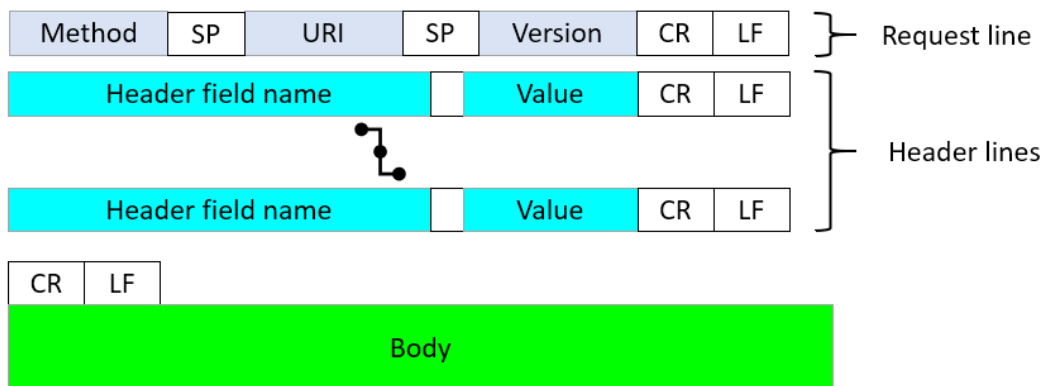


Figure 15.4 – HTTP request format

Each request message will indicate the method (such as `GET` or `POST`) and the URL, along with additional parameters. It's also common to see the following:

- The `\r` symbol, which is used to indicate a carriage return
- The `\n` symbol, which is used to indicate a newline

Both symbols in the header provide formatting guidance that represents the end of the content for that particular line. To see an example, return to `bigFlows.pcap` and select **Frame 183**. Then, expand the HTTP header, as shown here:

```

v Hypertext Transfer Protocol
  > GET /CSIS/CSISISAPI.dll/?request?b2bc13b2
    User-Agent: CSISHttpRequest\r\n
    Host: 172.16.139.250:5440\r\n
    Cache-Control: no-cache\r\n
    \r\n

```

Figure 15.5 – HTTP GET request

An HTTP request is used to retrieve objects. Most web servers provide rich, interactive content. As a result, the headers must indicate the types of content that the client can support. Within an HTTP request, you will see a reference to the **Multipurpose Internet Mail Extensions (MIME)** header. Let's explore this next.

## Viewing the MIME header

MIME is designed to present data in a variety of formats that include text, images, audio, applications, and video. Originally developed to support email, MIME headers became a part of the evolution of HTTP.

When the server receives the HTTP request, the **request line** will list what the client wants, along with the HTTP version that the browser supports. After that, the MIME header contains fields that relate to the type of content that the client can support. An example is shown here:

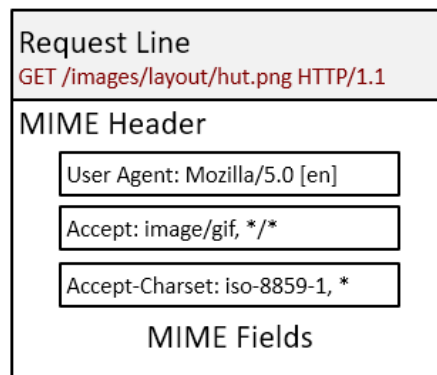


Figure 15.6 – HTTP request with a MIME header

The MIME header holds values such as the browser version, along with character encoding and objects that the client will accept.

Once the server receives the request, it will send the response to the client. The header format is much the same as the HTTP request; however, this contains details that are specific to the server response.

## Responding to the client

The HTTP response from the server begins with the status line, indicating the version, status code, and reason. After the status line, the structure is similar to the HTTP request, as we find header lines, as shown in the following diagram:

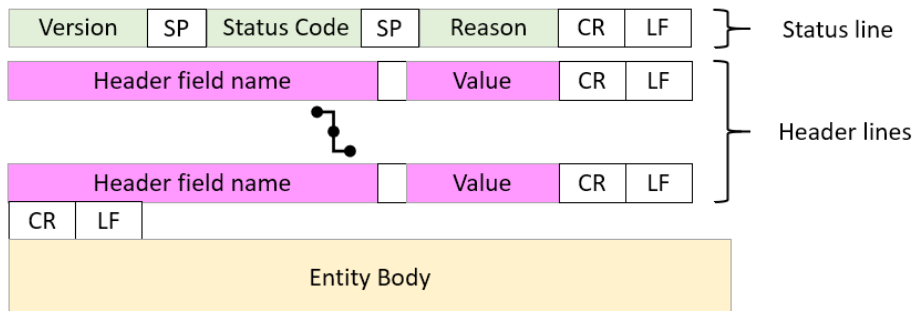


Figure 15.7 – HTTP response format

Following the status code and header lines, you will see the message body, which can contain client-requested items, along with details of the transaction.

One of the main elements of an HTTP response is the status of the message. HTTP indicates the status by using a specific code, as outlined next.

## Understanding status codes

A status code is shown in the first line of a web server to the client response message. Status codes are grouped by category, as follows:

- **1xx Informational:** Provides general notes about the transaction, such as the request has been received and/or the process is moving forward
- **2xx Success:** Indicates that the server was able to positively receive and process the request
- **3xx Redirection:** Indicates the client must be redirected to another resource to complete the request
- **4xx Client Error:** Means that the request contains invalid syntax or could not be satisfied for some reason
- **5xx Server Error:** Occurs when there is a failure on the server's part to complete an action or request

While many status codes are available, some of the most common codes include the following:

- **200 OK:** This is the most common status code and represents that the server is able to successfully return the requested object.
- **301 Moved Permanently:** This indicates that the requested object is no longer available at the URL and has moved permanently to a new location.
- **400 Bad Request:** This status means the HTTP client request was invalid and was unable to be processed by the server.
- **404 Not Found:** This status is returned if the requested object is not found on the server.

After the status line, you will see various elements within the MIME header related to the response, as outlined next.

## Describing the MIME header

Whenever you retrieve data from a web page, your browser will properly format the request. In turn, the server will respond to the request and deliver the data. The browser will then present the results to the client.

When a server returns objects to a client, there are various elements within the MIME header, as shown here:

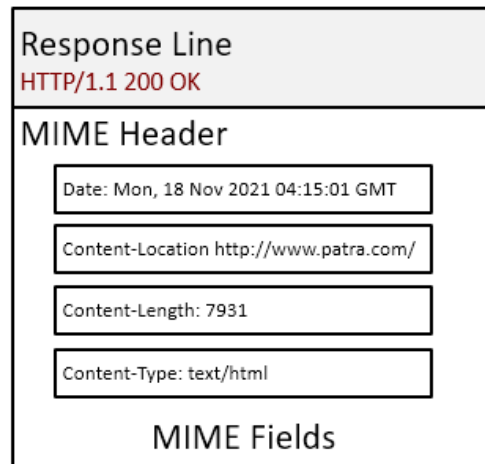


Figure 15.8 – MIME fields in an HTTP response

The field values define details such as the type of content, where it originated, and length.

In the next section, let's explore the process from making a request, to displaying an object, to closing a connection.

## Following an HTTP stream

To get a solid understanding of what happens when requesting and receiving a web page, we'll step through the process by following an HTTP stream. We'll then further break down each of the elements when examining an HTTP transaction.

### Note

Keep in mind that each HTTP session is different. This example will provide a sampling of what you can expect when viewing an HTTP conversation. In a true analysis exercise, you will most likely have to research the meaning of some of the various field values.

For this example, we'll use `HTTP.pcap`, as it is a complete conversation. To obtain a copy, go to <https://www.cloudshark.org/captures/0012f52602a3>, then download the file and open it in Wireshark. Once open, expand **Frame 1** under the TCP header, where you will see the following:

```
[Conversation completeness: Complete, WITH_DATA (31)]
```

This is a small capture with only 40 packets, so it isn't difficult to see all elements of the complete conversation. However, one way to view the activity between hosts is by running a flow graph. To view the entire conversation, go to the **Statistics** menu choice and then select **Flow Graph**. Wireshark will then run the graph, as shown in the following screenshot:

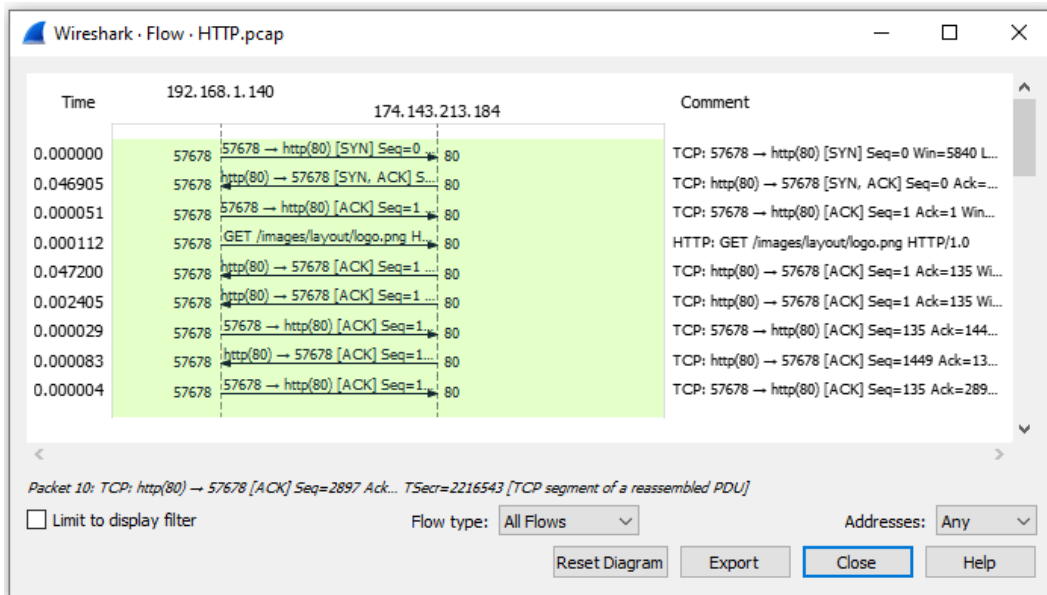


Figure 15.9 – Viewing the HTTP flow

Once in the **Flow Graph** setting, you can select **Limit to display filter**, as shown on the left-hand side of the graph. However, because this capture only contains one conversation, this isn't necessary.

After viewing the flow graph, exit the window so that we can focus on the capture. To see the dialog between the client and the server, place your cursor on **Frame 4**, right-click, and select **Follow | TCP Stream**, as shown in the following screenshot:

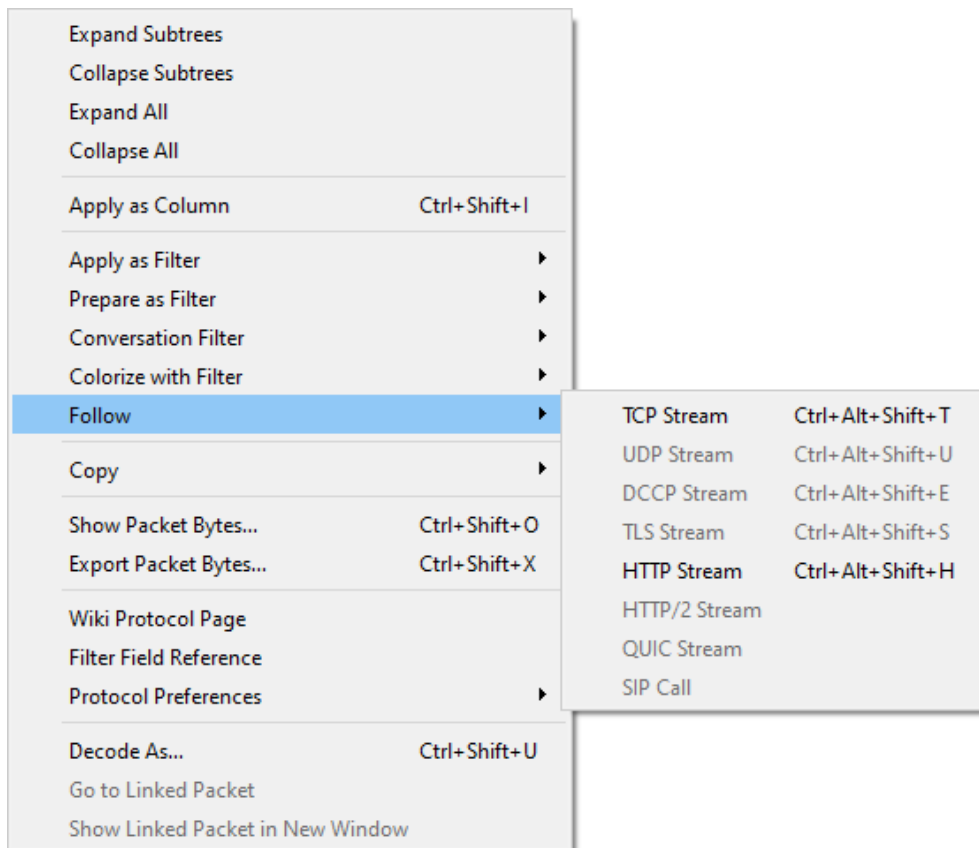


Figure 15.10 – Following the TCP stream

Once selected, Wireshark will present a window with the following conversation:

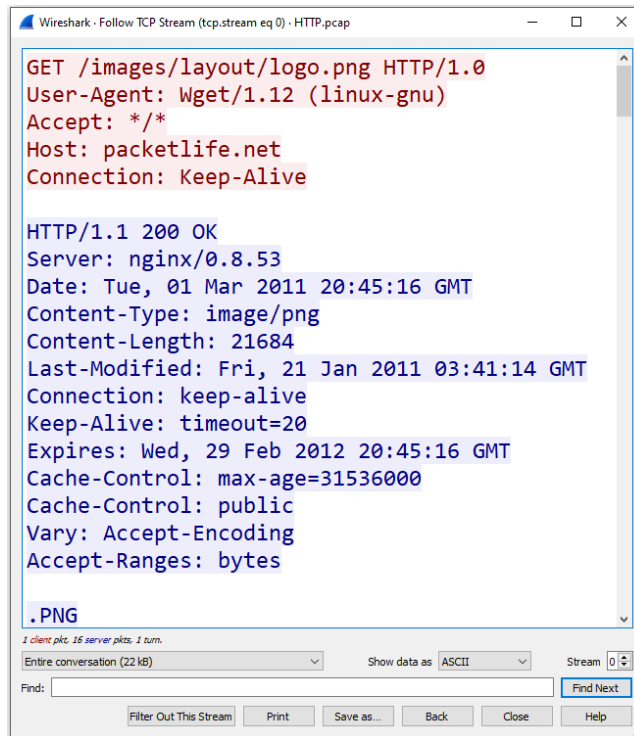


Figure 15.11 – Viewing Follow TCP Stream

Within the window, you will see the following:

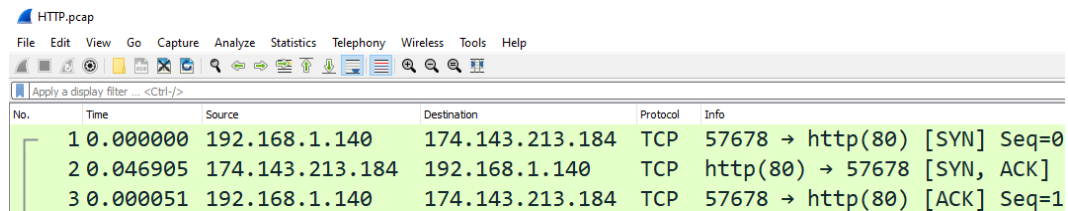
- The client request at the top of the window, as shown in *Figure 15.11* using red font
- The server response, which starts in the middle of the window, as shown in *Figure 15.11* using blue font

Now that we have seen the entire conversation, let's go through the process. The first step is for the client to establish contact with the server by using a three-way handshake.



## Beginning the conversation

Before any data is exchanged, the process begins with a three-way handshake, which we see in **Frames 1, 2, and 3** of `HTTP.pcap`, as shown in the following screenshot:



No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.140	174.143.213.184	TCP	57678 → http(80) [SYN] Seq=0
2	0.046905	174.143.213.184	192.168.1.140	TCP	http(80) → 57678 [SYN, ACK]
3	0.000051	192.168.1.140	174.143.213.184	TCP	57678 → http(80) [ACK] Seq=1

Figure 15.12 – Viewing a three-way handshake

A three-way handshake is an exchange of **synchronization (SYN)** and ACK packets between the client (192.168.1.140) and the server (174.143.213.184), as described in more detail here:

- The client begins the process by sending a SYN packet to the server.
- The server responds by sending a SYN-ACK to the client.
- The client responds by sending an ACK to the server.

No data is exchanged as the handshake simply establishes the connection. Once established, the next step is for the client to make a request to the server.

## Requesting data

**Frame 4** begins the conversation as the client makes a request to the server for an image. The HTTP header is shown in the following screenshot:

```

Hypertext Transfer Protocol
GET /images/layout/logo.png HTTP/1.0\r\n
  [Expert Info (Chat/Sequence): GET /images/layout/logo.png HTTP/1.0\r\n]
    Request Method: GET
    Request URI: /images/layout/logo.png
    Request Version: HTTP/1.0
  User-Agent: Wget/1.12 (linux-gnu)\r\n
  Accept: */*\r\n
  Host: packetlife.net\r\n
  Connection: Keep-Alive\r\n
\r\n
[Full request URI: http://packetlife.net/images/layout/logo.png]
[HTTP request 1/1]
[Response in frame: 36]

```

Figure 15.13 – Client-side HTTP request

When examining the client request, we see several values in the HTTP header. The first line is the GET method requesting data from the server. Here are the parameters of this request:

- Request Method: GET, which is used by the client to obtain information from the server.
- Request URI: The client is requesting the `logo.png` image using the `images/layout/logo.png` path.
- Request Version: The client is using HTTP/1.0.

Following the GET method, you will see these values:

- User-Agent: The user agent listed is `Wget/1.12: (linux-gnu)`, which is a simple non-interactive browser used to obtain files using methods such as HTTP and FTP.
- Accept: Lists `*/*`, which is a wildcard value that means the client will accept any MIME-type object.
- Host: The requested host is `packetlife.net`.
- Connection: Keep-Alive is listed as the connection method, which tells the server to continue to send data without having to open new connection requests.

Following the field values, we see three Wireshark-generated details, as outlined next:

- [Full request URI: `http://packetlife.net/images/layout/logo.png`]: This is a hyperlink to the exact location of the requested object.
- [HTTP request 1/1]: This indicates that this request is the first out of one (1) HTTP request.
- [Response in frame: 36]: This is a hyperlink to the frame containing the HTTP response.

After the client request is sent to the server, data is then sent to the client. From **Frame 5** to **Frame 35**, you will see a series of ACK packets, from both the client and the server as they acknowledge the data is being transferred and received.

Once complete, the server will respond to the client and indicate that all data has been sent.

## Responding to the client

In **Frame 36**, we see the server returning a `Status Code: 200 OK` response, meaning the data has successfully been delivered.

Unlike the request packet, we see more information contained in the HTTP response, as shown in the following screenshot:

```

Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
      Server: nginx/0.8.53\r\n
      Date: Tue, 01 Mar 2011 20:45:16 GMT\r\n
      Content-Type: image/png\r\n
    Content-Length: 21684\r\n
      [Content length: 21684]
      Last-Modified: Fri, 21 Jan 2011 03:41:14 GMT\r\n
      Connection: keep-alive\r\n
      Keep-Alive: timeout=20\r\n
      Expires: Wed, 29 Feb 2012 20:45:16 GMT\r\n
      Cache-Control: max-age=31536000\r\n
      Cache-Control: public\r\n
      Vary: Accept-Encoding\r\n
      Accept-Ranges: bytes\r\n
      \r\n
      [HTTP response 1/1]
      [Time since request: 0.152882000 seconds]
      [Request in frame: 4]
      [Request URI: http://packetlife.net/images/layout/logo.png]
      File Data: 21684 bytes
  
```

Figure 15.14 – Server-side HTTP response

When examining the server response, we see several values in the HTTP header. The first line of the data transfer is a Wireshark-generated value that indicates the status of the response, as shown here:

```
[Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
```

This line defines that `HTTP/1.1 200 (PNG)` falls under the Chat severity in the **Expert Information** console. To view this, select the cyan Expert Info icon in the lower left-hand corner of the Wireshark interface. Once open, expand the second Chat entry, which will display the following information:

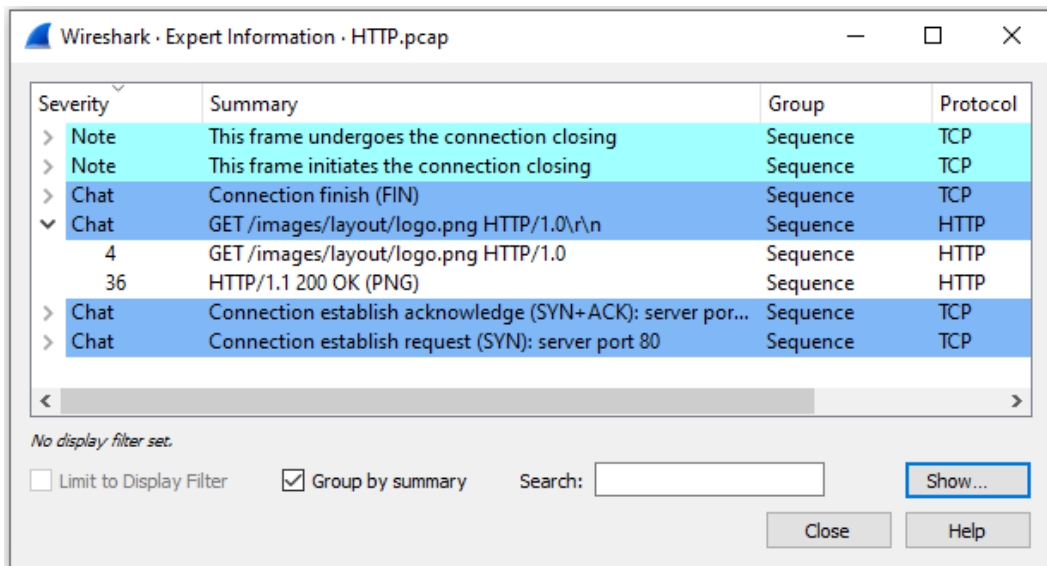


Figure 15.15 – Viewing Expert Information entries

After the Wireshark-generated values, you will see the following:

- **Response Version:** This value shows the server is using HTTP version 1.1.
- **Status Code:** In this case, the data transfer was successful, so the server returned a Status Code 200 response.
- **[Status Code Description: OK]:** This line is a Wireshark-generated value indicating that there were no issues with the response.
- **Response Phrase:** This field value is OK, which indicates a successful data transfer.

After the details of the transaction, you will see the following values:

- **Server:** This field lists the web server as `nginx/0.8.53`. nginx is a popular open source web server that is an alternative to Apache.
- **Date:** This field marks the date and time the object was accessed and is used to monitor the age of a resource for caching calculations. **Frame 35** lists `Tue, 01 Mar 2011 20:45:16 GMT` as the date.
- **Content-Type:** This field provides a reference as to what type of object is being returned to the client. **Frame 35** lists the Content-Type as `image/png`, which is an image in the **Portable Graphics Format (PNG)** format.

- **Content-Length:** This field lists the length of the image, which is 21684 bytes. Below the field value, you will see a [Content length: 21684] Wireshark-generated value. This value will match the Portable Network Graphics header, found at the bottom of the HTTP header. If you place your cursor on the header, you will see the value reflected in the status bar, as shown in the following screenshot:

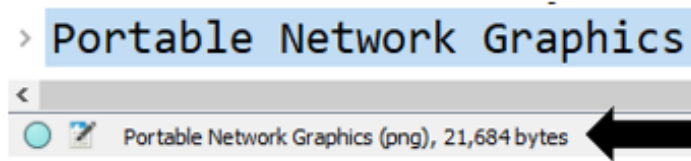


Figure 15.16 – Content length of the image

- **Last-Modified:** This field value will indicate when the page was last modified, which is listed as `Fri, 21 Jan 2011 03:41:14 GMT`.
- **Connection:** This value is listed as a keep-alive (or persistent) connection.
- **Keep-Alive:** This field value lists `timeout=20`, which is a value (in seconds) set by the server indicating how long to keep the conversation going before closing the connection.

On a network, the concept of cache is important as, in most cases, the requesting host wants to receive the freshest copy possible. Cache values are used to identify the age of the page.

## Controlling the cache

Most objects on the network typically have a timeout value. The next few values provide a method to monitor the age of a page. Entries include the following:

- **Expires:** This value specifies when an object will expire. In **Frame 35**, the expiration date is listed as `Wed, 29 Feb 2012 20:45:16 GMT`.
- **Cache-Control:** This value is listed as `max-age=31536000`.

### Note

On assets that rarely change, it's customary to set the `Cache-Control` value to `max-age=31536000`, which represents the number of seconds in 1 year.

- **Cache-Control:** This field value is listed as `public`, which means the values can be stored in a shared cache. Other values can include `private` or `no-cache`.
- **Vary:** This field represents the type of content supported by the caching server. In this case, the value is `Accept-Encoding`. If this indicator is not present, the formatting might not be stored appropriately, and when the client retrieves the page, it might render the page in an unreadable format.
- **Accept-Ranges:** This field value indicates which *unit* the server will use when responding to the client's request for data. The field value is listed as `bytes`, which is common.

At the end of the HTTP field values, we see the `Portable Graphics Format` header, which indicates that the object is a PNG file. Next, let's investigate the image.

## Exporting the object

If there are unencrypted objects in the capture, you can export them in Wireshark. To see what's in the HTTP .pcap file, go to the **File** menu choice and then **Export Objects | HTTP...**, which will bring up the following window:

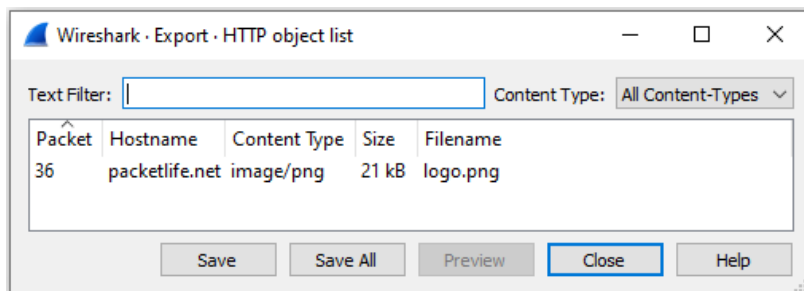


Figure 15.17 – Viewing Export HTTP object list

If you would like to view the contents, click on the image and select **Save**, which will bring up a dialog box giving you a choice as to where you would like to save the file. Once saved, you can open the image, as shown here:



Figure 15.18 – The extracted image

Once the data is transferred and the client is no longer requesting objects, the next step is to end the conversation.

## Ending the conversation

At the end of the transaction, the client and server exchange a series of **finish (FIN)** and ACK packets, as shown here:

37 0.000005	192.168.1.140	174.143.213.184	TCP	57678 → http(80)	[ACK] Seq=135
38 0.000625	192.168.1.140	174.143.213.184	TCP	57678 → http(80)	[FIN, ACK] Seq
39 0.046230	174.143.213.184	192.168.1.140	TCP	http(80) → 57678	[FIN, ACK] Seq
40 0.000019	192.168.1.140	174.143.213.184	TCP	57678 → http(80)	[ACK] Seq=136

Figure 15.19 – Exchange of FIN-ACK packets

While this was a simple example of a data transaction, it provides a solid way to understand the mechanics of a client interacting with a web server.

## Summary

HTTP is a rich protocol, and during a transaction, data will travel across several networks, along with encountering different clients and servers. As a result, there are numerous rules and variables. For this discussion, we focused on the key elements of an HTTP request-response session between a client and server.

We began by stepping through the key elements of a web page, and the role of the client and the server when retrieving data and objects. We reviewed the different HTTP versions, along with briefly touching on the available HTTP methods. We then moved on to learn about the different types of connections and how cookies help maintain state during a transaction. Finally, we summarized with a simple example of a complete HTTP conversation.

In the next chapter, we'll review **Address Resolution Protocol (ARP)** and begin with an overview of the role and purpose of this essential protocol. So that you understand how ARP works, we will cover an ARP transaction and take a closer look at the header and field values. We will see the importance of a gratuitous ARP and briefly mention some other types of ARP traffic that you might encounter. Finally, we will look at ARP attacks and how to identify and defend against these types of threats.

## Questions

Now, it's time to check your knowledge. Select the best response to the following questions and then check your answers, which can be found in the *Assessment* appendix:

1. The two main HTTP request methods are GET and \_\_\_\_\_.
  - A. PUT
  - B. DELETE
  - C. POST
  - D. STATE
2. \_\_\_\_\_ is one of the most popular open source web server applications in use today, as it has a full library of modules and can power even the largest sites.
  - A. Cloudflare
  - B. Microsoft IIS
  - C. LiteSpeed
  - D. Apache
3. HTTP version \_\_\_\_\_ reduces latency and improves efficiency by compressing headers to reduce overhead, multiplexes requests and responses, and proactively pushes content to the client.
  - A. 1.0
  - B. 1.1
  - C. 1.2
  - D. 2.0
4. When transferring data from a web server, a persistent connection keeps the session alive until all objects are retrieved. This type of connection is also referred to as an HTTP \_\_\_\_\_.
  - A. Flare state
  - B. Parallel connection
  - C. Keep-alive
  - D. Cookie state



5. In the HTTP header, you see the value 3600 listed in the Cache-Control field. This means the content can live in cache for \_\_\_\_\_.
  - A. 1 day
  - B. 1 hour
  - C. 1 year
  - D. 1 month
6. Originally developed to support email, \_\_\_\_\_ is designed to present data in a variety of formats that include text, images, audio, applications, and video.
  - A. LiteSpeed
  - B. MIME
  - C. Cookie Art
  - D. nginx
7. Most objects on the network typically have a timeout value. \_\_\_\_\_ values provide a method to monitor the age of a page.
  - A. Cache
  - B. MIME
  - C. Cookie
  - D. State

## Further reading

Please refer to the following links for more information:

- Visit [https://www.w3schools.com/tags/ref\\_httpmethods.asp](https://www.w3schools.com/tags/ref_httpmethods.asp) to learn more about HTTP request methods.
- Learn more about nginx by visiting <https://nginx.org/en/>.
- Cache control for objects that rarely change will default at a value of 1 year. See why this is the default by going to <https://ashton.codes/set-cache-control-max-age-1-year/>.
- To discover the most popular web server software in use today, visit <https://digitalintheround.com/what-is-the-most-popular-web-server/>.

- Learn about safe methods of requesting data from a server by visiting <https://datatracker.ietf.org/doc/html/rfc7231#section-4.2>.
- For a detailed list of HTTP methods, visit [https://www.tutorialspoint.com/http/http\\_methods.htm](https://www.tutorialspoint.com/http/http_methods.htm).
- Visit <https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml> to see a complete list of HTTP response codes.
- Visit <https://svn.apache.org/repos/asf/trafficserver/site/branches/2.0.x/docs/sdk/HTTPHeaders.html> for a discussion on HTTP headers.
- Visit <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control> to learn more about cache control.



# 16

# Understanding ARP

When data travels across networks, the packets use a logical or **Internet Protocol (IP)** address to identify the destination host. However, on a **local area network (LAN)**, the packets use a physical or **media access control (MAC)** address to deliver data to a host. When a packet is transported from a website across the internet to a host on a LAN, the switch only has an IP address to identify the host. How does the device locate the destination host? That is the responsibility of the **Address Resolution Protocol (ARP)**, which resolves an IP address to a MAC address so that the data gets delivered to the correct host.

In this chapter, we'll learn how ARP works, and why it is an important protocol in ensuring the timely delivery of data. We'll then take a closer look at ARP headers and fields in Wireshark. We'll also examine the different types of ARP that you may encounter while doing analysis, including gratuitous, reverse, inverse, and proxy. Finally, so that you are aware that ARP may be used in a malicious way, we'll discuss ARP attacks and possible ways to defend against these types of threats.

This chapter will cover the following:

- Understanding the role and purpose of ARP
- Exploring ARP headers and fields
- Examining the different types of ARP
- Comparing ARP attacks and defense methods

## Understanding the role and purpose of ARP

ARP is one of the three main network layer protocols (ARP, IPv4, and **Internet Control Message Protocol (ICMP)**), all of which are essential in delivering data.

In the following diagram, we see that ARP is actually in between Layer 3 and Layer 2 of the **Open Systems Interconnection (OSI)** model, as ARP resolves an IP address (network layer) to a MAC address (data link layer):

### OSI Model

Layer	Name	Role	Protocols	PDU	Address
7	Application	Initiates contact with the network	HTTP, FTP, DNS	Data	
6	Presentation	Formats data, optional compression and encryption		Data	
5	Session	Initiates, maintains and tears down a session		Data	
4	Transport	Transports data	TCP, UDP	Segment	Port
3	Network	Addressing, routing	IP, ICMP	Packet	IP
2	Data Link	Frame formation	Ethernet II	Frame	MAC
1	Physical	Data is transmitted on the media		Bits	

Figure 16.1 – The OSI model network layer protocols

#### Note

Although ARP resides in between Layer 2 and Layer 3 of the OSI model, many consider ARP as a Layer 3 protocol.

In this section, we'll see how ARP resolves a MAC address. We'll then learn how the ARP cache helps provide a speedier response. In addition, we'll discover how an IPv6 network uses the **Neighbor Discovery Protocol (NDP)** to resolve an IPv6 address to a MAC address.

On a LAN using IPv4, the frame header must use a MAC address to identify the receiving host. To obtain the correct MAC address, the sending device will issue an ARP broadcast on the network. In the next section, let's discuss why this is important.

## Resolving MAC addresses

When data travels through different networks, packets use an IP (or logical) address. However, to deliver the data to its final destination, a MAC (or physical) address is needed to place in the frame header. The device will first check its local ARP cache. If there is no entry, the device issues an ARP request in the form of a broadcast, and will wait for a reply.

### Note

A broadcast message is sent from one host to all devices on a network. While every host will receive the frame, only one will respond.

As shown in the following diagram, *Host A* needs the MAC address for the gateway, which is the router interface for the LAN:

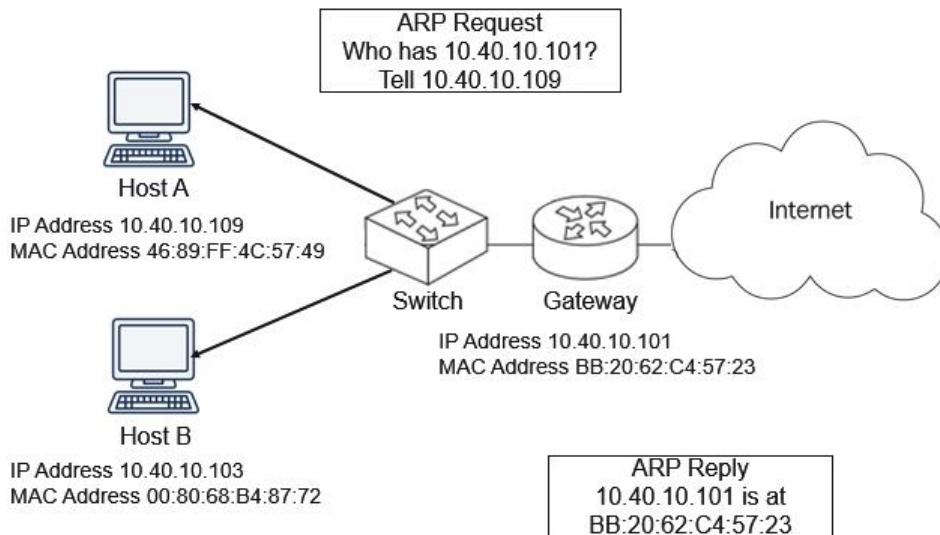


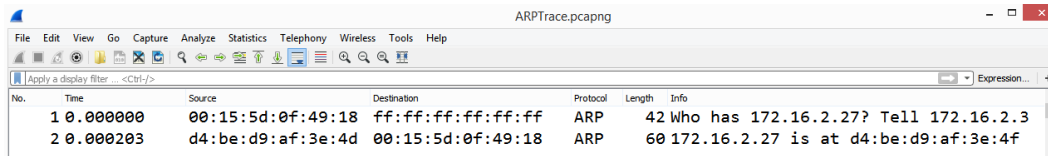
Figure 16.2 – ARP broadcast on a network

To obtain the MAC address of the gateway, *Host A* issues an ARP request. The ARP request asks the question, "Who has the IP address 10.40.10.101? Tell 10.40.10.109," and waits for a reply. The gateway then sends an ARP reply to let the host know that 10.40.10.101 is at MAC address BB:20:62:C4:57:23.

Now that we understand the basics of an ARP request and reply, let's step through what we will see while viewing the process in Wireshark.

## Viewing an ARP request and reply

To see an example of an ARP request and reply so that you can follow along, go to <https://crnetpackets.files.wordpress.com/2015/08/arptrace.zip>, download the file, extract it, and open it in Wireshark, as shown here:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	00:15:5d:0f:49:18	ff:ff:ff:ff:ff:ff	ARP	42	Who has 172.16.2.27? Tell 172.16.2.3
2	0.000203	d4:be:d9:af:3e:4d	00:15:5d:0f:49:18	ARP	60	172.16.2.27 is at d4:be:d9:af:3e:4f

Figure 16.3 – ARP request/reply

In the ARP trace file, the first two packets are the ARP request/reply.

In **Frame 1**, the device sends an ARP request. The source and destination addresses are as follows:

- The source MAC address is 00:15:5d:0f:49:18, which is the MAC address of the device requesting the resolution.
- The destination MAC address is ff:ff:ff:ff:ff:ff, which is a broadcast address.

In **Frame 2**, the device identifies itself with an ARP reply. The source and destination addresses are as follows:

- The source MAC address is d4:be:d9:af:3e:4d, which is the device with the IP address 172.16.2.27.
- The destination MAC address is 00:15:5d:0f:49:18, which is the MAC address of the device requesting the resolution.

Keep in mind, when doing a capture in Wireshark, it is normal to see several ARP broadcasts before seeing an ARP reply.

We now know that ARP resolves an IP address to a MAC address. This is so the device has a MAC address that can be placed in the frame header in order for the data to be delivered on a LAN. So that the device is able to quickly retrieve the MAC address of a device on the network, it holds the IP address to MAC address pairings in a temporary holding area called the **ARP cache**. The next section explains an ARP cache, how it's used, and how long the table entries remain.

## Investigating an ARP cache

Network devices, such as routers, switches, and PCs, hold a form of an ARP cache table, which is a storage area to store IP to MAC address pairings. To see your own ARP cache on a Windows machine, open a Command Prompt and enter `arp -a` to see the entries in the ARP table, as shown here:

```
C:\WINDOWS\system32>arp -a

Interface: 10.0.0.148 --- 0x3
    Internet Address      Physical Address      Type
    10.0.0.1              5c-e3-0e-d9-e8-57    dynamic
    10.0.0.59             f0-79-60-33-6d-06    dynamic
    10.0.0.255            ff-ff-ff-ff-ff-ff    static
    224.0.0.22            01-00-5e-00-00-16    static
    224.0.0.251           01-00-5e-00-00-fb    static
    224.0.0.252           01-00-5e-00-00-fc    static
    239.255.255.250       01-00-5e-7f-ff-fa    static
    255.255.255.255       ff-ff-ff-ff-ff-ff    static

Interface: 192.168.124.1 --- 0xf
    Internet Address      Physical Address      Type
    192.168.124.254       00-50-56-e8-da-39    dynamic
    192.168.124.255       ff-ff-ff-ff-ff-ff    static
    224.0.0.22            01-00-5e-00-00-16    static
    224.0.0.251           01-00-5e-00-00-fb    static
    224.0.0.252           01-00-5e-00-00-fc    static
    226.178.217.5         01-00-5e-32-d9-05    static
    239.255.255.250       01-00-5e-7f-ff-fa    static
    255.255.255.255       ff-ff-ff-ff-ff-ff    static
```

Figure 16.4 – Using the `arp -a` command

In the previous figure, the ARP cache table lists the following:

- **Internet Address:** Displays the IP address
- **Physical Address:** Displays the corresponding MAC address
- **Type:** Displays whether the entry is either static (stays the same) or dynamic (periodically refreshed)

The dynamic ARP cache values will time out after a period of time. Once the timeout limit is reached, the entry will go away. If the **operating system (OS)** needs the MAC address and there is no entry in the ARP table, it will need to issue a new ARP request.

### Note

The ARP table timeout values are specific to the system. For example, a Cisco switch has a default timeout timer of 4 hours.



In a Windows OS, you can determine the timeout value by going to the **command-line interface (CLI)** and running the `netsh interface ipv4 show interface NNN` command.

**Note**

When using the command, replace NNN with the name of the interface you want to check.

As shown in the screenshot, we see the output of running `netsh interface ipv4 show interface Wi-Fi`, which provides information on that interface:

```
C:\WINDOWS\system32>netsh interface ipv4 show interface wi-fi
Interface Wi-Fi Parameters
-----
IfLuid           : wireless_0
IfIndex          : 3
State            : connected
Metric           : 25
Link MTU         : 1500 bytes
Reachable Time   : 26500 ms
Base Reachable Time : 30000 ms
Retransmission Interval : 1000 ms
DAD Transmits    : 3
Site Prefix Length : 64
Site Id          : 1
Forwarding       : disabled
Advertising      : disabled
Neighbor Discovery : enabled
Neighbor Unreachability Detection : enabled
Router Discovery  : dhcp
Managed Address Configuration : enabled
Other Stateful Configuration : enabled
Weak Host Sends   : disabled
Weak Host Receives : disabled
Use Automatic Metric : enabled
Ignore Default Routes : disabled
Advertised Router Lifetime : 1800 seconds
Advertise Default Route : disabled
Current Hop Limit : 0
Force ARPND Wake up patterns : disabled
Directed MAC Wake up patterns : disabled
ECN capability    : application
```

Figure 16.5 – Issuing the netsh show interface command

Within the output, you will see **Base Reachable Time** is **30000 ms** or 30 seconds, which is how long ARP can live in the cache before going away. After 30 seconds, if a MAC address is needed for a specific IP address, the system must send an ARP request out on the network.

We now understand how ARP works in an IPv4 network, but what happens in an IPv6 network? The following section outlines how NDP takes the place of ARP in an IPv6 network.

## Replacing ARP with NDP in IPv6

ARP is essential in an IPv4 network, however, IPv6 doesn't use ARP. Instead, ARP is replaced with NDP, which resolves an IP address to a MAC address.

To see an example of NDP, go to <http://packetlife.net/captures/protocol/icmpv6/> and open the `IPv6_NDP.pcap` file in Wireshark. As shown in the following screenshot, the first packet in the trace file is a **Neighbor Solicitation (NS)**, followed by a neighbor advertisement:

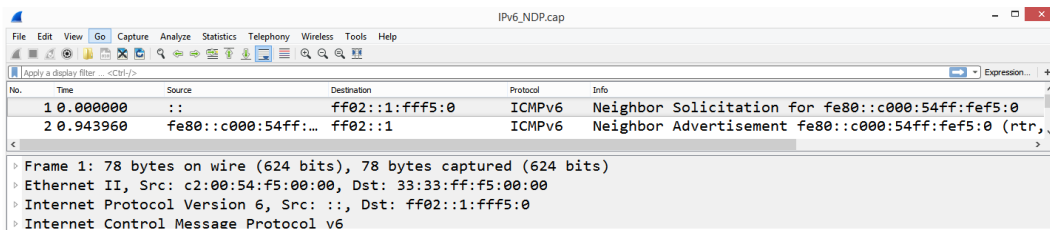


Figure 16.6 – Example of NDP

An NS has the same purpose as an ARP broadcast. However, IPv6 doesn't use broadcasts. It uses **Internet Control Message Protocol version 6 (ICMPv6)** with a solicited-node multicast address message that is directed to a specific host. As a result, if you are analyzing a network that only uses IPv6, you may only see a few ARP broadcasts, if any.

As we can now understand, ARP is a common protocol that you will most likely see while doing analysis, as IPv4 is still widely used. So that you better understand a standard ARP request and reply, the following section provides an overview of an ARP header and corresponding field values.

## Exploring ARP headers and fields

While in a trace file, if you enter `arp` in the display filter, you will most likely see a series of ARP requests and replies. Within each ARP header, there are several field values, such as **Operation Code (Opcode)**, sender, and target IP address, which help ensure that the ARP requests/replies are received. Let's first take a look at a typical ARP transaction.

## Identifying a standard ARP request/reply

Open `ARPTTrace.pcapng`, and take a look at the first two packets, which are the ARP request/reply. Expand each ARP request/reply, and you will see that ARP is wrapped in an Ethernet frame. However, there are no network or transport layer headers.

First, let's investigate the components of an ARP request.

## Inspecting an ARP request

To view a standard ARP request, expand **Frame 1**. Once expanded, you can see that there are several field values, which provide information about the transaction, as shown here:

```

> Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 3
> Ethernet II, Src: 00:15:5d:0f:49:18, Dst: ff:ff:ff:ff:ff:ff
* Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: 00:15:5d:0f:49:18
  Sender IP address: 172.16.2.3
  Target MAC address: 00:00:00:00:00:00
  Target IP address: 172.16.2.27

```

Figure 16.7 – Frame 1 ARP request

We see within this ARP request that the host at 172.16.2.3 is requesting the MAC address of the host that has the IP address 172.16.2.27. Key field values include Opcode, which shows that this is a request, along with the IP addresses of both the sender and target.

The ARP reply is similar to the request; however, the target MAC address will be present.

## Evaluating an ARP reply

**Frame 2** is the ARP reply, where the host at 172.16.2.27 replies with its MAC address, which is d4:be:d9:af:3e:4f, as shown here:

```

> Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 3
> Ethernet II, Src: d4:be:d9:af:3e:4d, Dst: 00:15:5d:0f:49:18
* Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: d4:be:d9:af:3e:4f
  Sender IP address: 172.16.2.27
  Target MAC address: 00:15:5d:0f:49:18
  Target IP address: 172.16.2.3

```

Figure 16.8 – Frame 2 ARP reply

Key field values include the Opcode, which shows that this is a reply, along with the MAC and IP addresses of both the sender and target. Once the ARP reply is received, the MAC address is resolved.

As shown, an ARP header has several field values, which we'll review next.

## Breaking down the ARP header fields

In this section, we'll review the ARP field values. So that you understand the role and purpose of each field, I will list the following:

- Name of the field
- Number of bytes
- Purpose of the field

To follow along, open `ARPTrace.pcapng` and select **Frame 22**. Expand the ARP header as follows:

```
~ Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: 00:15:5d:fd:0b:0a
  Sender IP address: 172.16.2.4
  Target MAC address: 00:00:00:00:00:00
  Target IP address: 172.16.2.27
```

Figure 16.9 – Frame 22 ARP request

Within an ARP header, the field values that provide information on the ARP transaction are outlined in the following table:

Field name	Bytes	Description
Hardware type	2	This lists the type of hardware used in the current transaction. In <i>Figure 16.9</i> , the hardware type is listed as Ethernet (1), which is common in today's networks.
Protocol type	2	This lists the network layer protocol in use. In <i>Figure 16.9</i> , the protocol type is listed as IPv4 (0x0800).
Hardware size	1	This lists the number of bytes of a hardware (or MAC) address. <i>Figure 16.9</i> lists Hardware size: 6. A MAC address is 6 bytes (or 48 bits), which is a standard MAC address length.
Protocol size	1	This lists the bytes reserved for the IP address. <i>Figure 16.9</i> lists Protocol size: 4. An IPv4 address is 4 bytes (or 32 bits), which is the length of an IPv4 address.
Opcode	2	This lists what operation the sender is executing. Although the Internet Assigned Numbers Authority (IANA) lists several Opcodes, the value is most likely request (1) or reply (2).
Sender MAC address	Variable	Frame 22 lists the sender MAC address as 00:15:5d:fd:0b:0a, which is the MAC address of the host sending the request.
Sender IP address	Variable	This is the network address of the sender. Frame 22 lists 172.16.2.4 as the sender's IP address.
Target MAC address	Variable	This is the MAC address of the target. In Frame 22, which is an ARP request, the target MAC is listed as all zeros, or 00:00:00:00:00:00, because the target MAC address is unknown.
Target IP address	Variable	This is the network address of the target. Frame 22 lists Target IP address: 172.16.2.27.

Table 16.1 – APR field values

Some of the parameters, such as the Opcode and hardware type are represented by a number. The values are defined by [iana.org](https://www.iana.org/assignments/arp-parameters/arp-parameters.xhtml), and can be found here: <https://www.iana.org/assignments/arp-parameters/arp-parameters.xhtml>.

Some of the defined values include the following:

- Hardware Type 31 – IPsec tunnel
- Hardware Type 18 – Fiber Channel

Now that we can see the header and fields in a standard ARP header, let's take a look at some other types of ARP you might encounter during an analysis.

## Examining different types of ARP

On an IPv4 network, the most common types of ARP messages are requests and replies:

- A standard ARP *request* is a broadcast message that is sent out on the network requesting a resolution of an IP address to a MAC address.
- A standard ARP *reply* is a unicast message that is sent to the requesting host that provides the address resolution.

However, as we'll outline in this section, there are a few other types of ARP messages. We'll start with the **Reverse Address Resolution Protocol (RARP)**, which is the reverse of ARP.

## Reversing ARP

The opposite of ARP is RARP, in that with RARP, a client requests its IPv4 address from a computer network by using its MAC address.

Using an example `rarp_request.cap` found at [https://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=view&target=rarp\\_request.cap](https://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=view&target=rarp_request.cap), we can see the host requesting its IP address, as shown in the following screenshot:

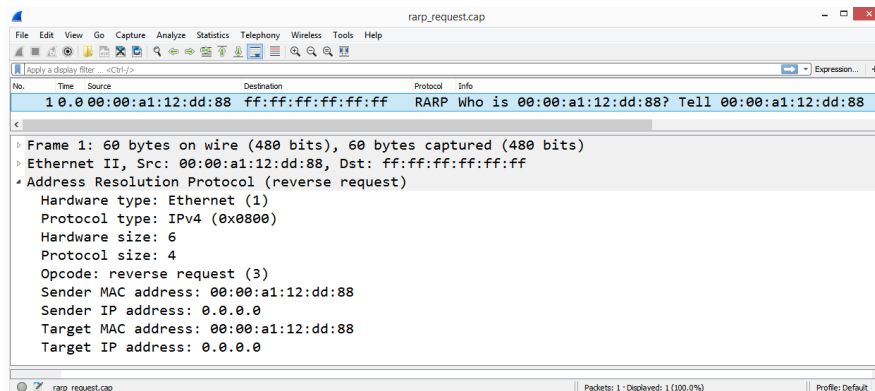


Figure 16.10 – Viewing a RARP request

The sender's MAC address is `00:00:a1:12:dd:88`. The IP address is unknown, so it is listed as `0.0.0.0`. The Opcode is `reverse request (3)`.

#### Note

RARP is an obsolete protocol that has been replaced by more efficient protocols, such as the **Dynamic Host Configuration Protocol (DHCP)**; therefore, you won't see any RARP traffic on a LAN. However, RARP is used in virtualized environments to keep the MAC tables updated.

Next, let's look at a lesser-known type of ARP called the **Inverse Address Resolution Protocol (InARP)**.

## Evaluating InARP

InARP is an extension of the ARP protocol. We can see an example of this by going to CloudShark at <https://www.cloudshark.org/captures/87be3b4b6625> and opening `FrameRelay-101.pcap` in Wireshark, as shown here:

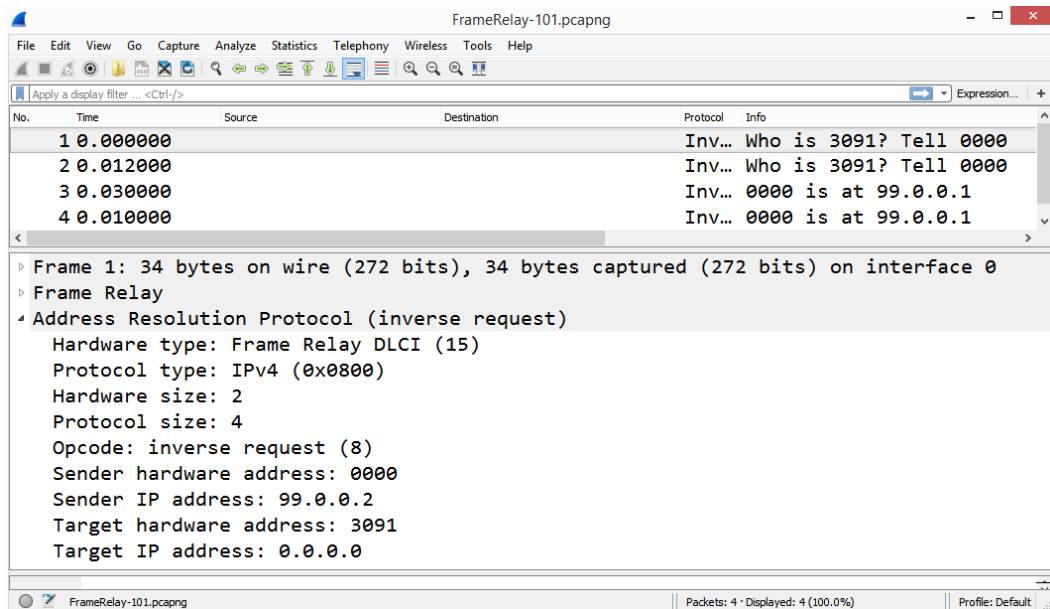


Figure 16.11 – An InARP example

On a **wide area network (WAN)**, Frame Relay was a widely used packet-switching method to transmit data between LANs. When communicating with one another, each router's interface uses a **data link connection identifier (DLCI)** of the desired station, which is the device's hardware address.

InARP is used in the same way as a standard ARP in resolving an IP address. However, it does not use broadcasts, as it already knows the DLCI of the desired station. As shown in the following diagram, the sending router, 99.0.0.2, attempts to resolve the IP address of DLCI 3091 by using InARP:

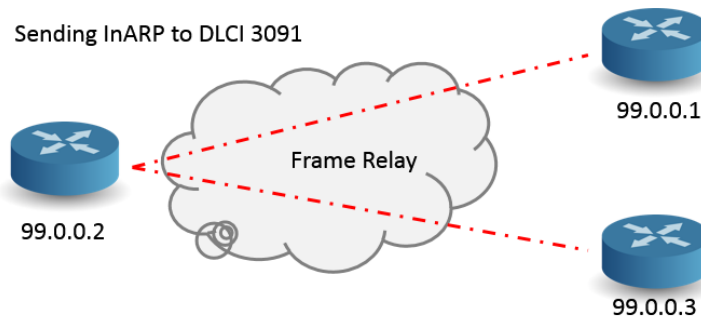


Figure 16.12 – InARP over Frame Relay

The use of Frame Relay is being phased out and replaced by a more efficient method of transmitting data on a WAN, **Asynchronous Transmission Mode (ATM)**. As a result, you will rarely, if ever, see InARP in a capture.

While you may not see InARP while conducting an analysis, you most likely will see a more common type of ARP called a gratuitous ARP, as outlined next.



## Issuing a gratuitous ARP

On a LAN, a duplicate IP address can cause conflicts. To prevent this, hosts issue a **gratuitous ARP**, which is an unsolicited ARP used to prevent duplicate IP addresses. To see an example, go to <https://www.cloudshark.org/captures/54af88021aa8>, and then download and open the file in Wireshark. Once open, expand the ARP header as shown:

```

  ▾ Address Resolution Protocol (ARP Announcement)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    [Is gratuitous: True]
    [Is announcement: True]
    Sender MAC address: VMware_37:5f:f5 (00:0c:29:37:5f:f5)
    Sender IP address: 192.168.130.128 (192.168.130.128)
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.130.128 (192.168.130.128)

```

Figure 16.13 – Gratuitous ARP

In this frame, we see Opcode: request (1). However, we also see that Sender IP address: 192.168.130.128 is the same as Target IP address: 192.168.130.128, which identifies the frame as a gratuitous ARP.

### Note

Wireshark will identify a gratuitous ARP as ARP Announcement.

A gratuitous ARP request is sent as a broadcast; however, no reply is expected. In addition to checking for duplicate IP addresses, this type of ARP can provide a way for a host to share IP and MAC address pairings. For example, if a network device's interface goes down and then is brought back up, a gratuitous ARP is sent so that all hosts can update their ARP tables.

Next, we'll see an example that is not actually an ARP type, but a *technique* used on the network called a proxy ARP.

## Working on behalf of ARP

A **proxy** is something that works on behalf of another entity. A **proxy ARP** is a technique whereby another entity resolves a MAC address. On a network, there are a few instances when a proxy ARP can be used.

One way to use a proxy ARP is when concealing a host behind a firewall.

## Obscuring a host

When a machine with a public IP address is in a private network behind a firewall, it's best to hide its existence.

If you do have a host behind a firewall, a way to resolve the MAC address is by having the firewall use a proxy ARP to and from the hidden device. This will maintain the illusion that the machine is on the public side in *front* of the firewall.

A proxy ARP can also be used in a LAN when a host in one subnetwork is separated by a proxy router.

## Acting as a proxy

On a LAN, it's sometimes necessary to use a proxy ARP. For example, a proxy ARP is necessary when an ARP broadcast is sent to a host on another subnetwork. Because an ARP request is a broadcast, it will stop at the router's interface.

In this case, the router responds with its own MAC address and acts as a proxy to the host on the other subnetwork. The router will encapsulate the ARP request packet with its own MAC address as the source and will then issue a broadcast on the appropriate network, as shown here:

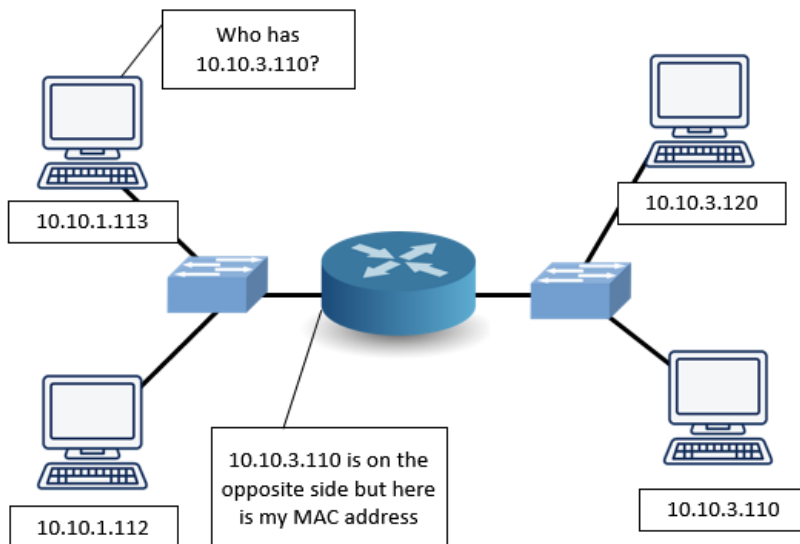


Figure 16.14 – Proxy ARP

You can now understand that there are many different types of ARP messages and techniques that may be used on a LAN. ARP is an essential protocol but can be a vulnerable target. In the next section, let's take a look at some ARP attacks along with defense methods.

## Comparing ARP attacks and defense methods

ARP is a widely used protocol that resolves an IP address to a MAC address. In most cases, ARP works well to ensure devices can find one another. However, the protocol was standardized many years ago, and there has never been a way to ensure the authenticity of ARP messages.

As a result, we'll outline a few ARP attacks that can misdirect traffic and interfere with normal network behavior. In response, we'll also take a look at some of the ways to defend against these types of attacks.

Let's begin by covering some of the attacks against ARP.

## Comparing ARP attacks and tools

ARP is used on a LAN, which can be a vulnerable target. Some of the attacks and techniques used to penetrate an ARP framework include spoofing and storming, which can misdirect traffic or cause problems on the network. In this section, we'll compare some of the attacks you or a colleague might encounter.

Let's start with using ARP in a way that can trick or deceive hosts on a network.

## Discovering ARP spoofing

On an IPv4 network, ARP is a critical protocol used when delivering data to the correct host. A malicious actor can redirect traffic to their machine by using an ARP cache poison (or ARP spoofing) attack on the LAN. This man-in-the-middle attack is done by tricking hosts on the network into believing the malicious actor's MAC address is some other host, with the intention of intercepting traffic.

### Note

For a more in-depth review of an ARP spoofing attack, go to *Chapter 1, Appreciating Traffic Analysis*, in the *Arming hackers with information* subsection.

In addition to ARP spoofing, an attacker can launch an ARP storm, as discussed next.

## Reviewing the ARP storm

On a LAN, it's normal to see ARP request/reply messages. However, when there is a large number of ARP requests, as shown in the following screenshot, this is an indication of an ARP storm, which is a form of **denial of service (DoS)** attack:

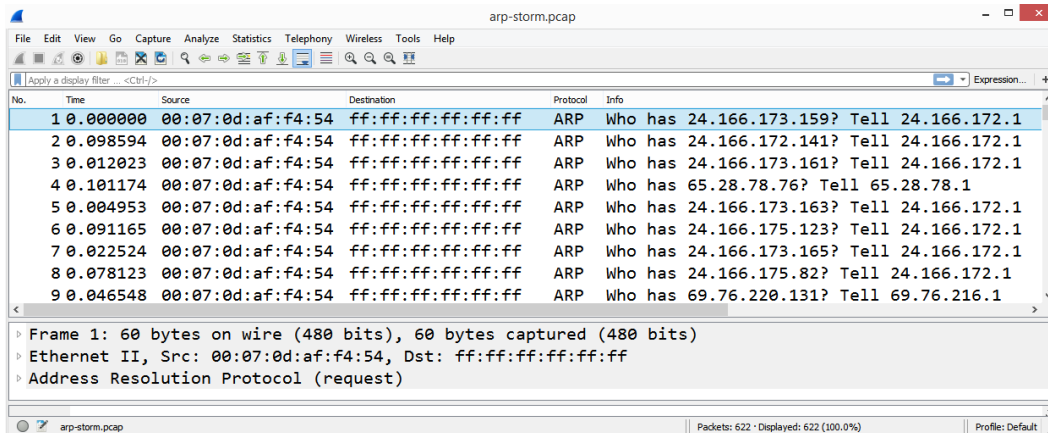


Figure 16.15 – An ARP storm

Let's step through how an ARP storm works:

1. In order to efficiently deliver data, a switch uses a **content-addressable memory (CAM)** table that contains pairings of MAC addresses and their associated physical switch ports.
2. An ARP storm floods the CAM table and overwhelms the switch with thousands of bogus entries.
3. At that point, the switch simply acts as a hub and sends data out from all of the ports, which can result in the following:
  - Allowing traffic sniffing that could possibly expose sensitive data
  - Preventing the network from functioning normally

As you can see, an ARP storm is possible. In Wireshark, you can monitor for ARP storms. To modify this, select an ARP header, right-click, and select **Protocol Preferences | Open Address Resolution Protocol Preferences...**, which will display the following screenshot:

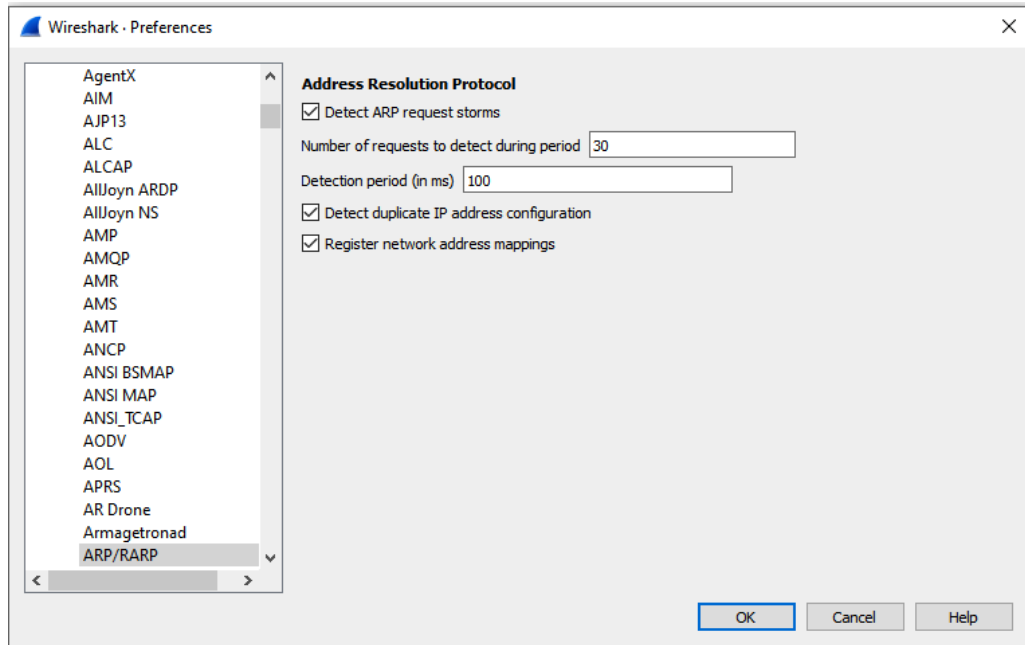


Figure 16.16 – ARP/RARP preferences

Once in the **Preferences** menu, you can modify the following:

- **Number of requests to detect during period:** Enter an appropriate value, for example, 30 requests.
- **Detection period (in ms):** Enter an appropriate value, for example, 100 ms.

Although Wireshark can't prevent an ARP storm, it can help you identify a potential attack.

We now know that there are attacks such as spoofing and storming. How are these attacks launched? The following section outlines some of the tools that are available to launch an ARP attack on a LAN.

## Understanding ARP attack tools

ARP attacks can occur on a LAN. Many tools that are available are built into Kali Linux. Kali Linux is a collection of software tools designed to assist the network administrator with conducting ethical hacking on a network. For a complete list, go to <https://tools.kali.org/tools-listing>.

Some of the tools in Kali Linux that are used to launch an ARP attack include the following:

- **dsniff** is a suite of tools that includes **arpspoof**, which allows a hacker to advertise a spoofed MAC address as a way to misdirect traffic.
- **Ettercap** is an easy-to-use tool for ARP attacks, along with other tools that make it possible to intercept network traffic.
- **Arpoison** is a free command-line tool that allows a hacker to custom build an ARP packet and define the sender and target addresses.

As shown, there are several ways to launch an ARP attack, and hackers have many tools at their disposal. The following section outlines some of the ways we can defend against ARP attacks.

## Defending against ARP attacks

As discussed, ARP can be a vulnerable target. In addition to the attacks listed, there are others as well. The network administrator should be aware of methods to detect, as well as defend against, these types of attacks.

Some of the tools and techniques to prevent an ARP attack include the following:

- **Intrusion detection systems/intrusion prevention systems (IDSs/IPSs):** Tune devices to monitor for abnormal ARP activity such as ARP storms, which generally have specific signatures. The device should send an alert if unsolicited replies are detected.
- **Static ARP entries:** Hardcode address mappings to prevent spoofing. Although static ARP entries can prevent an ARP spoof, this isn't the best defensive method, as it doesn't scale well with large networks.
- **Firewalls:** Use an access-control list with packet filtering to ensure only authorized traffic is allowed on the network segment.

- **Anti-ARP software:** This software monitors for spoofing, which can present itself as two IP addresses having the same MAC address. Many times, the software will also include methods to detect other malicious ARP behavior.
- **Secure Neighbor Discovery (SEND):** On an IPv6 network, it is possible to use the SEND protocol. This is an extension of NDP that provides authentication by using cryptographic techniques and reduces the ability to successfully launch an ARP spoofing attack on a LAN.

While it's possible for a network to fall victim to malicious ARP activity, it's best to activate possible defensive strategies, to prevent a crippling attack.

## Summary

By now, you have a better understanding of ARP, how it works, and why it is important for delivering data. So that you have an appreciation of what takes place during an ARP request and reply, we stepped through the process, and then reviewed the ARP headers and field values. We also discussed the fact that, in addition to a standard ARP, you may encounter different types of ARP while analyzing traffic in Wireshark, such as a gratuitous ARP or InARP. Finally, so that you are aware that ARP may be used in a malicious way, we covered some of the attacks, along with a few of the tools used to launch an ARP attack. We then summarized some of the methods you should employ to defend against various types of attacks.

The next chapter will look at how Wireshark can help identify and troubleshoot network latency issues. You'll be able to appreciate the importance of time values on a network and discover several ways to display time. In addition, you'll see the value of coloring rules within Wireshark and the Intelligent Scrollbar that helps highlight interesting traffic. Finally, you'll learn how to use the expert information, so that you can zero in on trouble spots during an analysis.

## Questions

Now, it's time to check your knowledge. Select the best response, and then check your answers with those in the *Assessment* appendix:

1. On a LAN, the \_\_\_\_\_ layer uses the MAC address of the destination machine, rather than the IP address:
  - A. Network
  - B. Presentation
  - C. Data link
  - D. Transport
2. The ARP Opcode lists what operation the sender is executing. Although there are many, the most common opcodes are \_\_\_\_\_.
  - A. 10 and 12
  - B. 0 and 8
  - C. 7 and 8
  - D. 1 and 2
3. With a(n) \_\_\_\_\_, a client requests its IPv4 address from a computer network by using its MAC address:
  - A. InARP
  - B. RARP
  - C. Gratuitous ARP
  - D. Proxy ARP
4. The \_\_\_\_\_ is an unsolicited ARP used to prevent duplicate IP addresses on a network. No reply is expected:
  - A. InARP
  - B. RARP
  - C. Gratuitous ARP
  - D. Proxy ARP



5. An ARP \_\_\_\_\_ is a large number of ARP requests that creates a DoS attack and prevents the network from functioning normally:
  - A. Storm
  - B. Spoof
  - C. Gratuitous
  - D. Inverse
6. The \_\_\_\_\_ defines what operation the sender is executing:
  - A. Storm
  - B. Spoof
  - C. Opcode
  - D. Protocol type
7. A possible defensive strategy to defend against an ARP attack is to use a(n) \_\_\_\_\_ and tune the device to monitor for abnormal ARP activity such as ARP storms:
  - A. Otter Box
  - B. IDS/IPS
  - C. Gratuitous Code
  - D. CAM alert

## Further reading

Please refer to the following links for more information:

- For a list of hardware types identified in the ARP header, go to the following: <https://www.iana.org/assignments/arp-parameters/arp-parameters.xhtml#arp-parameters-2>.
- The Opcode defines what operation the sender is executing. View the entire approved list of Opcodes at the following: <https://www.iana.org/assignments/arp-parameters/arp-parameters.xhtml#arp-parameters-1>.
- Discover when hosts issue a gratuitous ARP by visiting the following: [https://wiki.wireshark.org/Gratuitous\\_ARP](https://wiki.wireshark.org/Gratuitous_ARP).

# Part 5

# Working with Packet Captures

In this section, we'll discover ways to enhance your analysis skills by using Wireshark's built-in tools to troubleshoot network latency issues. We'll also learn how to subset traffic, and save and export data. Additionally, we'll explore the statistics menu and create I/O and stream graphs. Finally, we'll investigate CloudShark, a browser-based app used to study and share captures online.

The following chapters will be covered under this section:

- *Chapter 17, Determining Network Latency Issues*
- *Chapter 18, Subsetting, Saving, and Exporting Captures*
- *Chapter 19, Discovering I/O and Stream Graphs*
- *Chapter 20, Using CloudShark for Packet Analysis*



# 17

# Determining Network Latency Issues

On an enterprise network, it's challenging to manage the everyday demands of keeping the network operational 99.999 percent of the time. The network administrator constantly monitors for issues that can cause a disruption. This situation can be volatile as one error can cause the network to go down. Is there a solution that can help us mitigate this challenge? Fortunately, there is, as Wireshark has several built-in tools that help you troubleshoot the network.

In this chapter, we will address network latency and recognize some of the reasons why packet loss and slow response times occur. You'll gain a better appreciation of the importance of time values while troubleshooting. We'll cover the coloring rules that help identify issues and are used in the Intelligent Scrollbar, so you can quickly identify and move to trouble spots in the capture. Finally, you'll also learn how to navigate the expert information Wireshark generates, which subdivides alerts into categories and guides the analyst through a more targeted evaluation.

This chapter will address all of this by covering the following:

- Analyzing latency issues
- Understanding the coloring rules
- Exploring the Intelligent Scrollbar
- Discovering expert information

## Analyzing latency issues

Today, there are many different types of devices that communicate and exchange information across the network, which include intermediary devices, the **Internet of Things (IoT)**, and mobile devices. All of those, along with the many other types of traffic that are added to the network on a daily basis, can make network administration challenging.

Because of these factors, there are multiple reasons why packet loss and slow response times occur. Once it's determined that there is an issue, the troubleshooting process begins.

When troubleshooting connectivity issues, there are many approaches. All have the same goal: identify the trouble spots and narrow the scope to determine the root cause of the problem. Root causes can include misconfiguration, malware, or even hardware malfunction. In the following sections, we will analyze some of the root causes behind network delays and discuss three main concepts: latency, throughput, and packet loss.

## Grasping latency, throughput, and packet loss

When users complain of slow response times, the network administrator can do a quick packet capture and observe evidence of trouble. In this section, we will walk through some examples to demonstrate how you can identify issues on the network. If you would like to follow along, go to <https://www.cloudshark.org/captures/9a5385b43846>, download the `client-fast-retrans.pcap` capture, and open it in Wireshark.

Once the file is open, we can scroll through the capture. Around packets 20-21, we can see potential problems, as indicated by the black coloring rule seen within the capture:

No.	Time	Source	Destination	Protocol	Info
17	0.000	74.203.22.229	230.211.187.172	TCP	49683 → http(80) [ACK] Seq
18	0.000	230.211.187.172	74.203.22.229	TCP	http(80) → 49683 [ACK] Seq
19	0.000	74.203.22.229	230.211.187.172	TCP	49683 → http(80) [ACK] Seq
20	0.000	230.211.187.172	74.203.22.229	TCP	[TCP Previous segment not
21	0.000	74.203.22.229	230.211.187.172	TCP	[TCP Dup ACK 19#1] 49683 →
22	0.000	230.211.187.172	74.203.22.229	TCP	http(80) → 49683 [ACK] Seq
23	0.000	74.203.22.229	230.211.187.172	TCP	[TCP Dup ACK 19#2] 49683 →
24	0.000	230.211.187.172	74.203.22.229	TCP	[TCP Fast Retransmission]
25	0.000	74.203.22.229	230.211.187.172	TCP	49683 → http(80) [ACK] Seq

Figure 17.1 – Viewing client-fast-retrans.pcap

As shown, both duplicate **Acknowledgments** (ACKs) and fast retransmission errors are evident in packets 20-21 and 23-24. Both can periodically occur on a network. However, when there is an excessive number of them, this is generally an indication of congestion.

#### Note

We'll learn more about the different types of transmission errors later in the chapter.

Commonly, there are three indicators when measuring performance:

- Latency
- Packet loss
- Throughput

Let's take a look at each of these, beginning with latency.

## Computing latency

Latency is a measurement of how long it takes to transmit a packet from one point to another. Network latency bogs down the network and can create delays. In addition, it can cause web pages to slow down when retrieving content and can also have a negative effect on voice and video applications as well.

Latency can be measured using **Round-Trip Time (RTT)**, which is how long it takes to make a complete round trip from A to B, and then from B to A.

An optimal RTT remains steady, as shown in the following screenshot:

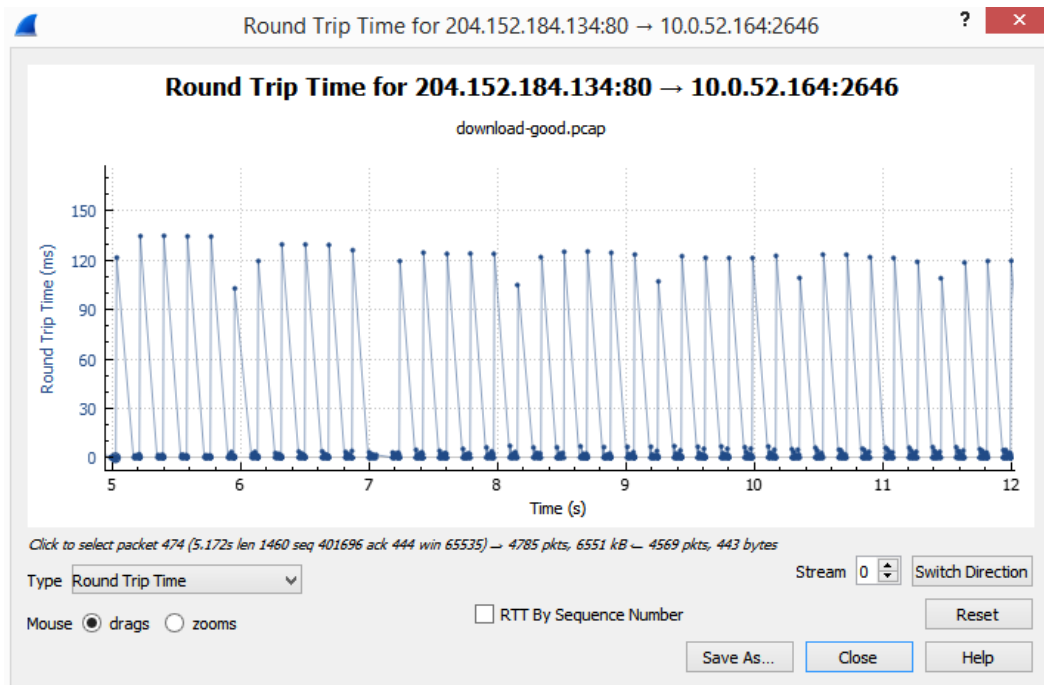


Figure 17.2 – Steady RTT

However, you won't always see a steady RTT, as this value can increase and vary during the course of transmission. When there is latency, you will see an increase in the RTT.

To see the RTT for a particular stream in Wireshark, go to the menu and choose **Statistics | TCP Stream Graphs | Round Trip Time**, as shown here:

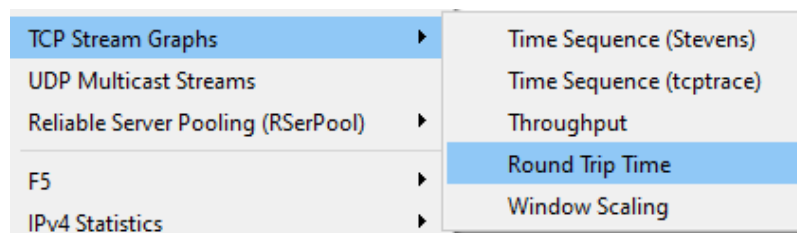


Figure 17.3 – TCP Stream Graphs menu

When you select **Round Trip Time**, Wireshark will open the graph. Once in, make sure you are viewing the correct stream direction, which you can modify by using the **Switch Direction** button in the lower right-hand corner.

To see an example of an RTT slowly increasing, follow these steps:

1. Open the `client-fast-retrans.pcap` file in Wireshark.
2. Generate a stream graph by going to **TCP Stream Graphs | Round Trip Time**. Once selected, Wireshark will open the graph, as shown here:

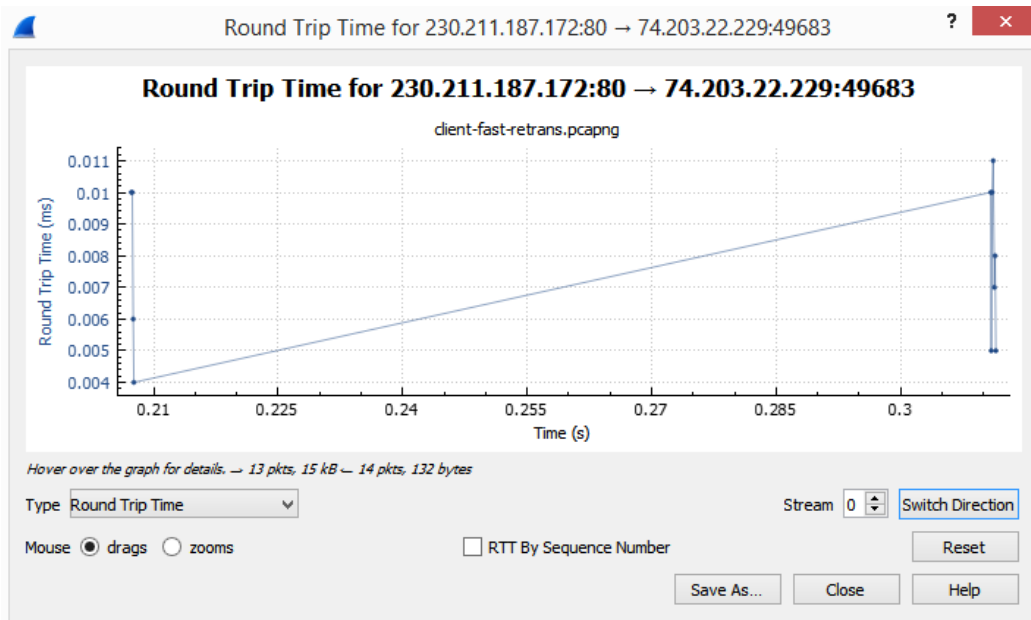


Figure 17.4 – RTT graph for stream 0

When looking at the graph from `client-fast-retrans.pcap`, we see that the RTT between `230.211.187.172:80` and `74.203.22.229:49683` (or stream 0) is increasing over time. This is most likely due to latency on the network.

#### Note

We will cover the different types of TCP stream graphs in more detail in *Chapter 19, Discovering I/O and Stream Graphs*.

Latency refers to how long it takes to transmit a packet and is measured in RTT. When there is high latency, the sender has difficulty sending data, and as a result, less data is able to get to the receiver. Next, let's take a look at throughput.



## Measuring throughput

Throughput is how much data is sent and received (typically in bits per second) at any given time. In Wireshark, we can measure this as well as goodput, which is useful information that is transmitted.

Network media can affect throughput. For example, fiber optics has better throughput than copper. Congestion and delays can also affect how much data is getting through. When there is decreased throughput, packet loss can occur, as discussed next.

## Experiencing packet loss

When there is latency, data may not be getting through, which can lead to packet loss. Losing or dropping packets on a network occurs for a variety of reasons. Packet loss is determined by the number of packets lost for every 100 packets sent.

Endpoints and applications work to manage transmission delays and network congestion. However, there are times when excessive packet loss occurs, and the network goes into recovery mode. In Wireshark, we see evidence of packet loss with indicators such as keep-alivess, duplicate ACKs, and retransmissions.

Wireshark is capable of identifying many common transmission errors and can calculate delays and interruptions in the data flow by using time values. The following section provides an insight into the significance of time while doing an analysis.

## Learning the importance of time values

When doing an analysis on a packet capture, time values can provide an insight into the delays in transmission. In Wireshark, there are choices as to how you can display the time value, which include the following:

- **Seconds Since Beginning of Capture**
- **Seconds Since Previously Captured Packet**
- **Seconds Since Previously Displayed Packet**

It's important to use the correct time format. In most cases, it's best to select **Seconds Since Previously Displayed Packet**, which will show delays whether or not you used a display filter.

Network latency and transmission errors occur on the network. Fortunately, Wireshark has a way to help identify common issues in the form of coloring rules. The following section outlines Wireshark's coloring rules and how you can use them in your analysis.

## Understanding coloring rules

Built within Wireshark are coloring rules or filters, which identify or highlight specific traffic. Locate the default coloring rules by going to the menu and choosing **View | Coloring Rules**, as shown in the following screenshot:

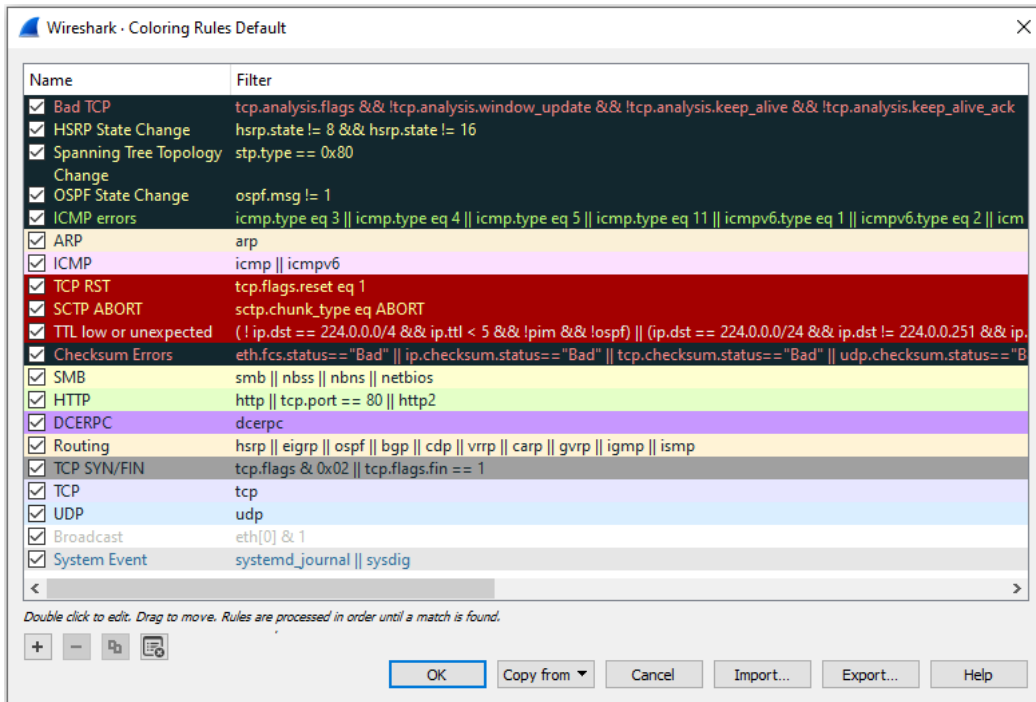


Figure 17.5 – Default coloring rules

Once you are in the **Coloring Rules** menu, you can edit, delete, or add your own as needed. In addition to using the default coloring rules, you can create and share rules. An example can be found at [https://wiki.wireshark.org/Jay%27s\\_Coloring\\_Rules](https://wiki.wireshark.org/Jay%27s_Coloring_Rules).

Each rule is processed until Wireshark finds a match, according to the order shown in the console. To modify the order of a particular rule, select the rule and then drag it to the desired position.

A checkmark on the left-hand side indicates an active rule. To deactivate it, deselect the rule you do not want Wireshark to consider.

To edit a rule, do the following:

1. Select and double-click the coloring rule you want to modify.
2. You can then edit the name, or the filter used, along with the background and foreground colors.

Although Wireshark can colorize packets, in some cases, the coloring can be distracting. You can disable the coloring rules by selecting the icon, which is generally underneath the **Telephony** menu item, as outlined in the following screenshot. However, the position of the icon can vary in different versions, platforms, or layouts:

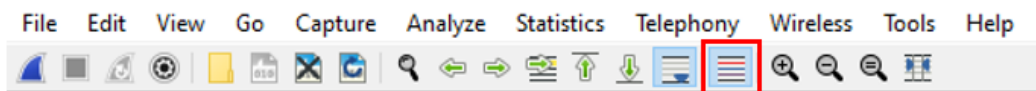


Figure 17.6 – Viewing the coloring rules icon

Wireshark summarizes the coloring rules that are in use in the frame metadata. In addition to the information listed pertaining to the time, frame, and protocols, you will see the coloring rules used. To see an example of the coloring rules summary, follow these steps:

1. Open the `client-fast-retrans.pcap` file.
2. Go to frame **20** and expand the frame metadata by clicking the arrow to the right of the **Frame 20** label.
3. At the bottom of the metadata list, you will see the coloring rules, as shown in the following screenshot:

```

~ Frame 20: 1434 bytes on wire (11472 bits), 1434 bytes captured (11472 bits)
  Encapsulation type: Ethernet (1)
  Arrival Time: Jun  2, 2015 10:11:59.966187000 Eastern Daylight Time
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1433254319.966187000 seconds
  [Time delta from previous captured frame: 0.000103000 seconds]
  [Time delta from previous displayed frame: 0.000103000 seconds]
  [Time since reference or first frame: 0.311136000 seconds]
  Frame Number: 20
  Frame Length: 1434 bytes (11472 bits)
  Capture Length: 1434 bytes (11472 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertype:ip:tcp]
  [Coloring Rule Name: Bad TCP]
  [Coloring Rule String: tcp.analysis.flags && !tcp.analysis.window_update &&

```

Figure 17.7 – Coloring rules in the frame metadata

When there is trouble on the network, you will most likely see coloring evident in the packet list. The rules provide guidelines on what traffic to home in on during analysis.

#### Note

For the coloring rules to work, they must be enabled. However, in most cases, the coloring rules are active.

In addition to the colors used in the packet list, there is also a distinct pattern on the right-hand side that is based on the active coloring rules. This pattern represents the Intelligent Scrollbar. Next, let's take a look at how Wireshark incorporates the use of the coloring rules within the Intelligent Scrollbar to easily spot problems.

## Exploring the Intelligent Scrollbar

In addition to seeing indications of problems within a capture, you can also easily spot issues using the Intelligent Scrollbar. To see an example of an active Intelligent Scrollbar, go to <http://tcpreplay.appneta.com/wiki/captures.html#bigflows-pcap>. Once there, download `bigFlows.pcap` and open it in Wireshark so that you can follow along. Open the capture and then go to frame 586 by doing the following:

1. Select the *go to specified packet* icon.
2. In the form box, type 586, then select **Go to packet**.
3. The Intelligent Scrollbar will display indications of network congestion within the packet list, as shown in the following screenshot:

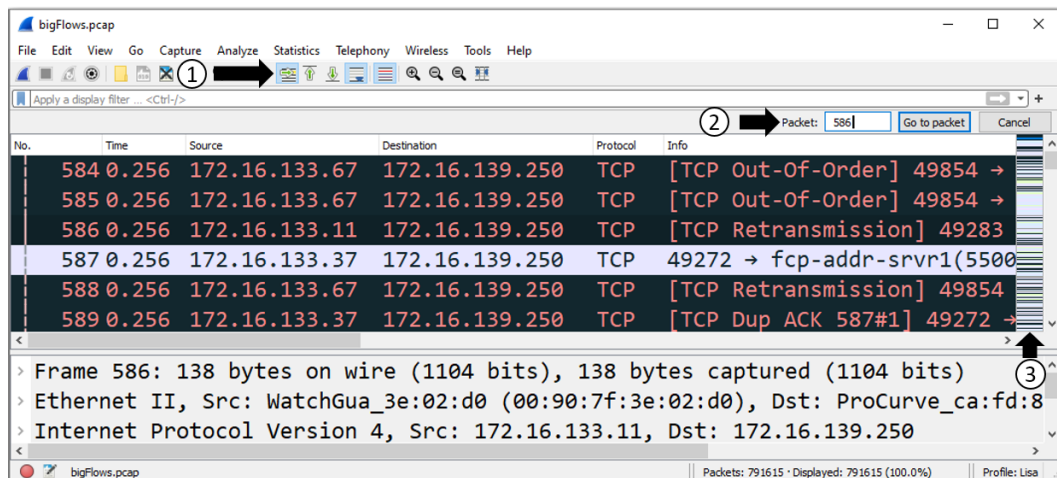


Figure 17.8 – Visualizing network congestion

The **Info** column header on the right-hand side lists several indications of trouble that warrant further investigation. These include the following:

- [TCP Out-Of-Order]
- [TCP Retransmission]
- [TCP Dup ACK 587#1]

The administrator can click on a color band and go directly to the specified packet in order to zero in on a possible problem. Once you click on a band, Wireshark will adjust the packet list to display the area of concern.

We can see how the coloring rules and Intelligent Scrollbar help to identify transmission errors and trouble spots in the capture. In the next section, we will explore common transmission errors that occur on a network.

## Common transmission errors

Long delays in intermediary devices, such as routers and switches, can cause latency, dropped packets, and/or other negative effects. When troubleshooting congestion issues in Wireshark, you may see evidence of transmission errors.

Some common indications include duplicate ACKs, keep-alive segments, and fast retransmissions.

As this information may indicate latency and gaps in the delivery of data, it's important to understand the meaning of what the packets are trying to tell you. Let's start with an overview of duplicate ACKs.

### Seeing duplicate ACKs

In a normal TCP conversation, the client recognizes every byte received by transmitting an ACK, with the ACK field value set as the next expected byte. When more than one ACK is sent by the client (with the same ACK field value), this is said to be a duplicate ACK.

To understand what a duplicate ACK is, let's step through a standard TCP transaction:

1. In the course of a normal TCP data transaction, TCP sequences and acknowledges every byte of data received.
2. The client acknowledges the data received by setting the ACK flag in the TCP header, as shown here:

```

Flags: 0x010 (ACK)
000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
.... .... 0... = Push: Not set
.... ..... 0.. = Reset: Not set
.... ..... ..0. = Syn: Not set
.... ..... ...0 = Fin: Not set
[TCP Flags: .....A....]

```

Figure 17.9 – The TCP ACK flag set

3. The client places the value of the next expected byte in the **Acknowledgment Number** field.
4. When the client sends an ACK 180 (acknowledgment number: 180), the client is saying to the server, *So far, I've received 179 bytes of data, and I am ready for more (bytes), starting with (byte number) 180*, as shown in the following diagram:

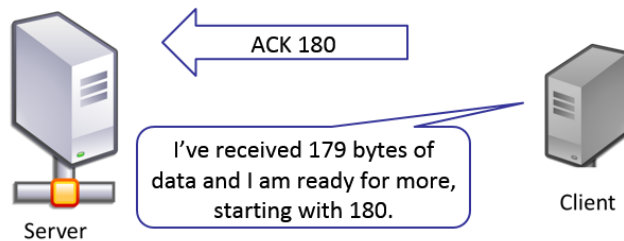


Figure 17.10 – Normal TCP ACK

5. The server doesn't wait for confirmation of delivery to send more data. Instead, the data is sent concurrently with the ACKs.

**Note**

With TCP, an ACK is expectational, in that the ACK is sent with the *next expected byte* to be sent by the server.

6. If the client sends another ACK 180 flag, the client is (again) saying to the server: *So far, I've received 179 bytes of data and I am ready for more (bytes), starting with (byte number) 180.*
7. Wireshark recognizes this as the second ACK 180 flag sent by the client and identifies this packet as a duplicate ACK. This means the client did not receive the next expected byte and is politely asking the server to send the data.

In *Figure 17.8*, as shown in the *Exploring the Intelligent Scrollbar* section, you can see a duplicate ACK in frame **589**. This indicates that the client is patiently re-requesting the missing data. In addition, in the **Info** column header, you can see [TCP Dup ACK 587#1]. This means this is the second (or duplicate) ACK flag sent after the original ACK sent in frame **587**.

Latency and delays in transmission can be caused by any number of things, such as processing and queuing delays along with general network congestion. As a result, duplicate ACKs may be sent over and over again by the client until it receives the expected data.

Network congestion is part of today's landscape and has many negative effects, such as slow web page retrieval. Another indication of transmission errors and congestion is keep-alive packets, which we will explore next.

## Observing keep-alive segments

When communicating with a web server, the client and server both use **Hypertext Transport Protocol (HTTP)** to communicate with each other. If during a session the network becomes sluggish and both sides begin to experience slow response times, HTTP uses a method called keep-alive. A keep-alive packet doesn't have any data; it has the ACK flag set, and the sequence number is set to one less than the current sequence number.

Keep-alive packets are sent between the client and the server to verify that both sides are still responding. Using this method keeps a session alive instead of dropping the connection and having to go through the expensive negotiation of reestablishing the connection.

If you would like to see an example of a keep-alive packet, go to <https://www.cloudshark.org/captures/5618ff446df8>. Once open, select **Export**, found on the right-hand side of the interface, and then select **Download the original file**, as shown here:

### Download cloudshark\_tcp-keep alive.pcapng

CloudShark retains the originally uploaded file which may be retrieved unaltered. You may also export a `pcapng` formatted file that includes all the annotations and comments added by CloudShark users.

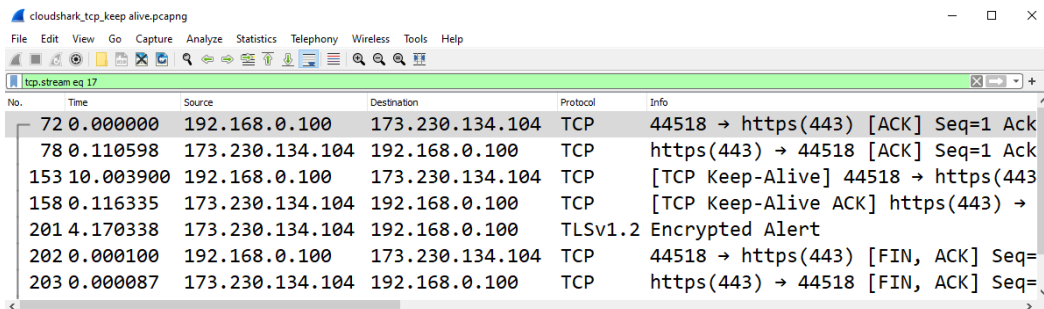
#### File selection:

- ☐ Export a new `pcapng` with CloudShark comments and annotations  
☒ Download the original file

[Download file](#) or [cancel](#)

Figure 17.11 – Export file from CloudShark

Open the `cloudshark_tcp-keep alive.pcapng` file in Wireshark. Once open, select packet **158**, right-click, and then select **Follow | TCP Stream**. You can also use the `tcp.stream eq 17` display filter. Once you have filtered the traffic, you should see the following:



No.	Time	Source	Destination	Protocol	Info
72	0.000000	192.168.0.100	173.230.134.104	TCP	44518 → https(443) [ACK] Seq=1 Ack
78	0.110598	173.230.134.104	192.168.0.100	TCP	https(443) → 44518 [ACK] Seq=1 Ack
153	10.003900	192.168.0.100	173.230.134.104	TCP	[TCP Keep-Alive] 44518 → https(443)
158	0.116335	173.230.134.104	192.168.0.100	TCP	[TCP Keep-Alive ACK] https(443) →
201	4.170338	173.230.134.104	192.168.0.100	TLSv1.2	Encrypted Alert
202	0.000100	192.168.0.100	173.230.134.104	TCP	44518 → https(443) [FIN, ACK] Seq=
203	0.000087	173.230.134.104	192.168.0.100	TCP	https(443) → 44518 [FIN, ACK] Seq=

Figure 17.12 – HTTP keep-alive packets

I have removed the coloring so you can see the exchange of keep-alive packets in frame **153** and frame **158**. In this capture, it is most likely that the network is congested, and latency is preventing the exchange of data. As a result, HTTP uses keep-alive packets between both endpoints to keep the session alive.

Therefore, in addition to seeing duplicate ACKs when there is network congestion, you may also see multiple keep-alive packets.

Next, let's take a look at another indication of slow network speeds and congestion: the presence of retransmissions.



## Issuing retransmissions

On a congested network, it's common to see retransmissions, fast retransmissions, and spurious retransmissions. All three are related. However, each has subtle differences.

First, let's talk about retransmissions and fast retransmissions on the network.

### Recognizing retransmissions

In a TCP connection, each side of a conversation actively monitors the data transaction. When congestion is evident and the data is not getting through, recovery efforts are triggered when certain conditions are met. Depending on the algorithm, you will see retransmissions or fast retransmissions where the server is actively trying to resend the missing data.

Sometimes, the data does get through and the server is not aware that the client has received the data. In that case, the server sends an unnecessary or spurious retransmission.

### Sending a spurious retransmission

During the course of the data transaction, the server may resend data that is not needed. The client has previously acknowledged that it received the data, but the server has resent the data, most likely because it did not receive the ACK. This is called spurious retransmission. Although the data is not needed, this can still be a cause for concern, as somehow, the communication to the server has been interrupted.

When just starting out with learning how to do packet analysis, it can be overwhelming. While you may not be able to identify all possible issues, Wireshark provides a guide in the form of curated expert information on the capture. This tool groups common issues together so you can quickly investigate network delays, as we'll explore next.

## Discovering expert information

While analyzing a packet capture, you may observe a colored circle in the lower left-hand corner of the interface. That is the **Expert Information** guide, which is a feature built within Wireshark that helps to alert the network administrator of possible issues once a capture has been made.

As shown in *Figure 17.8* in the *Exploring the Intelligent Scrollbar* section, the expert information icon is a red circle, which indicates an error; this is the highest expert information level.

Return to the `bigFlows.pcap` packet capture. Double-click the expert information icon in the lower left-hand corner, which will open a console, as shown in the following screenshot:

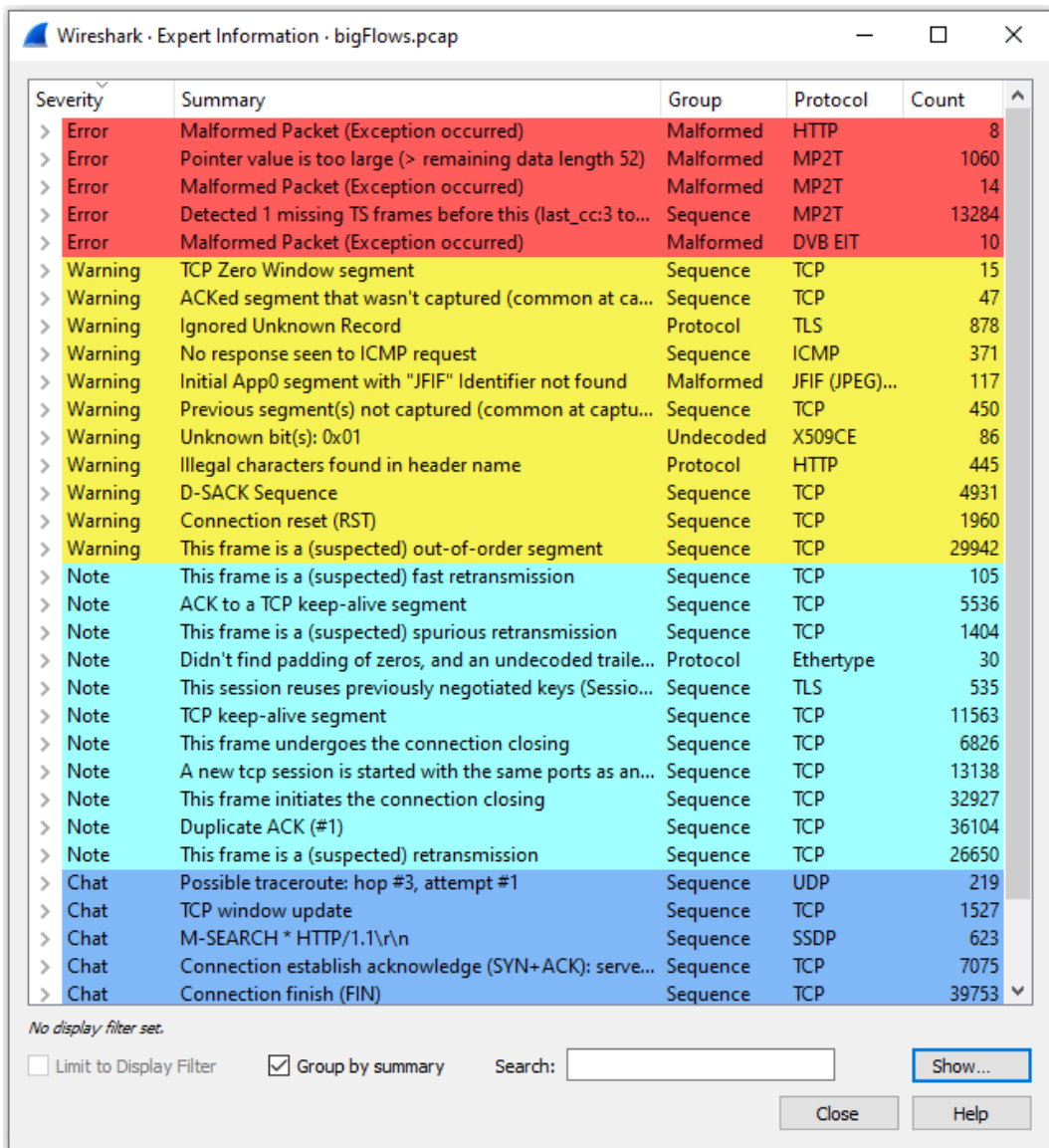


Figure 17.13 – Expert Information grouped by severity

This may take a few minutes to load, depending on the size of the capture. In addition, there may be a lot of information.

The **Expert Information** console is a GUI that allows you to see details of what Wireshark identified in the capture, so you can investigate further. The interface is intuitive, with column headers, selection checkboxes, and drop-down lists so you can customize your viewing.

Now, let's take a look at each column header in the following section.

## Viewing the column headers

While in the **Expert Information** interface, I selected **Note** and then expanded the caret next to **Duplicate ACK (#1)**, as shown:

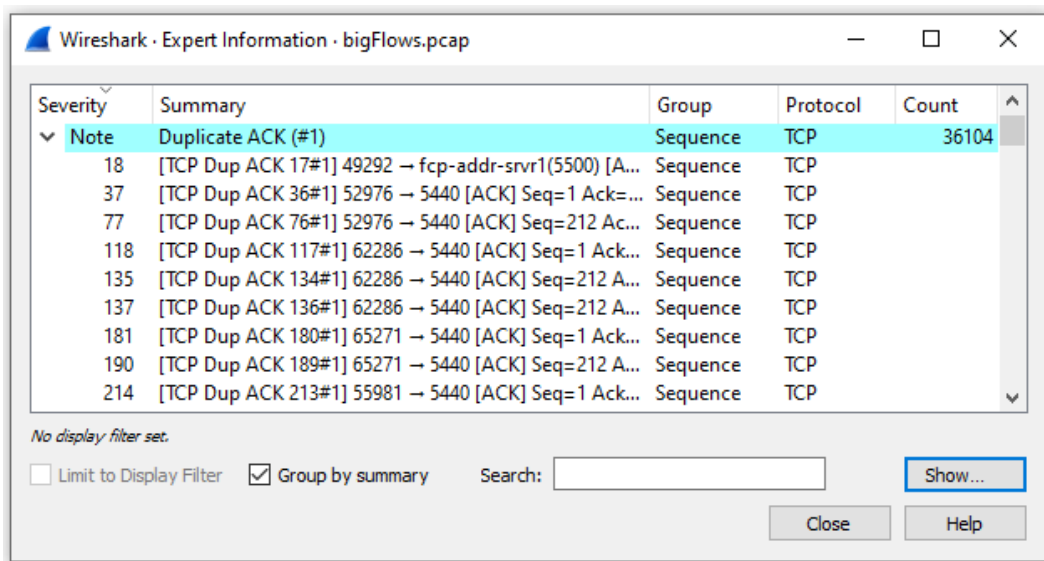


Figure 17.14 – Note: Duplicate ACK (#1)

Across the top of the interface, you will see column headers. The following bullet points outline what each header indicates:

- **Severity:** Indicates the severity of the error identified. In the preceding screenshot, the severity is listed as **Note**.
- **Summary:** Provides a summary of the error and combines all the errors that are the same under one drop-down summary. For example, in the preceding screenshot, the summary is **Duplicate ACK (#1)**. Once you expand the line, you can drill down into the individual packets to see more details on each error listed.

- **Group:** Within each summary, there are several common groupings. For example, the items listed under **Duplicate ACK (#1)** are grouped under **Sequence**. The following outline some of the groups that you might see:
  - **Checksum:** Indicates an invalid checksum
  - **Protocol:** A violation of the **Request for Comments (RFC)** for a particular protocol
  - **Sequence:** Suspicious protocol behavior
- **Protocol:** Lists the main protocol that was in use that caused the alert, such as TCP, as shown in the preceding screenshot.
- **Count:** Provides a count of the number of references for the particular event grouping. For example, on the top right-hand side of *Figure 17.14*, we see there is a count of **36104 Duplicate ACK (#1)**.

As shown, the column headers highlight the details of what the packet contains. In addition, the **Expert Information** outlines the level of severity by using color, as outlined in the next section.

## Assessing the severity

When looking at the **Expert Information** console, there are five possible categories that indicate the severity of an issue, as shown here:

Category	Color	Meaning
Error	Red	Possible serious issue and is the highest warning. Issues include a malformed packet or a new fragment overlapping old data.
Warning	Yellow	This indicates a warning, which means there may be problems that you will want to investigate further.
Note	Cyan	General notes of interest that, many times, are part of a connection. For example, a TCP keep-alive packet. Notes can also list unusual errors or a nonstandard use of a protocol, such as reusing previous session keys in a <b>Transport Layer Security (TLS)</b> conversation.
Chat	Gray	Specifies typical workflow and state changes, such as a connection finish or a TCP window update.
Comment	Green	Indicates that there is a comment found in at least one of the packets.

Table 17.1 – Expert Information severity levels

Having a visual of the issues in the packet capture is helpful, but there are even better ways to present the information. In the next section, we'll learn about ways to sort, search, and display the data.

## Organizing the information

When you open the **Expert Information** console, you'll need to make sense of the data. The interface provides ways to sort and search, along with ways to show only a certain kind of data.

We'll start with an overview of ways to sort data within the interface.

### Sorting the data

After you launch the **Expert Information** console, all the information may not be sorted. As you'll find, you can easily sort any of the column headers. I typically sort the results in order of severity.

To view all the packets for a specific summary, select the caret on the left-hand side of the summary, as shown in *Figure 17.14*.

If you have applied a display filter, you can select **Limit to Display Filter**, which is found in the lower left-hand corner, to show only your filtered results. This could be handy if you are troubleshooting a particular conversation and want to only display the filtered conversation.

The default view lists all errors, warnings, notes, and chats. However, you may only be interested in the errors. In that case, you can limit your results by using the drop-down menu in the lower right-hand corner. Once there, you can select or deselect what you would like to display, as shown here:

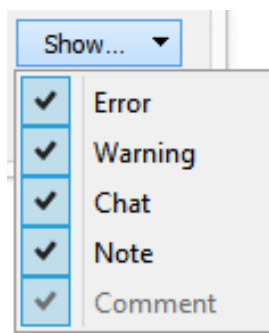


Figure 17.15 – Expert Information | Show categories

In addition, if there are any comments, you can display them as well.

As shown, you can easily sort data within the **Expert Information** dialog box. In the next section, we'll see how searching data helps to improve your ability to focus on specific issues.

## Searching for values

When you need to locate a specific value while in the **Expert Information**, enter the value in the search box and press *Enter*. The following screenshot shows the results for the `ssdp` search:

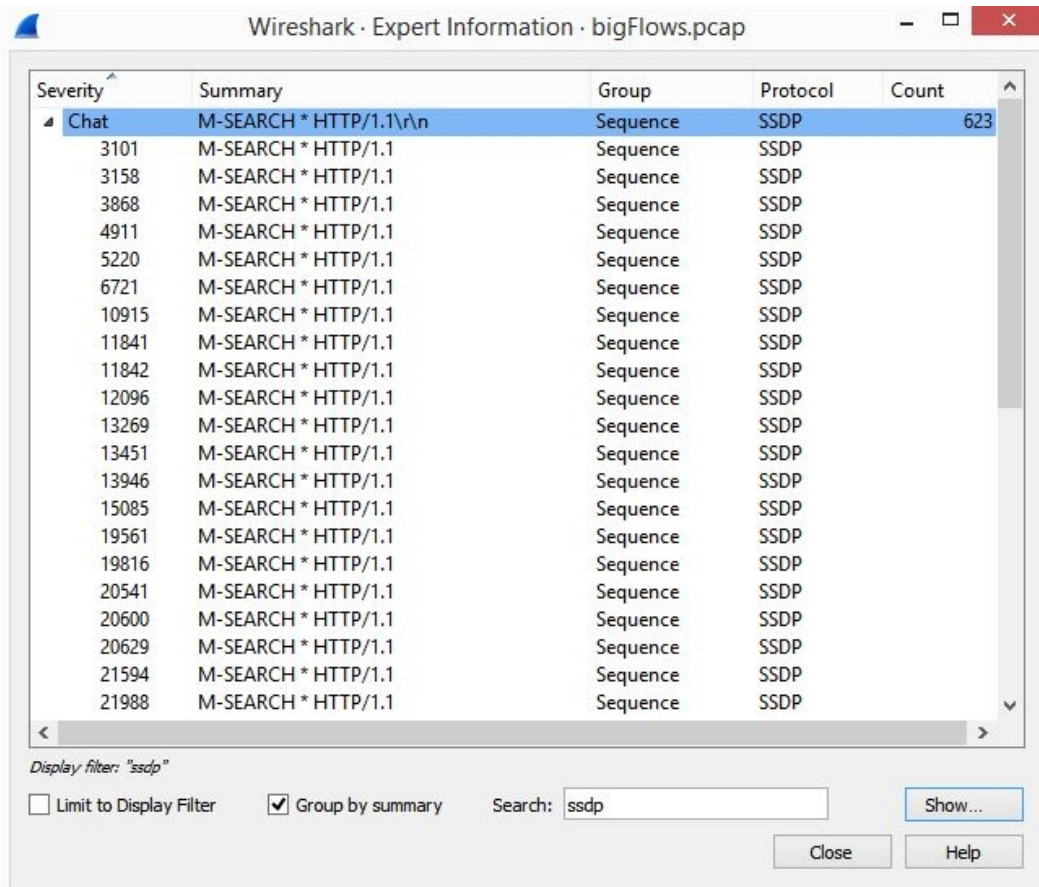


Figure 17.16 – Expert Information search results

The **Expert Information** console has an advanced menu function. As shown in the following screenshot, when you right-click on a value, you can select any of the menu choices listed:

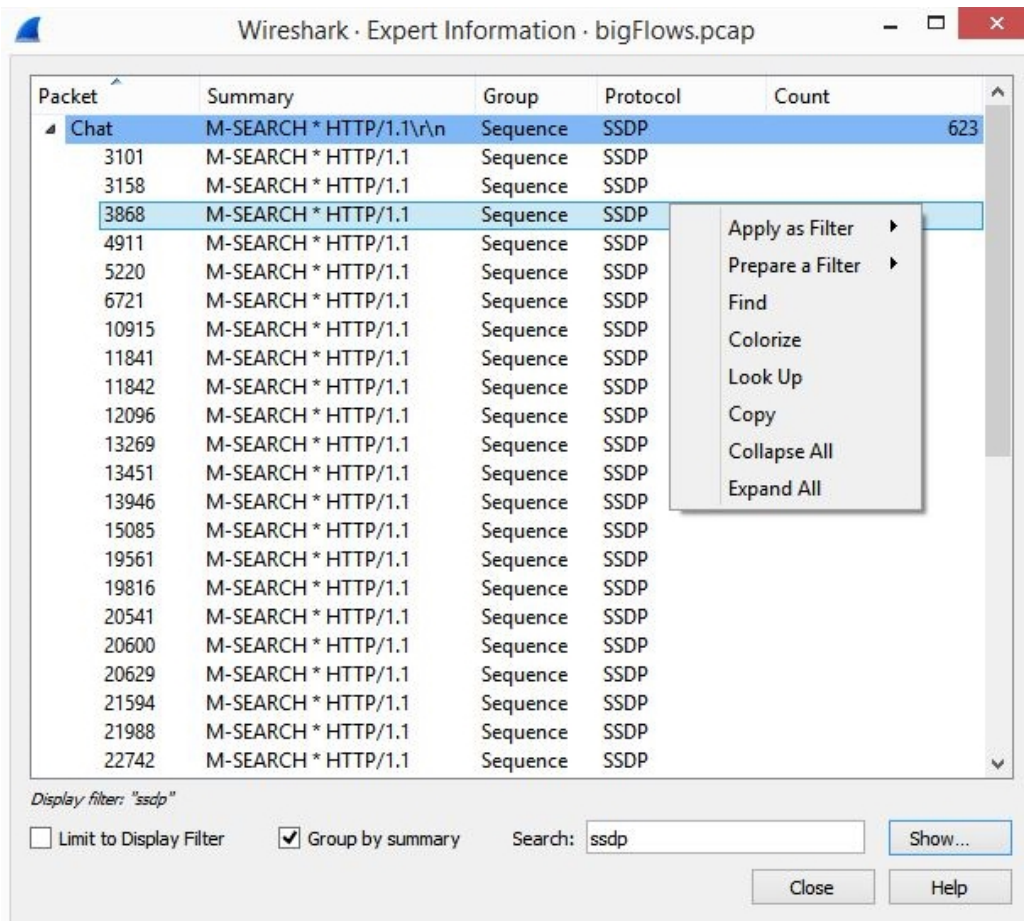


Figure 17.17 – Expert Information menu choices

Similar to the menu choices offered when you right-click on a field value when in the packet details, you can select any of the following:

- **Apply as Filter** will select the highlighted conversation and run the filter in the main interface.
- **Prepare a Filter** will select the highlighted conversation and prepare the filter in the main interface. To run the filter, you must press *Enter*.

- **Find**, when selected, will place the variables in the **Display Filter** in the main interface, as shown in the following screenshot:

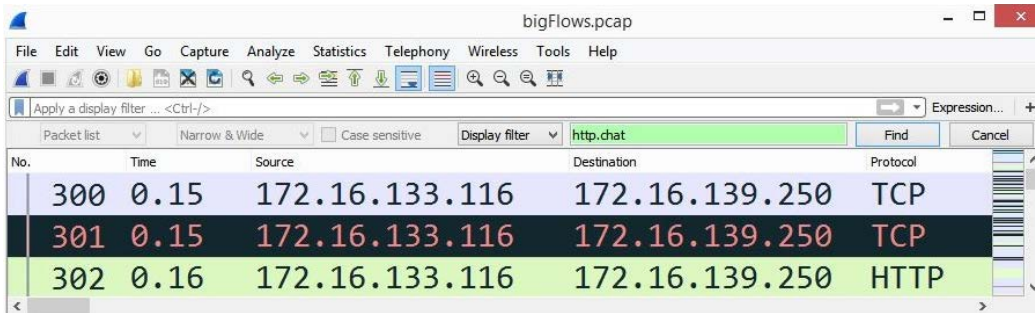


Figure 17.18 – Results of the Find menu choice

- **Colorize** will open the **Coloring Rules** dialog box and allow you to create a custom coloring rule.
- **Look Up** will open a browser, do a Google search, and present the results.
- **Copy** will copy the selected line onto the clipboard. For example, if I right-click on packet **10915** and select **Copy**, Wireshark will copy the results to the clipboard. I can then paste the results, as follows:

```
10915 SSDP: M-SEARCH * HTTP/1.1
```

- **Collapse All** will collapse the results to a single summary line.
- **Expand All** will expand the results to show all the packets.

The **Expert Information** console can provide a great deal of insight into possible problems in a packet capture. Wireshark presents the results in an easy-to-read format in the console, where you can view and analyze any errors, warnings, notes, and chats.

## Summary

Networks need to be available nearly 100 percent of the time. A single device failure, malware, or misconfiguration can significantly impact network performance. In this chapter, we reviewed how we measure performance using three main metrics: latency, throughput, and packet loss. We then looked at a few of the many tools available in Wireshark to identify trouble on the network. We discovered the importance of time values and how they factor in discovering latency issues. In addition, we learned how coloring rules can highlight specific types of traffic. We also discovered how any of the rules can be edited, deleted, or moved up or down in priority.



We then looked at the Intelligent Scrollbar, which provides a visual so that we can easily spot and further investigate trouble in a capture. We covered the ways **Expert Information** helps to alert the network administrator on possible issues once a capture has been made. We summarized by reviewing the **Expert Information** console, which we can use to drill down on specific issues and subset errors, warnings, notes, and chats.

In the next chapter, we will cover ways to work with large packet captures and break them down into smaller files for analysis. We will look at filtering packets to narrow down the results, as well as reasons and ways to add comments to a single packet or an entire capture. We will then conclude with the many ways and formats that allow us to save and export packet captures.

## Questions

Now it's time to check your knowledge. Select the best response, and then check your answers, which can be found in the *Assessment* appendix:

1. \_\_\_\_ is a metric that measures the time it takes to transmit a packet from one point to another and can be measured using RTT.
  - A. Latency
  - B. Packet loss
  - C. Goodput
  - D. Throughput
2. \_\_\_\_ is the amount of data that is sent and received (typically in bits per second) at any given time.
  - A. Latency
  - B. Packet loss
  - C. Goodput
  - D. Throughput
3. In Wireshark, the \_\_\_\_ is on the right-hand side of the packet list panel and displays a distinct coloring pattern based on the coloring rules set in the application.
  - A. Group metrics
  - B. Time values
  - C. Intelligent Scrollbar
  - D. Goodput meter

4. A \_\_\_\_\_ is a special type of packet that does not have any data. It only has the ACK flag set, so the client knows to keep the session active during an HTTP session.
- A. Duplicate acknowledgment
  - B. Keep-alive
  - C. Retransmission
  - D. Fast retransmission
5. When viewing the expert information icon, a cyan circle indicates a(n) \_\_\_\_\_, which is general information, unusual errors, or a nonstandard use of a protocol.
- A. Error
  - B. Warning
  - C. Note
  - D. Chat
6. \_\_\_\_\_ is determined by the number of packets lost for every 100 packets sent.
- A. Latency
  - B. Packet loss
  - C. Goodput
  - D. Throughput
7. It's important to use the correct time format. In most cases, it's best to select Seconds Since \_\_\_\_\_, which will show delays whether or not you used a display filter.
- A. 1970-01-01
  - B. Beginning of Capture
  - C. Previously Captured Packet
  - D. Previously Displayed Packet



# 18

## Subsetting, Saving, and Exporting Captures

Not every packet capture is the perfect size or representative of the data you need to analyze. Whether you have captured the traffic yourself or someone has sent a file to you to examine, some packet captures are large and cumbersome. Many times, large files have to be subdivided into smaller files for effective analysis. In addition, when done, you will most likely need to save the file, or in some cases, export the capture into a specific format.

In this chapter, we'll cover several methods and techniques that we can use to work with packet captures. So that you can reduce a large file to a more manageable size, we'll look at filtering the capture to narrow down the results. You'll learn how versatile Wireshark is in exporting different components of a capture. Finally, you'll see how you can export files, along with specified packets, packet dissections, and objects. In addition, you'll discover reasons and ways to add comments to a single packet or an entire capture.

This chapter will address all of this by covering the following:

- Discovering ways to subset traffic
- Understanding options to save a file
- Recognizing ways to export components
- Identifying why and how to add comments

## Discovering ways to subset traffic

Packet analysis is used for a variety of reasons, including troubleshooting, testing, monitoring, and baselining the network. Although we can reduce the file size before we begin our analysis by using a capture filter, many times we gather all the traffic so that we don't miss any important details. Then, once captured, the file can be shared with other members of the team for further analysis or to point out specific issues.

While capturing traffic, it's optimal to get a capture that is the perfect size and includes only the troublesome packets. However, that is not always the case. At times, you may find you have to work with a large file, for a variety of reasons that include the following:

- You have obtained the capture from a network device with a large amount of traffic. Tapping into the network, even for a short time, can generate a huge number of packets. Even if you used a capture filter while obtaining the file, you may still end up with a large amount of data.
- You have received a file from someone with good intentions, who felt a large capture would help your analysis. For example, you have received a large file from a co-worker that captured traffic off of the server and they need your help in analyzing a specific problem.

Whatever method you've used to obtain the capture, you'll need to work with it in Wireshark. Keep in mind that while Wireshark can load a large file, it can be very resource-intensive and slow in responding. This is because Wireshark will attempt to dissect all the protocols before displaying the capture. In addition, when you apply a display filter to a large capture, it will take a while to filter the traffic. As a result, the best option is to subset the capture and focus on the problem areas.

When we subset traffic, we break it down into smaller files for analysis. There are many ways to break down or subset traffic, which include subsetting by IP address, port number, protocol, or a specific stream.

Together, we can examine ways of breaking apart a large file using `bigFlows.pcap`, located here: <http://tcpplay.appneta.com/wiki/captures.html#bigflows-pcap>. Once there, download the file and open it in Wireshark so you can follow along.

When you open `bigFlows.pcap`, you can easily see how cumbersome it is to work with a large file, as this capture has 791,615 packets. For example, even when entering a simple filter to display only **Transmission Control Protocol (TCP)** traffic, it will take time for Wireshark to rescan the capture and present the data. Wireshark has a status bar that indicates the process while rescanning the capture, as shown on the lower left-hand side of the following screenshot:

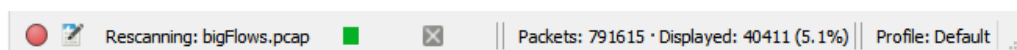


Figure 18.1 – Rescanning the capture

Depending on the system used to analyze the capture, it may run very slowly, freeze up, or even shut down Wireshark.

Once you have opened the file, you'll need to plan what data you want to subset. There are many ways to achieve this, and it really depends on what you want to analyze. In this section, we'll look at a few ways to break down a large capture. First, we'll examine how to use an **Internet Protocol (IP)** address to subset traffic.

## Dissecting by an IP address

One way to break down a large capture is by filtering a specific IPv4 or IPv6 address and then using the subset for your analysis. For example, you suspect that a specific host is causing excessive bursty traffic. By homing in on the activity of a specific IP address, you can better troubleshoot the issue.

Let's review how to narrow your search. In any large capture, you will most likely have gathered many IP addresses. Go to the bottom of the **Statistics** menu, where you will see menu choices for both **IPv4 Statistics** and **IPv6 Statistics**, as shown in the following screenshot:

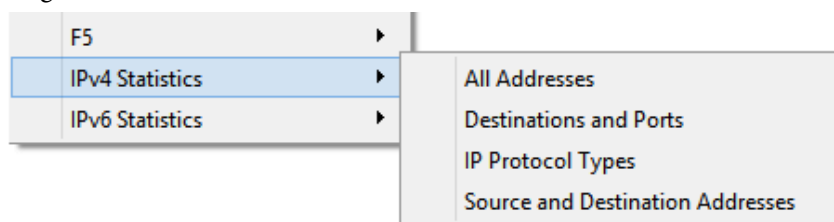
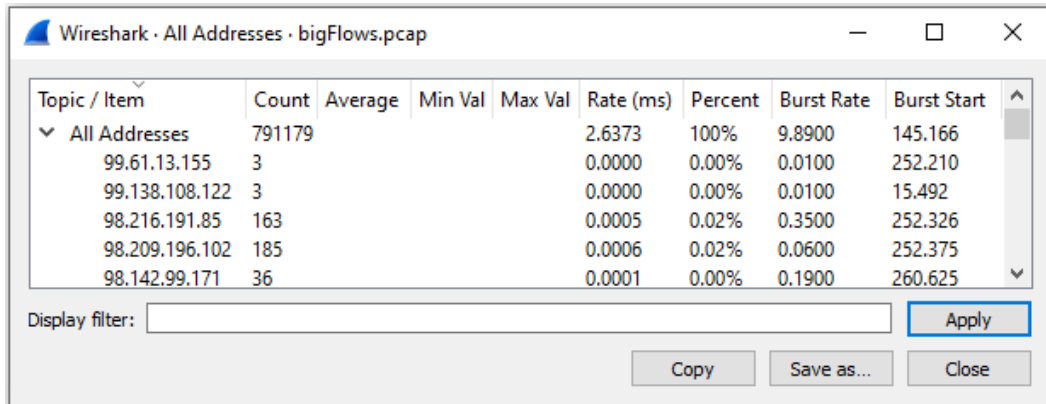


Figure 18.2 – IPv4 and IPv6 statistics

The IP statistics have four choices for either IPv4 or IPv6, which include the following:

- **All Addresses:** Provides a sortable list of IP addresses, as shown:



Wireshark · All Addresses · bigFlows.pcap

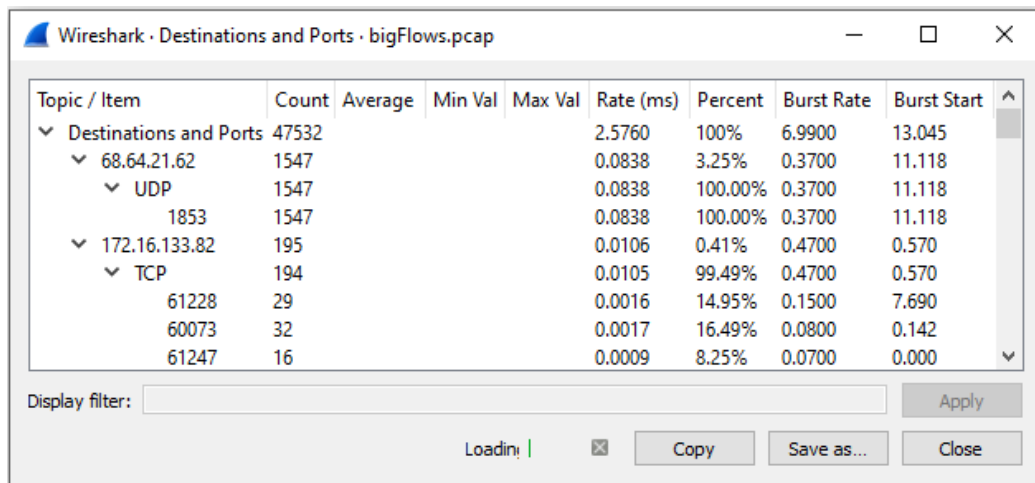
Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
▼ All Addresses	791179				2.6373	100%	9.8900	145.166
99.61.13.155	3				0.0000	0.00%	0.0100	252.210
99.138.108.122	3				0.0000	0.00%	0.0100	15.492
98.216.191.85	163				0.0005	0.02%	0.3500	252.326
98.209.196.102	185				0.0006	0.02%	0.0600	252.375
98.142.99.171	36				0.0001	0.00%	0.1900	260.625

Display filter:  Apply

Copy Save as... Close

Figure 18.3 – Statistics: All Addresses

- **Destinations and Ports:** This report shows a detailed list that breaks down each IP address with additional statistics on TCP and User Datagram Protocol (UDP), as displayed in this screenshot:



Wireshark · Destinations and Ports · bigFlows.pcap

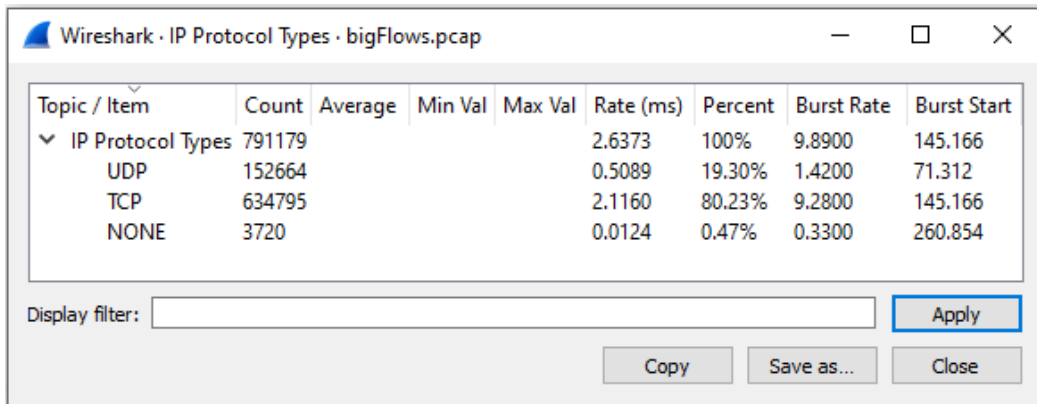
Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
▼ Destinations and Ports	47532				2.5760	100%	6.9900	13.045
▼ 68.64.21.62	1547				0.0838	3.25%	0.3700	11.118
▼ UDP	1547				0.0838	100.00%	0.3700	11.118
1853	1547				0.0838	100.00%	0.3700	11.118
▼ 172.16.133.82	195				0.0106	0.41%	0.4700	0.570
▼ TCP	194				0.0105	99.49%	0.4700	0.570
61228	29				0.0016	14.95%	0.1500	7.690
60073	32				0.0017	16.49%	0.0800	0.142
61247	16				0.0009	8.25%	0.0700	0.000

Display filter:  Apply

Loading | ☒ Copy Save as... Close

Figure 18.4 – Statistics: Destinations and Ports

- **IP Protocol Types:** This report breaks down by protocols that follow the IP header, which can be TCP, UDP, or NONE, as shown:



Wireshark - IP Protocol Types - bigFlows.pcap

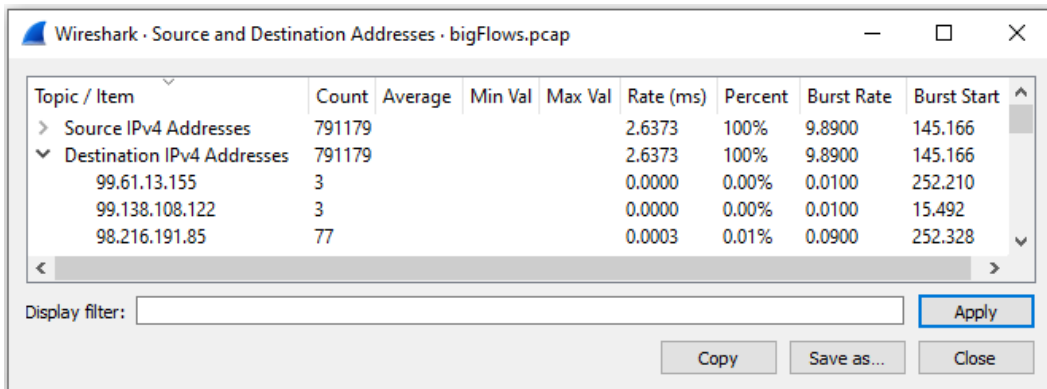
Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
IP Protocol Types	791179				2.6373	100%	9.8900	145.166
UDP	152664				0.5089	19.30%	1.4200	71.312
TCP	634795				2.1160	80.23%	9.2800	145.166
NONE	3720				0.0124	0.47%	0.3300	260.854

Display filter:

Buttons: Apply, Copy, Save as..., Close

Figure 18.5 – Statistics: IP Protocol Types

- **Source and Destination Addresses:** This breaks down all addresses by source and destination IP addresses.



Wireshark - Source and Destination Addresses - bigFlows.pcap

Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
Source IPv4 Addresses	791179				2.6373	100%	9.8900	145.166
Destination IPv4 Addresses	791179				2.6373	100%	9.8900	145.166
99.61.13.155	3				0.0000	0.00%	0.0100	252.210
99.138.108.122	3				0.0000	0.00%	0.0100	15.492
98.216.191.85	77				0.0003	0.01%	0.0900	252.328

Display filter:

Buttons: Apply, Copy, Save as..., Close

Figure 18.6 – Statistics: IP Protocol Types



All filters have additional information, such as **Count** and **Burst Rate**, along with the ability to filter and sort. In addition, you can select **Save as...** to save any of the statistics in a variety of file types, as shown here:

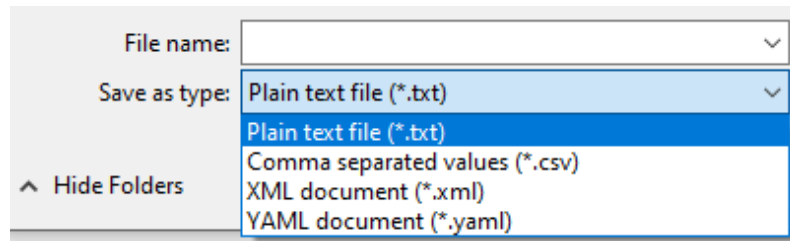


Figure 18.7 – Statistics: Save as

While subsetting traffic by IP addresses may be helpful to home in on troublesome hosts, another way to break down a large capture is by using conversations.

## Narrowing down by conversations

A conversation is two endpoints communicating with one another. In a large capture, you will most likely have many conversations. To see a list, go to the **Statistics** menu and select **Conversations**, as shown:

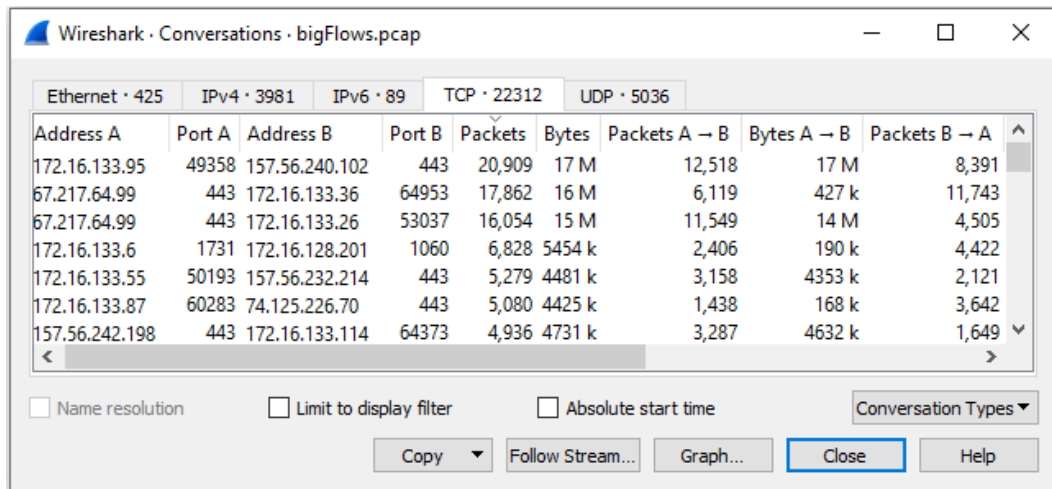


Figure 18.8 – Displaying all conversations

Once in the **Conversations** dialog box, we can sort by columns to identify top talkers, which are the two endpoints that are exchanging the most data. We can also select a conversation between two known endpoints, such as a **Voice over IP (VoIP)** client and server exchanging data. Once identified, we can use the data to create graphs and flow charts for analysis.

Within the window, you will see tabs along the top that allow you to view a specific type of conversation, such as **Ethernet**, **IPv4**, **TCP**, and **UDP**. Select a conversation, such as the one between 172.16.133.95 and 157.56.240.102. Once selected, you can filter the results so that you only see the traffic you want to use as your subset, as shown in the following screenshot:

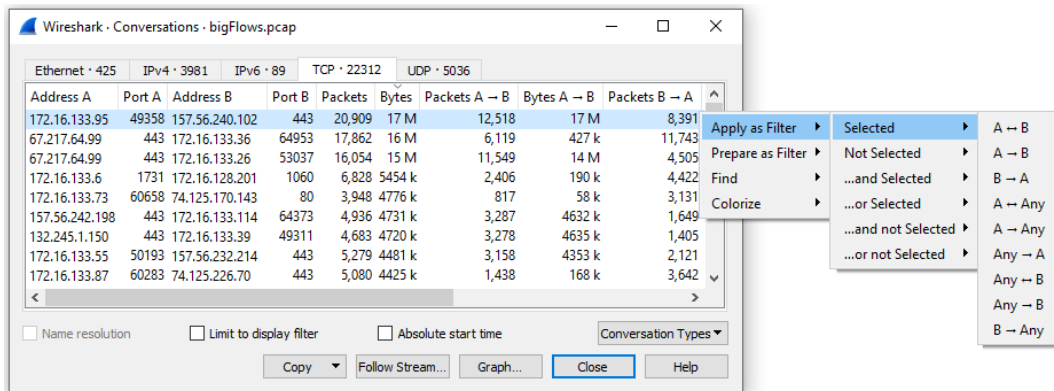


Figure 18.9 – Conversations: Filter options

Within one capture, there may be many conversations. Although you may find that subsetting a large capture by conversations is helpful, sometimes, you may want to zero in on a specific *port* and use that as your subset. The following section illustrates how you can filter by port numbers, so you can work with the resulting smaller file.

## Minimizing by port number

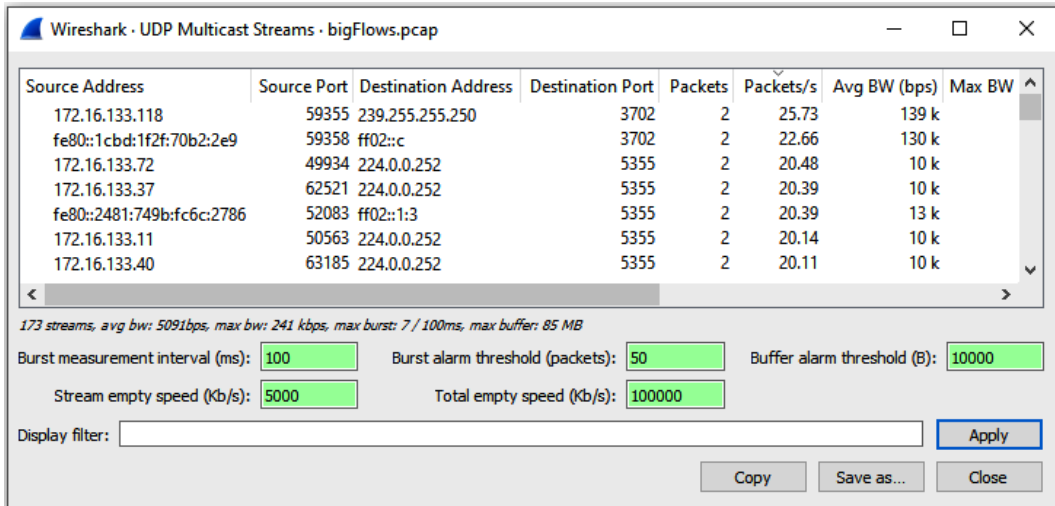
While subsetting by IP address or conversation may be helpful, sometimes you may want to study a specific port. You might be looking through the conversation under either the **TCP** or **UDP** tab and identify suspicious port usage. Or you may want to further investigate a specific port used in a multicast stream when checking for bursty traffic.

There are many reasons to subset by port numbers. In Wireshark, you can find a list of UDP/TCP ports in a few areas, which include the following:

- **Conversations**
- **Endpoints**

- **IPv4 or IPv6 Destinations and Ports**
- **UDP Multicast Streams**

For example, return to `bigFlows.pcap`, then go to **Statistics** and then **UDP Multicast Streams**, as shown:



Wireshark · UDP Multicast Streams · bigFlows.pcap

Source Address	Source Port	Destination Address	Destination Port	Packets	Packets/s	Avg BW (bps)	Max BW
172.16.133.118	59355	239.255.255.250	3702	2	25.73	139 k	
fe80::1cbd:1f2f:70b2:2e9	59358	ff02::c	3702	2	22.66	130 k	
172.16.133.72	49934	224.0.0.252	5355	2	20.48	10 k	
172.16.133.37	62521	224.0.0.252	5355	2	20.39	10 k	
fe80::2481:749b:fc6c:2786	52083	ff02::1:3	5355	2	20.39	13 k	
172.16.133.11	50563	224.0.0.252	5355	2	20.14	10 k	
172.16.133.40	63185	224.0.0.252	5355	2	20.11	10 k	

173 streams, avg bw: 5091bps, max bw: 241 kbps, max burst: 7 / 100ms, max buffer: 85 MB

Burst measurement interval (ms):  Burst alarm threshold (packets):  Buffer alarm threshold (B):

Stream empty speed (Kb/s):  Total empty speed (Kb/s):

Display filter:

Figure 18.10 – UDP Multicast Streams

Once you run the report, you can isolate the port you want to analyze, apply a filter, and select only the traffic you want to use as your subset.

Another way to dissect a large capture is by filtering on a specific protocol. Let's take a look.

## Breaking down by protocol

Wireshark is capable of dissecting hundreds of protocols. To see a list, go to **Statistics** and then **Protocol Hierarchy**, which will provide a list of what protocols appear in the capture. As with many other options, within **Protocol Hierarchy Statistics**, you can apply a filter and create your subset, as shown in the following screenshot:

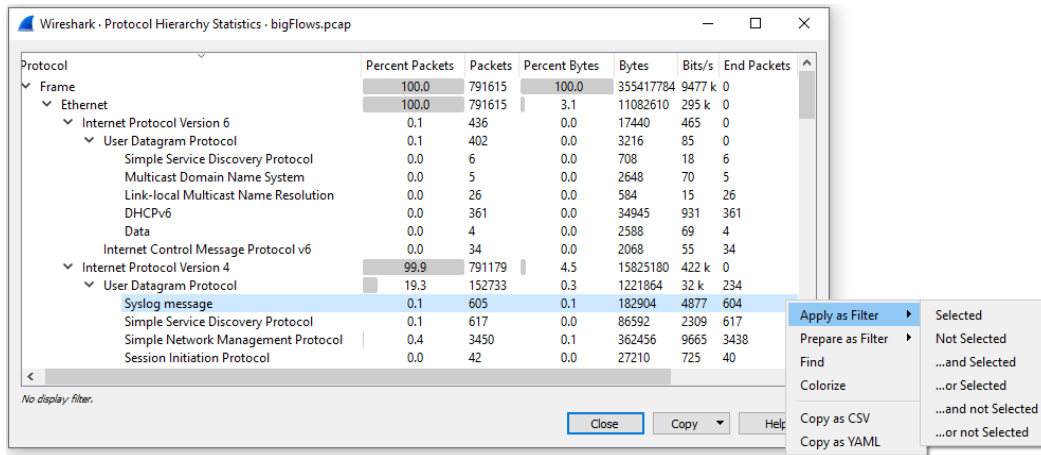


Figure 18.11 – Protocol Hierarchy: Apply as Filter

In addition, if you know the protocol you want to review, you can use a display filter, enter a specific protocol, and use that as your subset.

One of the common ways of analyzing traffic is by examining a particular traffic stream. In the final segment, we will see what elements of a capture we can view by using the follow the stream feature.

## Subsetting by stream

There are times when you may want to see only the details of a single traffic stream. An easy way to do this in Wireshark is to use the follow the stream option.

You must first select either a TCP or UDP conversation, right-click and select **Follow**, and then select the appropriate stream, such as TCP, UDP, TLS, or HTTP.

For our example, we'll use `bigFlows.pcap`. In the display filter, enter `tcp.stream eq 946`. It will take a while to filter. Once complete, you will see the contents of the communication stream, which is a web page, as shown in the following screenshot:

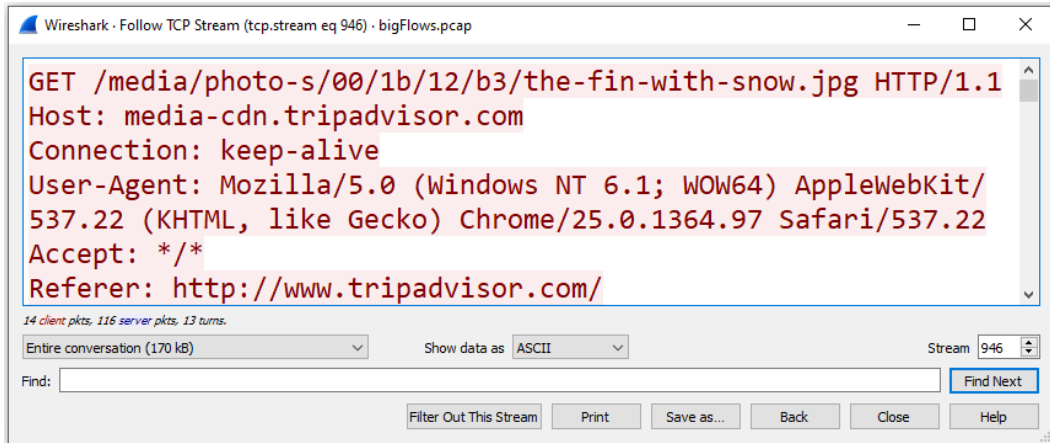


Figure 18.12 – Follow TCP Stream 946

Now that we have reduced the file to a more manageable size by using any of the preceding methods to subset traffic, the next step is to preserve the file in some way. You can save the file in the default `.pcapng` format, or in one of the many other formats that have been added and enhanced over the years.

With Wireshark, there are many ways to subset a file to a more practical size. After you have subsetting the capture, you will most likely want to save the file to preserve your work. The following section provides various ways to save a file in Wireshark.

## Understanding options to save a file

Whenever you run and then stop a capture, Wireshark will hold the capture in a temporary file. Wireshark will display the temporary filename in the **Status** bar, which is found on the lower left-hand side of the interface. In addition, along the top, you will see the name of the interface used in the capture and an asterisk, as shown in the following screenshot:

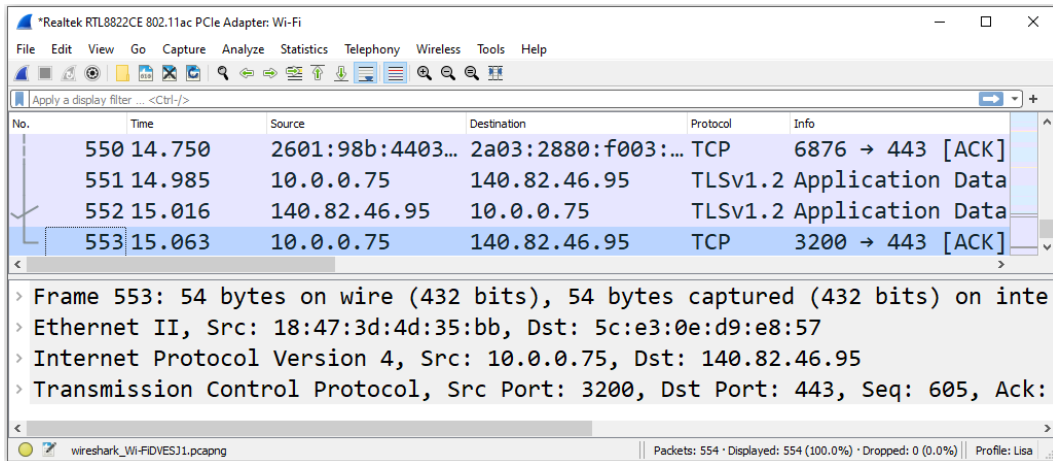


Figure 18.13 – Temporary file in Wireshark

At some point, you will most likely want to save the file. To save the file, go to the **File** menu choice and then click on **Save**. Once you save the file, the filename will appear along the top of the Wireshark interface, as shown in the following screenshot:

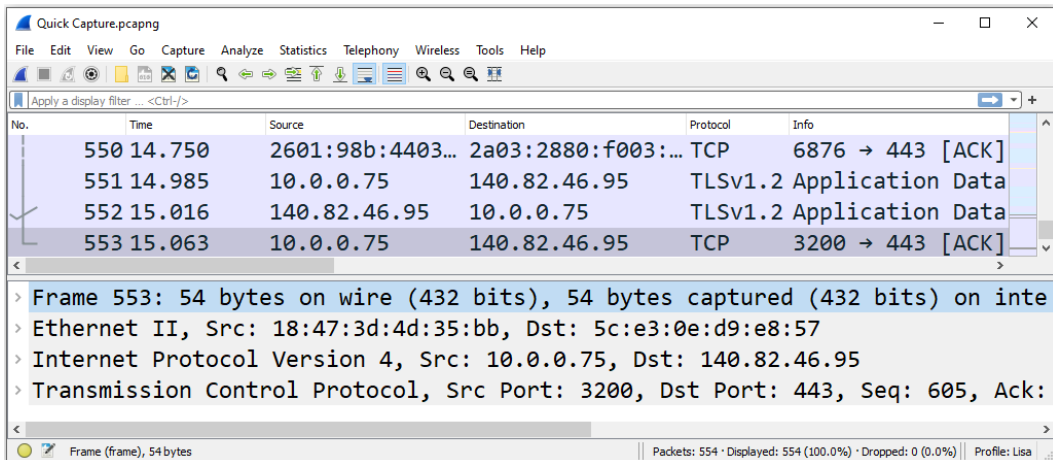


Figure 18.14 – File saved in Wireshark

When you do go to **File** and then **Save**, you will find that Wireshark will allow you to save the capture file in many different formats, as discussed next.

## Using Save as

The **File** menu choice has many common options to work with files, such as **Open**, **Import**, **Save**, **Print**, and **Export**. One option is to use **Save as** when you need to save the file as something other than the default extension, which is `.pcapng`.

When you are ready to save the file, go to the **File** menu choice and select **Save as**, which will open a dialog box, as shown in the following screenshot:

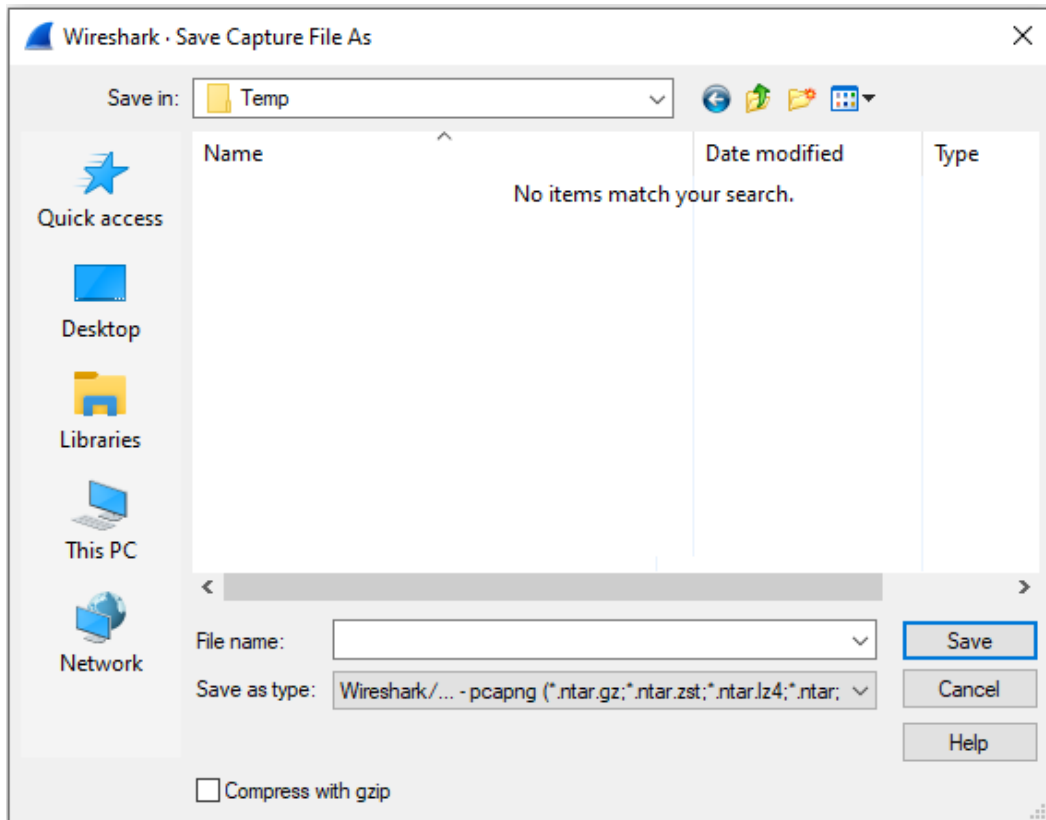
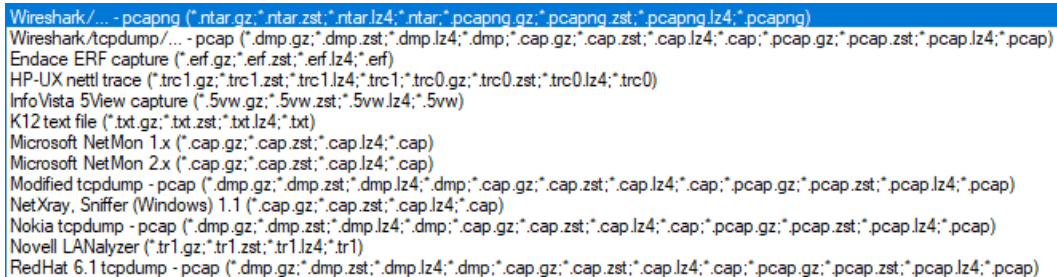


Figure 18.15 – The Save Capture File As dialog box

Over the years, developers have added many different file formats to Wireshark. As a result, when you click on the **Save as type** dropdown, you will see a list of all the supported file formats. The following screenshot shows a partial list of the formats available:



```

Wireshark/... - pcapng (*.ntar.gz;*.ntar.zst;*.ntar.lzf;*.ntar.pcapng.gz;*.pcapng.zst;*.pcapng.lzf;*.pcapng)
Wireshark/tcpdump/... - pcap (*.dmp.gz;*.dmp.zst;*.dmp.lzf;*.dmp;*.cap.gz;*.cap.zst;*.cap.lzf;*.cap;*.pcap.gz;*.pcap.zst;*.pcap.lzf;*.pcap)
Endace ERF capture (*.erf.gz;*.erf.zst;*.erf.lzf;*.erf)
HP-UX nettl trace (*.trc1.gz;*.trc1.zst;*.trc1.lzf;*.trc1;*.trc0.gz;*.trc0.zst;*.trc0.lzf;*.trc0)
InfoVista 5View capture (*.5vw.gz;*.5vw.zst;*.5vw.lzf;*.5vw)
K12 text file (*.txt.gz;*.txt.zst;*.txt.lzf;*.txt)
Microsoft NetMon 1.x (*.cap.gz;*.cap.zst;*.cap.lzf;*.cap)
Microsoft NetMon 2.x (*.cap.gz;*.cap.zst;*.cap.lzf;*.cap)
Modified tcpdump - pcap (*.dmp.gz;*.dmp.zst;*.dmp.lzf;*.dmp;*.cap.gz;*.cap.zst;*.cap.lzf;*.cap;*.pcap.gz;*.pcap.zst;*.pcap.lzf;*.pcap)
NetXray, Sniffer (Windows) 1.1 (*.cap.gz;*.cap.zst;*.cap.lzf;*.cap)
Nokia tcpdump - pcap (*.dmp.gz;*.dmp.zst;*.dmp.lzf;*.dmp;*.cap.gz;*.cap.zst;*.cap.lzf;*.cap;*.pcap.gz;*.pcap.zst;*.pcap.lzf;*.pcap)
Novell LANalyzer (*.tr1.gz;*.tr1.zst;*.tr1.lzf;*.tr1)
RedHat 6.1 tcpdump - pcap (*.dmp.gz;*.dmp.zst;*.dmp.lzf;*.dmp;*.cap.gz;*.cap.zst;*.cap.lzf;*.cap;*.pcap.gz;*.pcap.zst;*.pcap.lzf;*.pcap)

```

Figure 18.16 – Partial list of Save as selections

Some of the formats available include Microsoft NetMon ( . cap), Novell LANalyzer ( . tr1), Sniffer (Windows: . cap), and K12 text file ( . txt). While some of the formats are legacy and might not ever be used, it's nice to know you have a variety of options.

One common use of the **Save as** menu choice is to open a file in one format and then save it in another format. An example is obtaining a file with a . pcap extension. Although you can do many things with a . pcap file, that format can be restrictive. For example, when analyzing a file in the . pcap format, you cannot save any comments. If you do add any comments, when you close the file, Wireshark will prompt you to save as . pcapng if you want to preserve your comments.

Therefore, in most cases, saving as a . pcapng is a better option, as this format has several enhancements and is able to dissect and display payloads better.

While saving an entire file is common, you may want to export only a portion of the file. The next section covers the various ways to export specific packets, along with various objects found within the capture file, such as images or web pages.

## Recognizing ways to export components

We discussed the many ways you can subset a capture to reduce the file to a more practical size, such as by IP address, port number, or stream. Another option is to export the subset as specified packets or packet dissections, or even export various objects that exist in the capture.

Let's take a look at the many export options Wireshark offers, starting with specified packets.



## Selecting specified packets

After you have filtered a capture, you may want to export a portion of the capture. With Wireshark, you can be very specific in what you select to export. Let's step through an example.

Return to the `bigFlows.pcap` capture and enter `tcp.stream eq 946` in the display filter. Once you have run the filter, you are ready to preserve this subset. In this case, we will go to the **File** menu choice and then select **Export Specified Packets**. Once open, you will see that you have several ways to export file components, as shown in the following screenshot:

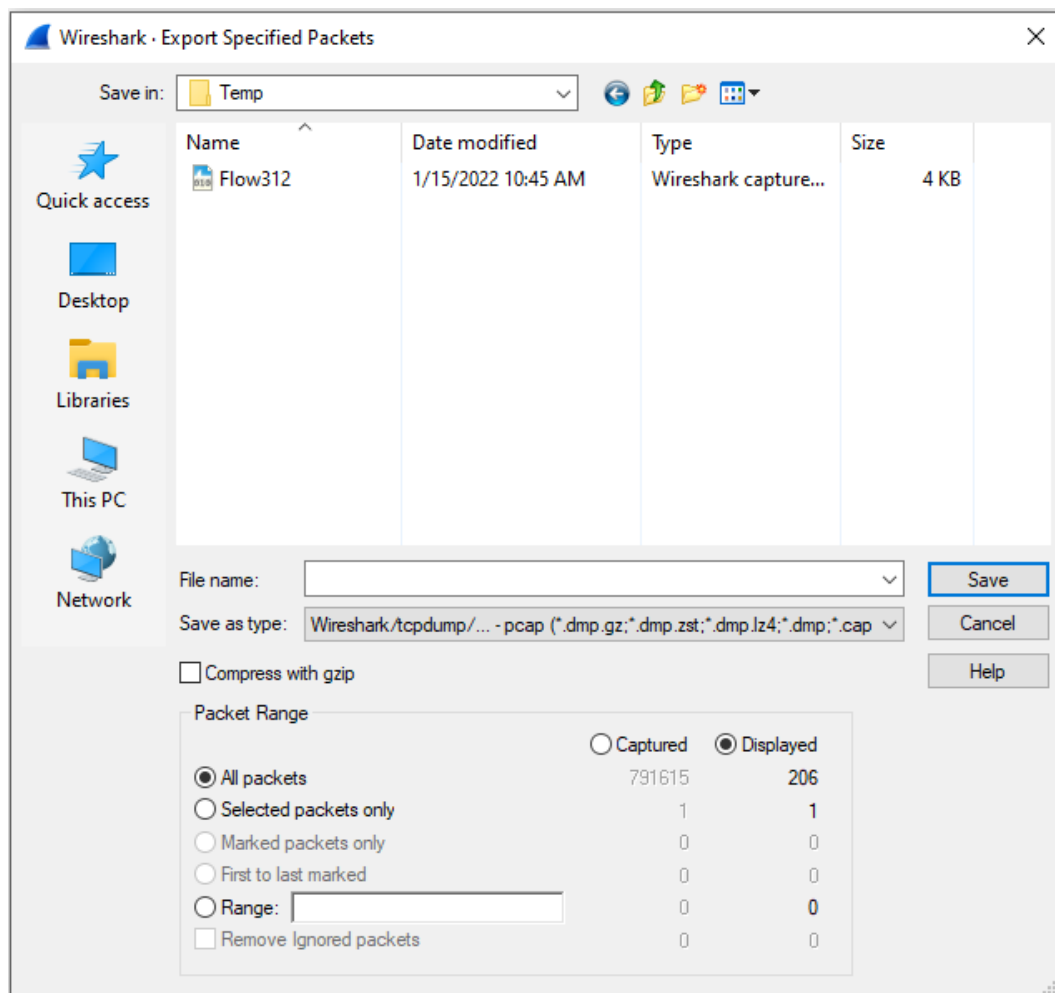


Figure 18.17 – Export Specified Packets

Near the bottom of the dialog box, you will see a header named **Packet Range**, where you will make your selections. If you have filtered the capture, Wireshark will assume you would like to export *only* the displayed packets, and the radio button for **Displayed** will be active. However, if you want to export *all* the packets, select **Captured**.

Below that, you will see other choices for the packet range you would like. The options include the following:

- **All packets:** This will export all packets. Wireshark will show how many are either **Captured** or **Displayed**.
- **Selected packets only:** This will only export the packet selected. In most cases, you will have placed your cursor on one of the packets, so Wireshark will assume you have selected that packet. That is why in *Figure 18.17*, Wireshark shows one (1) packet in the **Selected packets only** option.
- **Marked packets only:** Marking packets allows you to right-click and mark a specified packet or packets of interest, causing the packet(s) to turn black. When selecting this option, Wireshark will only process marked packets.
- **First to last marked:** If you have marked several packets in your capture, Wireshark will export all marked packets, from the first to the last.
- **Range:** This will allow you to specify a packet range. For example, if you enter 233 - 799, Wireshark will only export that range.
- **Remove Ignored packets:** If in a capture you have ignored certain packets (see *Chapter 4, Exploring the Wireshark Interface*, under the *Marking or ignoring packets* section) and you select this option, Wireshark will not include the ignored packet(s) in the export.

TCP stream 946 is a web page retrieved from a travel site, so we'll name the file `Web Page`. When exporting, you will need to force the file format to be `.pcapng`, as Wireshark will default to the original file format, which is `.pcap`, for `bigFlows.pcap`. To export only TCP stream 946, follow these steps:

1. Go to the **File** menu choice and then click on **Export Specified Packets**.
2. Leave the default values as they are, as shown in *Figure 18.17*.
3. Select a location in which to save the file.
4. In **File name**, enter `Web Page`.
5. Under the drop-down menu for **Save as type**, select **Wireshark/...- pcapng**.

Once the export is complete, close `bigFlows.pcap`, and then open the newly created file: `Web Page.pcapng`.

Within the **File** menu choice, we will also find **Export**, which includes several options. The options include the ability to export specific packets or bytes, TLS session keys, and other objects, as outlined next.

## Exporting various objects

When working with a capture, there may be a variety of objects within the file. Wireshark reassembles the objects, which can be collected and analyzed, as long as the object is unencrypted.

There are several reasons you may need to collect objects within a capture file. For example, during an active malware investigation, you may need to see what types of files are being transferred. Or there may be some concern that an individual may be sending sensitive information out of the organization. Wireshark makes it easy to export objects so that you can take a closer look at the type of traffic that is being sent across the network.

Some of the possible objects that can be exported include those identified within the following protocols:

- **Digital Imaging and Communications in Medicine (DICOM)**
- **HyperText Transfer Protocol (HTTP)**
- **Internet Message Format (IMF)**
- **Server Message Block (SMB)**
- **Trivial File Transfer Protocol (TFTP)**

If you suspect that any of the preceding protocols contain objects, you can export them for examination by going to the **File** menu choice and then **Export Objects**, as shown in the following screenshot:

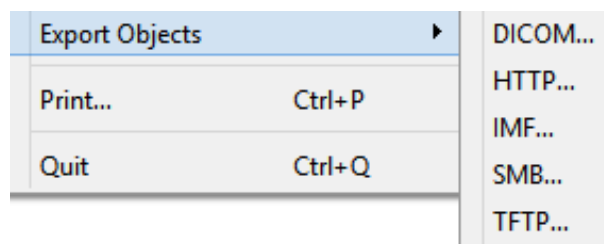


Figure 18.18 – Export Objects

For example, open the `bigFlows.pcap` file. Once open, select **Export Objects** and then **HTTP...** Wireshark will locate all objects, such as **text/css**, **applications/javascript**, images, and **text/html**. This will take a few seconds, depending on the size of the file. Wireshark will then present a list, as shown here:

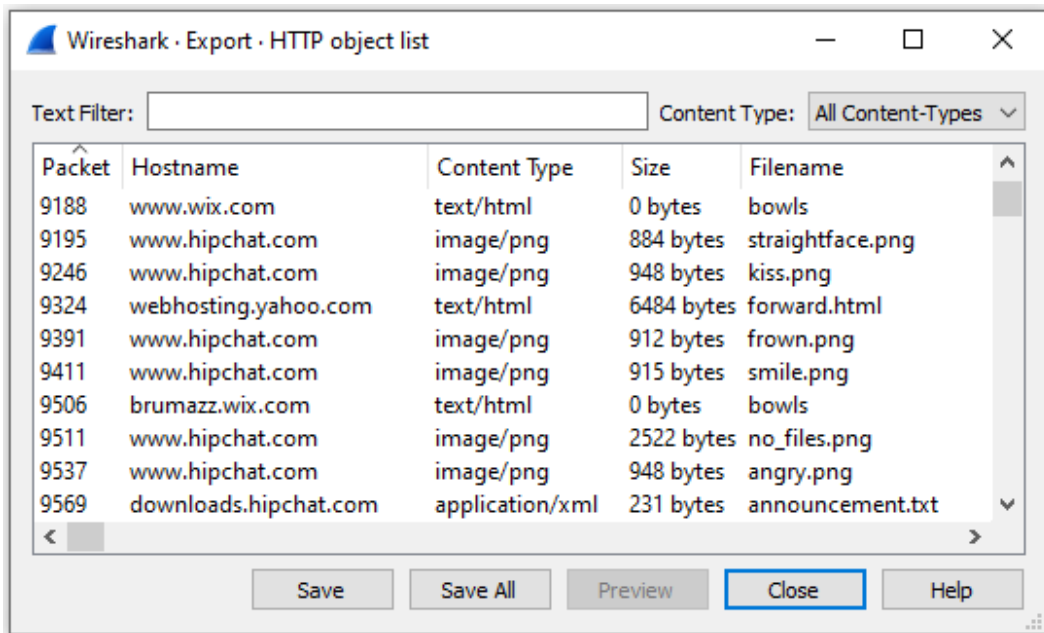


Figure 18.19 – HTTP object list

In the dialog box, you can search for text strings. In the lower left-hand corner, you'll see a **Text Filter** label. Enter `footstesp-to-the-summit.jpg` [sic], exactly as shown in the following screenshot:

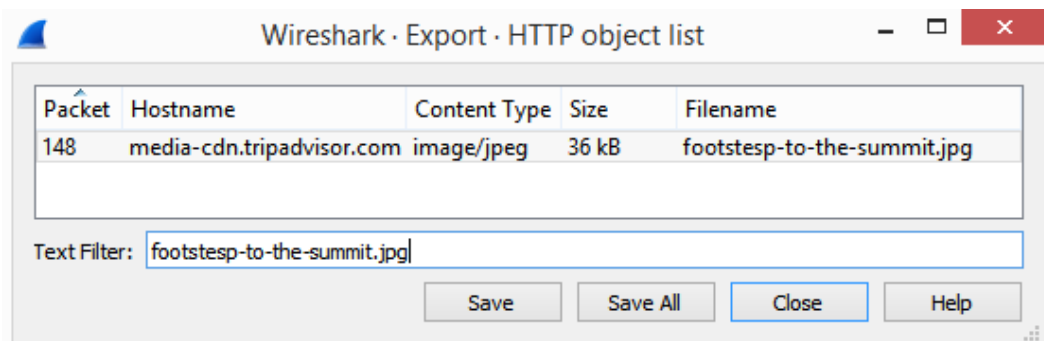


Figure 18.20 – Searching footstesp-to-the-summit.jpg

Select **Save**, and when the dialog box opens, enter the filename and the appropriate extension. In this case, I used `footstesp-to-the-summit.jpg`. After you save the object, you can then locate, open, and view the image.

If there are other objects of interest, you can save them individually as well. Alternatively, you can select **Save All**, and Wireshark will save all objects found in the file.

As evidenced, Wireshark provides many ways to preserve and export components and objects. But what happens when you're done working with a file?

While doing analysis, you may know why you are working on a particular capture. However, when you return to the file, you may not remember what caused you to look at the capture in the first place. In addition, if you share the file with a co-worker, they may not be able to identify why the file was significant. In either case, it's best to identify key elements and concerns by adding comments.

To preserve the reasons why the file was important, Wireshark provides ways to add comments, as discussed in the following section.

## Identifying why and how to add comments

When working with trace files, you might need to make a note on a single packet or the entire capture, for future reference.

Wireshark has options when working with comments. You can add comments to preserve the details of the entire capture or a single packet, as outlined next.

## Providing file and packet comments

Within Wireshark, you can comment on the entire capture to document what you found within the file. You might preserve this information, either for yourself or to share with others when working with a team. Let's walk through an example of adding a comment using the `Web Page.pcapng` subset.

To add a comment to the file, you can do one of the following:

- Select the comments icon in the lower left-hand corner, which looks like a pad and pencil.
- Go to **Statistics | Capture File Properties** and include your comments in the space below **Capture file comments**.

For example, in my `Web Page.pcapng` file, I went to **Statistics | Capture File Properties** and entered the comment `HTTP traffic with interesting images`. I then clicked **Save Comments**, as shown in the following screenshot:

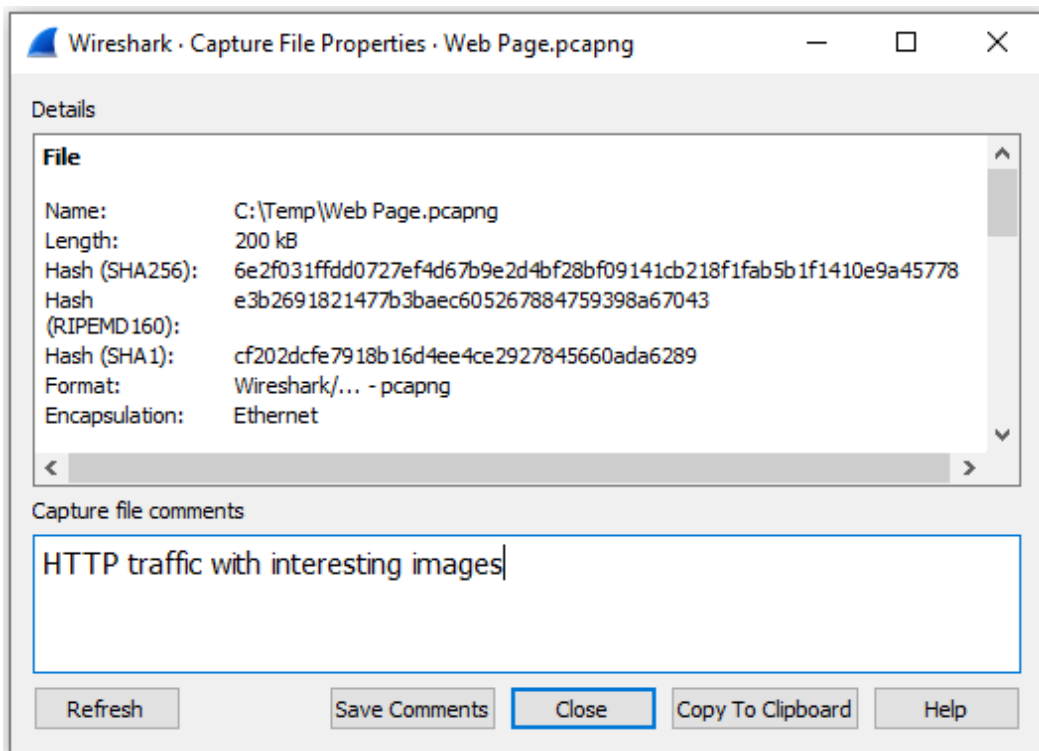


Figure 18.21 – Capture file comments

Keep in mind that when adding comments, Wireshark does not highlight spelling errors. Therefore, if you want the comments in your file to look professional, take the time to check your spelling.

Adding a comment to an entire file is handy. However, sometimes you may want to preserve the details of one or more packets within the file that you found to be interesting. Adding a comment to a single packet is similar to adding a comment to the entire file. However, while in a *single packet*, go to the **Edit** menu choice and select **Packet Comments**, as shown in the following screenshot:

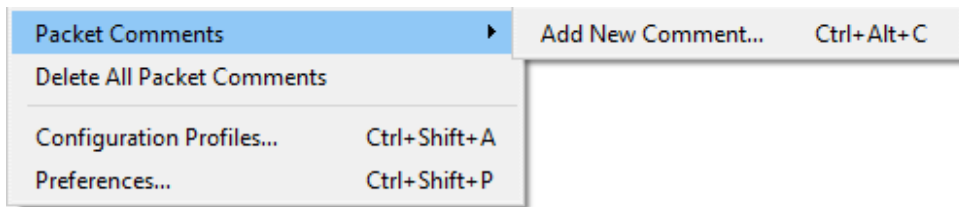


Figure 18.22 – Packet Comments selection

Wireshark will open a form where you can add your comment. If you would like to add more comments later, simply select the same packet and repeat the steps you took to add the original comment.

In addition, you can delete all packet comments by going to the **Edit** menu choice and selecting **Delete all Packet Comments**, as shown in *Figure 18.22*.

After adding or editing comments, you'll need to save them so you can view them later, as discussed next.

## Saving and viewing comments

Once you are done adding comments, either on the entire file or a single packet, you'll see that the filename has an asterisk *in front* of the name, as shown along the top of the Wireshark interface:

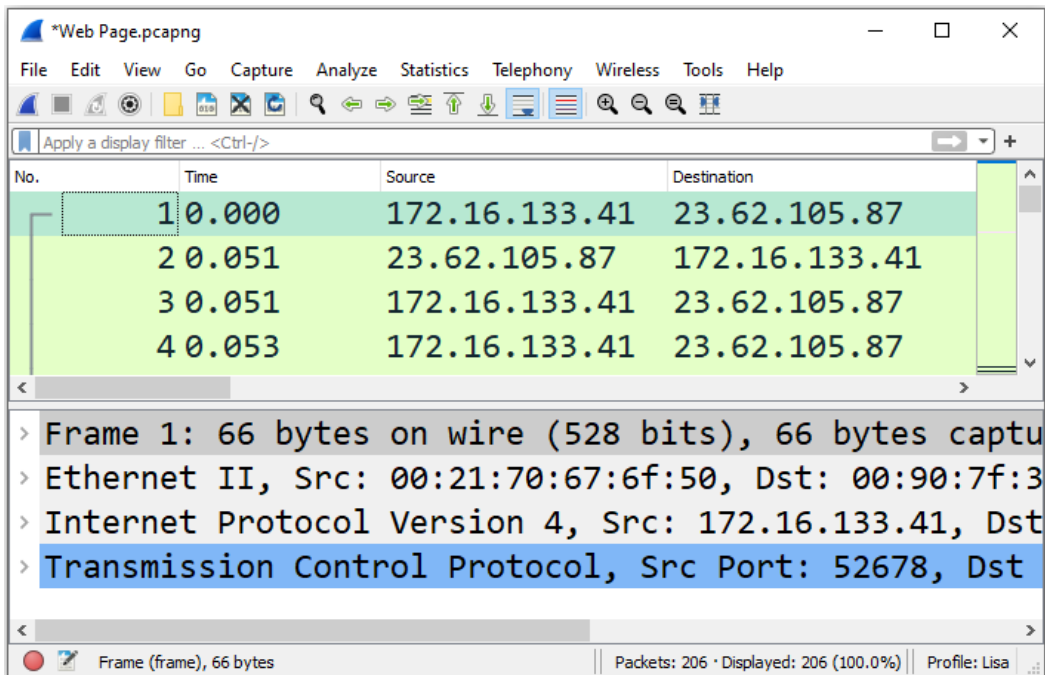


Figure 18.23 – Filename with an asterisk

The asterisk serves as a reminder that you have modified the capture. When you close the capture, Wireshark will prompt you to save the modified file. It's important to note that you must save in .pcapng format when using comments.

Once preserved, there are several ways to view the comments:

- To see comments on a *file*, go to **Statistics | Capture File Properties**. This will open a dialog box, as shown in *Figure 18.21*.



- To see comments on *individual packets*, go to **Expert Information** and select **Show Comments**, which is on the lower right-hand side of the console. You will then see the comments listed, as shown here:

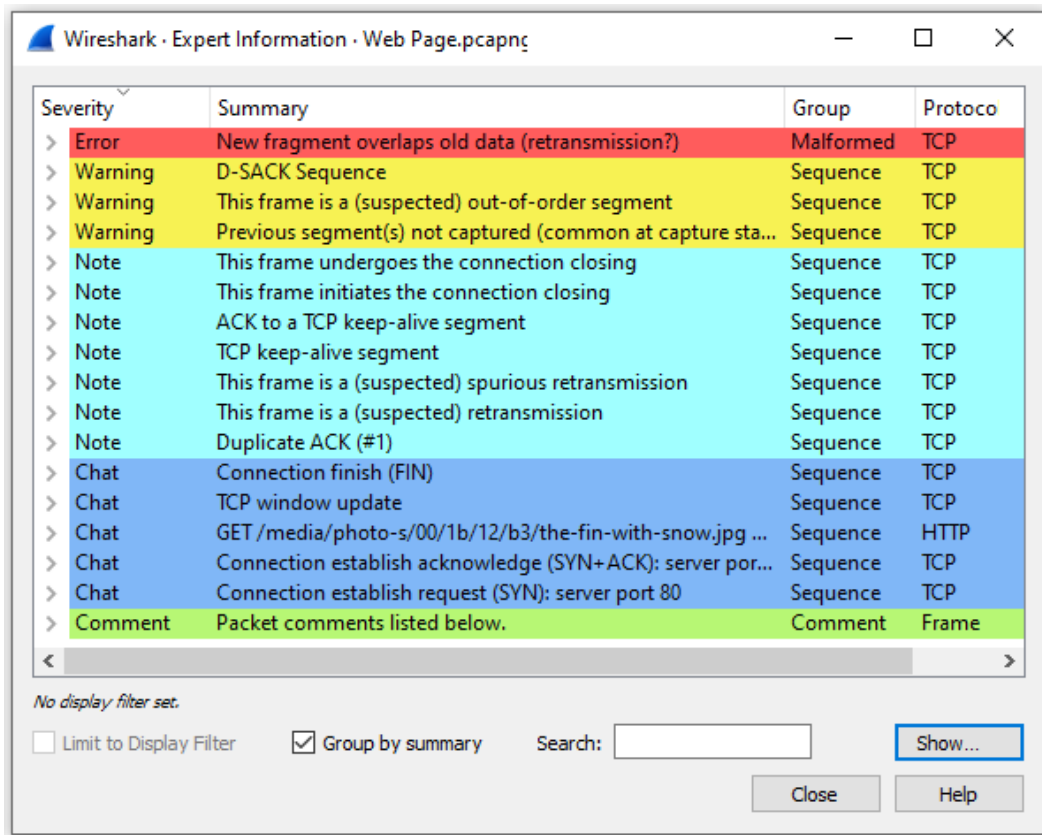


Figure 18.24 – Expert Information: show comments

As evidenced, it's easy to add comments to an entire capture or a single packet so that you or your team can view the comments at a later date.

## Summary

In this chapter, we discovered how you can take a large, unmanageable file and turn it into a smaller, more manageable file. Once reduced, we can then share the file with co-workers or preserve the capture for future reference. You learned about the many ways to subset traffic, which includes filtering by IP address, conversation, port number, or stream. We discovered that, after working with a packet capture, there are many options and formats available in Wireshark to preserve the capture. In addition, you now know about the many ways to export files, objects, session keys, and packet bytes. Finally, in order to preserve the reasons why the file was important, we discovered how we can add comments to a single packet or an entire capture.

In the next chapter, we'll first discover the many ways the **Statistics** menu can help us when analyzing a capture file. We'll also learn how to create basic I/O graphs to help visualize network issues, such as dropped connections, lost frames, and duplicate acknowledgments. We'll then look at ways we can modify the settings along with other options when working with an I/O graph. We'll summarize by comparing how the different TCP Stream Graphs can provide a visual representation of the streams.

## Questions

Now, it's time to check your knowledge. Select the best response, and then check your answers with those in the *Assessment* appendix:

1. A \_\_\_\_\_ in Wireshark represents two endpoints that are communicating with each other.
  - A. Match point
  - B. Tuple
  - C. Conversation
  - D. Filter
2. Wireshark is capable of dissecting hundreds of protocols. To see a list of all of the protocols present in a given capture, go to **Statistics** and then \_\_\_\_\_.
  - A. **Protocol Hierarchy**
  - B. **Conversations**
  - C. **IPv4 Statistics**
  - D. **Match point**

3. Currently, when you save a file, the default file format in Wireshark is \_\_\_\_\_.
  - A. `snoop.gz`
  - B. `.pcapng`
  - C. `.pcap`
  - D. `erf.gz`
4. When working with packets, right-click on a specified packet or packets of interest and select \_\_\_\_\_, which will turn the selected packet(s) black.
  - A. **Ignore**
  - B. **Snoop**
  - C. **Spatter**
  - D. **Mark**
5. When you select **Export Objects** \_\_\_\_\_, Wireshark will locate and include all objects that include **applications/javascript**, images, and **text/html**, and then display a list of the objects found.
  - A. **DNS**
  - B. **DICOM**
  - C. **HTTP**
  - D. **SMB**
6. Prior to saving a capture you have run, along the top of the interface, you will see the name of the interface used in the capture and a(n) \_\_\_\_\_.
  - A. `.temp` extension
  - B. Pencil icon
  - C. Em dash
  - D. Asterisk

7. To add a comment to an entire capture file, you can go to the \_\_\_\_\_ menu choice and then **Capture File Properties** and include your comments in the space below **Capture file comments**.
- A. **Statistics**
  - B. **View**
  - C. **Edit**
  - D. **File**



# 19

## Discovering I/O and Stream Graphs

Within Wireshark, there are numerous tools that help the analyst understand what is happening on the network. In this chapter, we'll first review some of the key elements found in the **Statistics** menu. So that you are aware of the many choices of built-in reports, we'll cover some general information and ways to assess the data found within a capture. We'll also discover several reports that help assess protocol effectiveness, along with a survey of basic graphs found in the **Statistics** menu.

We'll then focus on two main ways to visualize traffic, by using **input/output (I/O)** and **Transmission Control Protocol (TCP)** stream graphs. We'll explore how to create basic I/O graphs to help visualize network issues such as dropped connections, lost frames, and duplicate **acknowledgments (ACKs)**. We'll then compare the four types of TCP stream graphs and learn how each of the graphs can lend insight into the health of the network.

This chapter will cover the following topics:

- Discovering the **Statistics** menu
- Creating I/O graphs
- Comparing TCP stream graphs

## Discovering the Statistics menu

Within Wireshark, there are several ways to view data. Some of the data can be viewed in a numeric way, and some can provide a visual, in the form of a graphic. The **Statistics** menu has several ways to assess the health of your devices and network.

To view all of the choices, drop down the **Statistics** menu, as shown here:

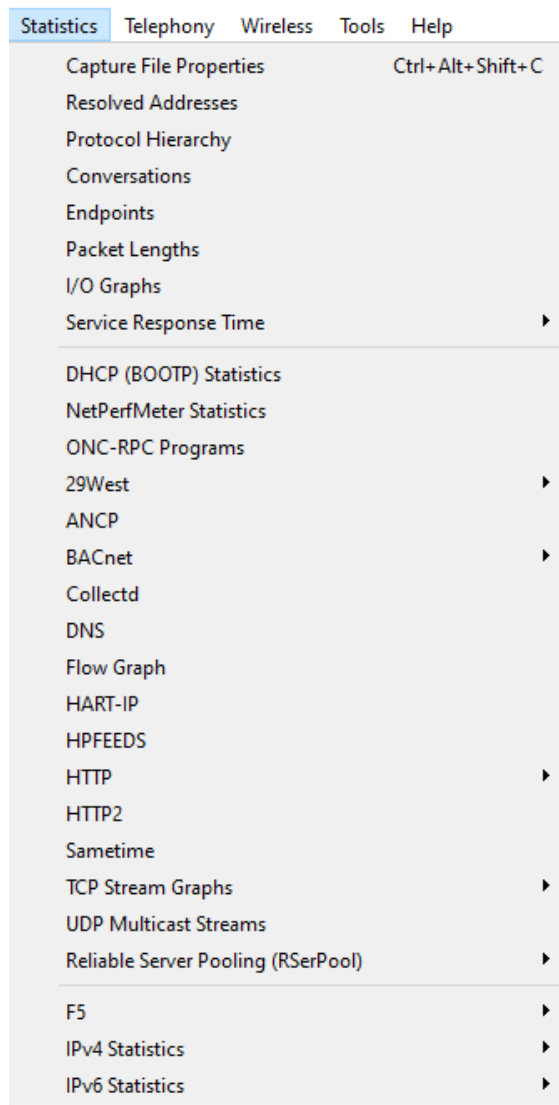


Figure 19.1 – The Statistics menu

In this section, we'll highlight some of the key features of the **Statistics** menu. We'll first take a look at general statistics and then move to ways we can evaluate protocol effectiveness. We'll then summarize by covering the different ways we can graph and visualize issues within a capture.

To follow along with a few examples, we will use `bigFlows.pcap`. To get a copy of the file, go to <http://tcpreplay.appneta.com/wiki/captures.html#bigflows-pcap>. Once there, download and open `bigFlows.pcap` in Wireshark.

## Viewing general information

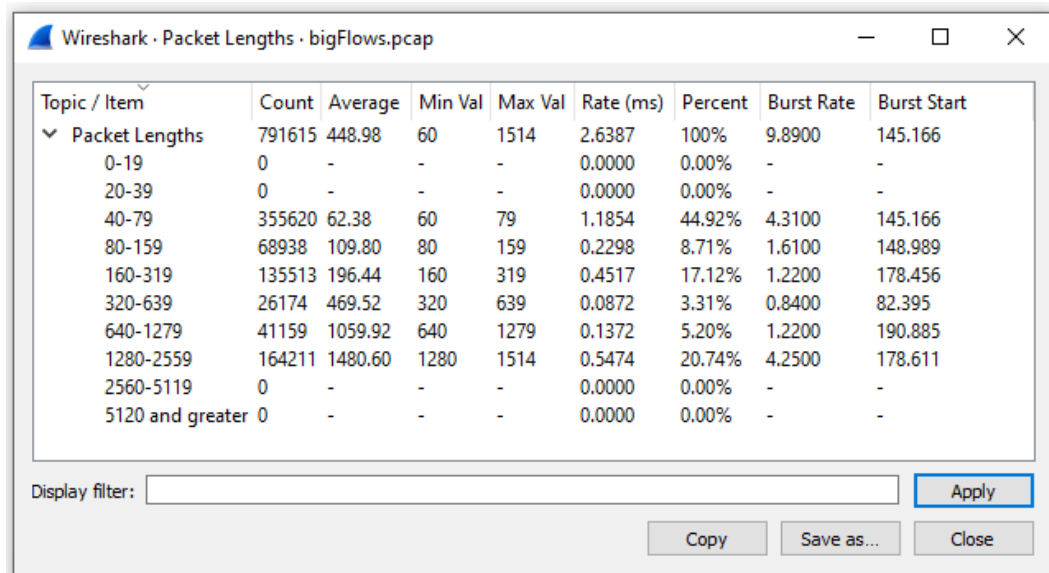
Within the **Statistics** menu, there are several menu choices that provide general information about the capture, protocols, and conversations contained in a file.

Some examples include the following options:

- **Capture File Properties:** This provides general file information, such as packet lengths, size, and time elapsed, along with a section to add comments.
- **Resolved Addresses:** This provides a summary of resolved hostnames and **Internet Protocol (IP)** addresses.
- **Protocol Hierarchy:** This provides a visual of a tiered list of protocols contained in the file.
- **Conversations:** This provides a list of conversations between two endpoints.
- **Endpoints:** This lists all of the endpoints found in the capture.
- **Packet Lengths:** This outlines a list of packet lengths and related information.



Some of the information can provide a snapshot of the traffic on your network. For example, if we go to `bigFlows.pcap` and select **Statistics | Packet Lengths**, Wireshark will generate this report:



The image shows a Wireshark window titled "Wireshark · Packet Lengths · bigFlows.pcap". It displays a table of packet length statistics. The table has columns for Topic / Item, Count, Average, Min Val, Max Val, Rate (ms), Percent, Burst Rate, and Burst Start. The data is organized into a tree view under "Packet Lengths".

Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
Packet Lengths	791615	448.98	60	1514	2.6387	100%	9.8900	145.166
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	355620	62.38	60	79	1.1854	44.92%	4.3100	145.166
80-159	68938	109.80	80	159	0.2298	8.71%	1.6100	148.989
160-319	135513	196.44	160	319	0.4517	17.12%	1.2200	178.456
320-639	26174	469.52	320	639	0.0872	3.31%	0.8400	82.395
640-1279	41159	1059.92	640	1279	0.1372	5.20%	1.2200	190.885
1280-2559	164211	1480.60	1280	1514	0.5474	20.74%	4.2500	178.611
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

At the bottom of the window, there is a "Display filter:" text box, an "Apply" button, and "Copy", "Save as...", and "Close" buttons.

Figure 19.2 – Viewing packet lengths

In addition to being able to view general statistics on traffic flowing across the network, Wireshark has several ways to assess the behavior of specific protocols.

## Assessing protocol effectiveness

Wireshark can dissect hundreds of protocols. In addition, the **Statistics** menu offers a variety of ways to analyze the effectiveness of several protocols.

One way to assess the health of the network is by viewing the **Service Response Time** of the different protocols interacting on the network. This menu choice offers a submenu of multiple protocols, as shown in the following screenshot:

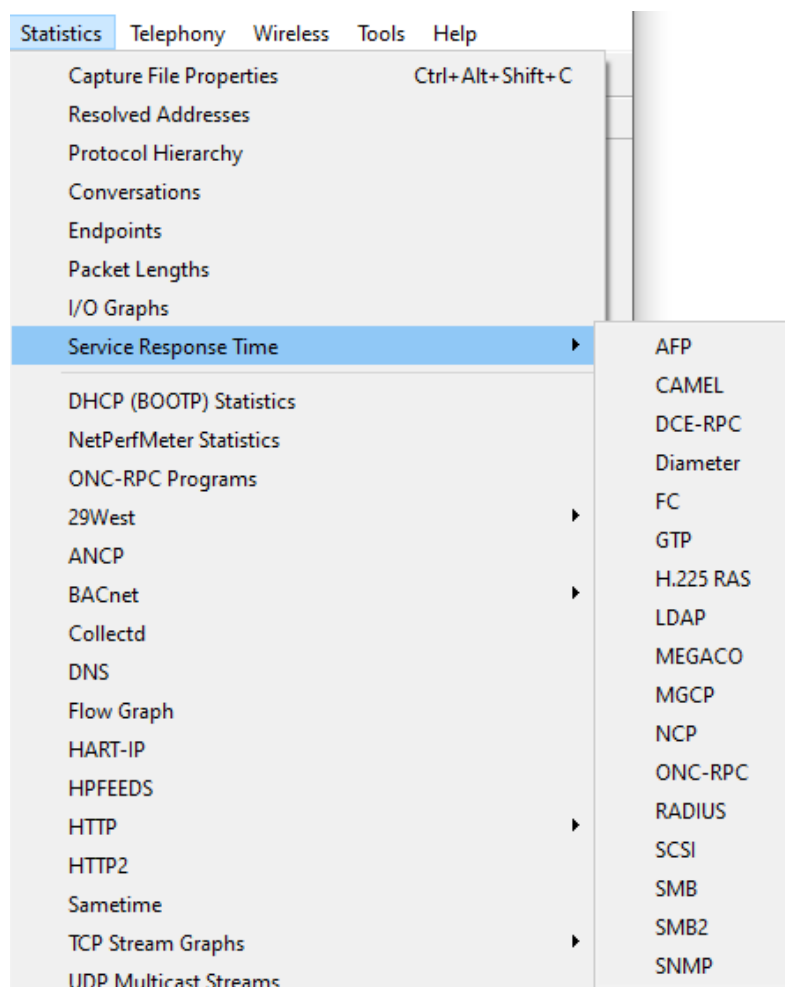
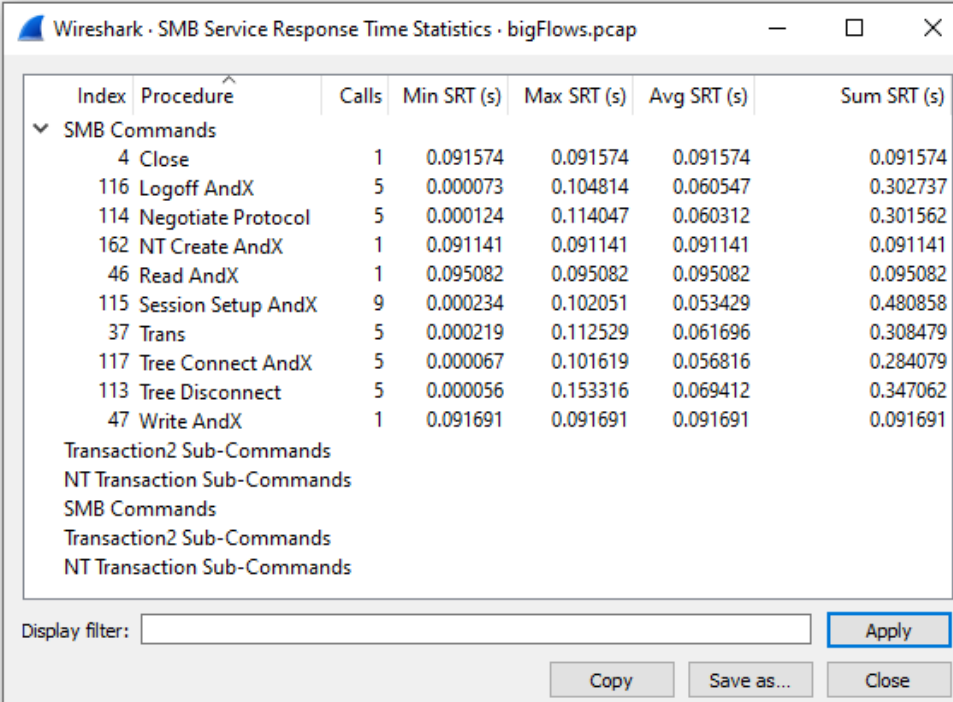


Figure 19.3 – Service Response Time menu

When using this option, Wireshark will evaluate the time between a request and a response for a particular protocol. For example, to generate statistics on **Server Message Block (SMB)**, return to `bigFlows.pcap` and select **Statistics | Service Response Time | SMB**, which will result in the following report:



Wireshark - SMB Service Response Time Statistics - bigFlows.pcap

Index	Procedure	Calls	Min SRT (s)	Max SRT (s)	Avg SRT (s)	Sum SRT (s)
▼ SMB Commands						
4	Close	1	0.091574	0.091574	0.091574	0.091574
116	Logoff AndX	5	0.000073	0.104814	0.060547	0.302737
114	Negotiate Protocol	5	0.000124	0.114047	0.060312	0.301562
162	NT Create AndX	1	0.091141	0.091141	0.091141	0.091141
46	Read AndX	1	0.095082	0.095082	0.095082	0.095082
115	Session Setup AndX	9	0.000234	0.102051	0.053429	0.480858
37	Trans	5	0.000219	0.112529	0.061696	0.308479
117	Tree Connect AndX	5	0.000067	0.101619	0.056816	0.284079
113	Tree Disconnect	5	0.000056	0.153316	0.069412	0.347062
47	Write AndX	1	0.091691	0.091691	0.091691	0.091691
	Transaction2 Sub-Commands					
	NT Transaction Sub-Commands					
	SMB Commands					
	Transaction2 Sub-Commands					
	NT Transaction Sub-Commands					

Display filter:  Apply Copy Save as... Close

Figure 19.4 – Service response time for SMB

The **Statistics** menu has many other options to evaluate protocols, including the following options:

- **DHCP (BOOTP)**: This provides metrics on the effectiveness of the **Dynamic Host Configuration Protocol (DHCP)**.
- **BACnet**: This provides details used when evaluating **Building Automation and Control Network (BACnet)** traffic.

- **DNS:** This provides metrics on the effectiveness of the **Domain Name System (DNS)** transactions.
- **HTTP:** This allows you to view details of **Hypertext Transfer Protocol (HTTP)** statistics. This menu choice includes submenu selections, as shown in the following screenshot:

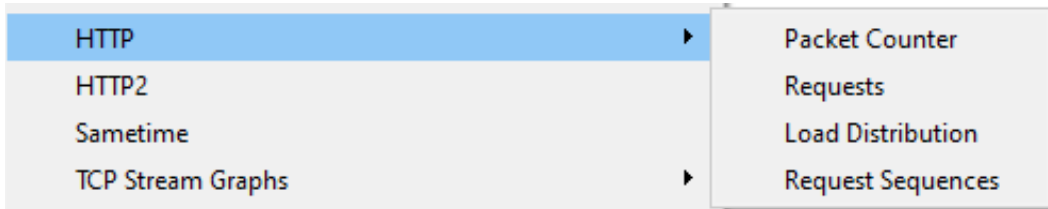


Figure 19.5 – Viewing HTTP submenu choices

The **Statistics** menu also has a **UDP Multicast Streams** report that provides the ability to monitor metrics for multicast **User Datagram Protocol (UDP)**. This is a valuable tool, as multicast streams sometimes create bursty traffic that can interfere with the effectiveness of a device.

In addition, you can also view statistics for **IP version 4 (IPv4)** and **IP version 6 (IPv6)**, as outlined in *Chapter 18, Subsetting, Saving, and Exporting Captures*.

One of the more impressive options in Wireshark is the ability to create graphs. Within the **Statistics** menu, you'll find several opportunities to generate a graph to showcase a data flow.

## Graphing capture issues

Using graphs to illustrate your point is powerful, as the visual can cut right to the core of the issue. In addition, using graphs instead of plain data will more easily engage your audience.

One type of graph you can use is a **Flow Graph**, which shows the data exchanged between hosts. To see an example of what you might see, go to `bigFlows.pcap`. In the display filter, enter `udp.stream eq 22` and run the filter. Once complete, go to **Statistics | Flow Graph**. Wireshark will then display the graph. In the lower left-hand corner of the graph, select **Limit to display filter**, which will then display the following output:

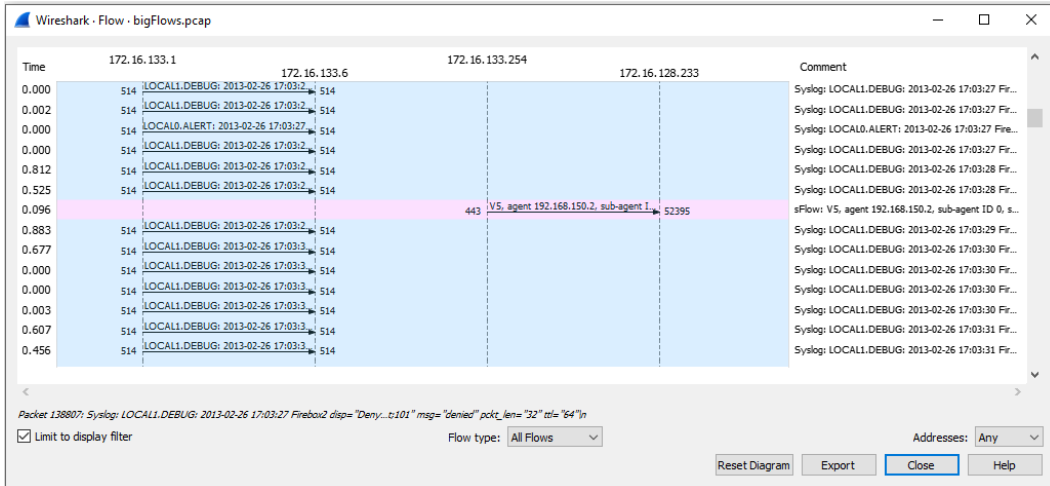


Figure 19.6 – Displaying a flow graph

Once run, the **Flow Graph** setting has several options to display the data or export the data in a variety of different formats, such as an image or a **Portable Document Format (PDF)** file.

Additionally, you can also run other graphs in the **Statistics** menu, as follows:

- **I/O Graph** will show traffic flowing in both directions.
- **TCP Stream Graph** will show traffic flowing in one direction.

Either graph will help to provide a visual of common issues, such as dropped frames, duplicate ACKs, and interruptions in the data flow.

In the next section, we'll evaluate how to create a few basic I/O graphs so that we can visualize specific types of traffic while troubleshooting the network.

## Creating I/O graphs

An I/O graph provides a way to analyze traffic flowing in both directions and can be created using a live capture or an existing trace file.

To generate graphs on a specific stream, return to `bigFlows.pcap`. In the display filter, enter `tcp.stream eq 198` and run the filter. Once Wireshark presents the information, we'll reduce the file by going to **File | Export Specified Packets...**, which will bring up this window:

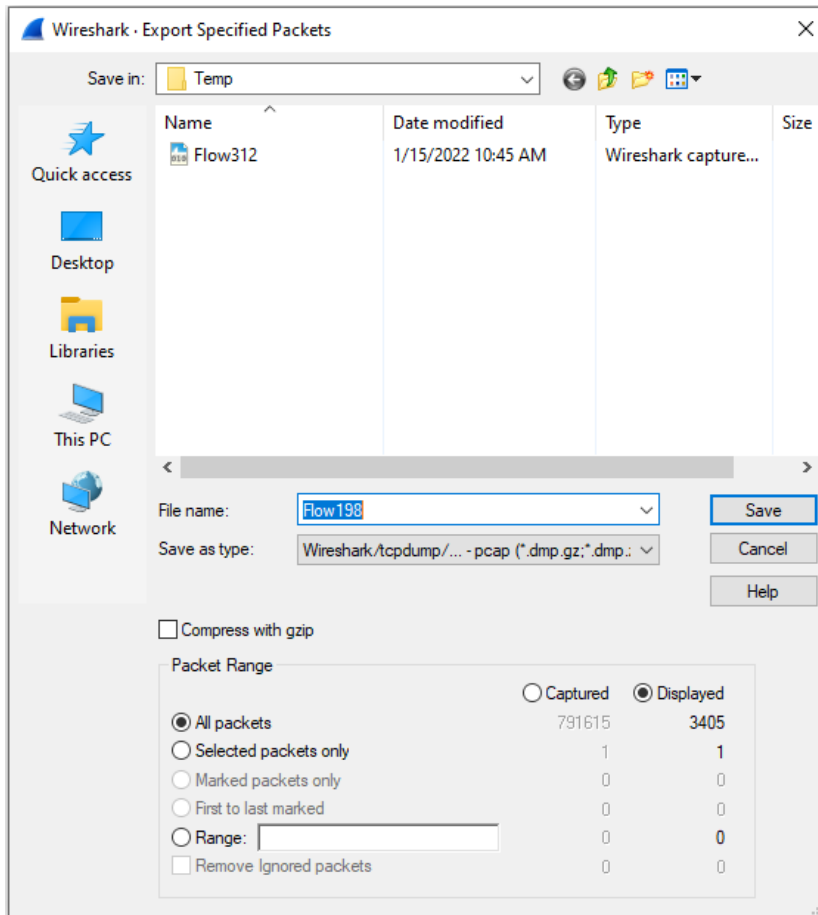


Figure 19.7 – Export Specified Packets window

Make sure you have selected **All Packets | Displayed**, as shown in *Figure 19.7*. Then, save the file as `Flow198.pcap`.

Close `bigFlows.pcap` and then open `Flow198.pcap`, where we will see evidence of a congested network.

## Examining errors

Within `Flow198.pcap`, you'll see several areas of concern, as the intelligent scroll bar is littered with black lines, which generally indicates signs of latency. By clicking on one of the black striped areas on the scroll bar, we see TCP DUP ACK (duplicate ACKs) and TCP Fast Retransmissions, as shown in the following screenshot:

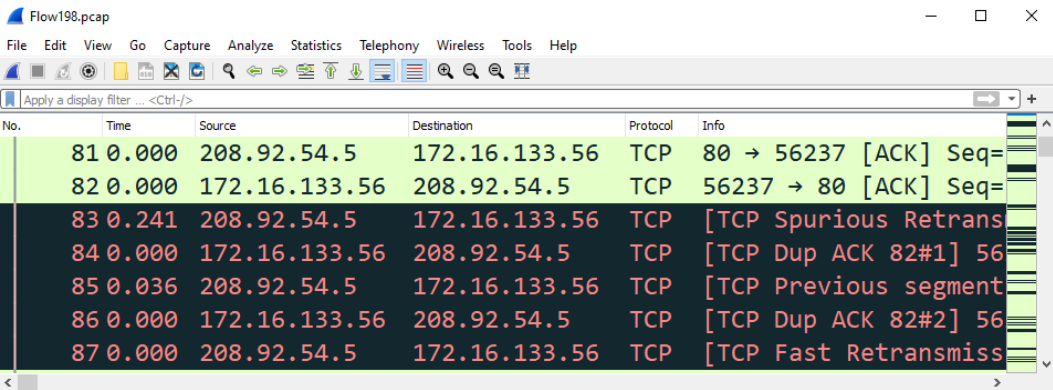


Figure 19.8 – Indications of a congested network

Next, select the **Expert Information** icon, found on the lower left-hand side of the interface. Once there, narrow the focus by displaying only the **Note** section, as shown in the following screenshot:

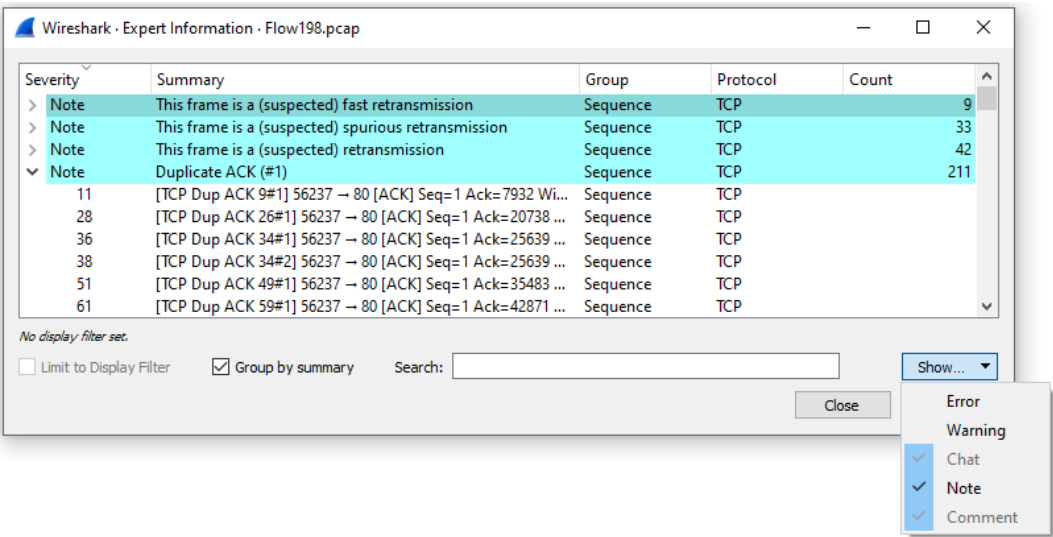


Figure 19.9 – Viewing the Expert Information Note section

Within the **Note** section, we see that Wireshark suspects that the following are contained within the capture:

- Fast retransmissions
- Spurious retransmissions
- Retransmissions
- Duplicate ACKs

All are indicative that the traffic is not moving as expected. Next, let's create a simple I/O graph of traffic that contains duplicate ACKs.

## Graphing duplicate ACKs

While in `Flow198.pcap` file, enter `tcp.analysis.duplicate_ack` in the display filter, and then press *Enter*. Next, go to **Statistics** and select **I/O Graph**. Once run, Wireshark will automatically assume you would like to view a graph of the filtered capture and display the results as shown:

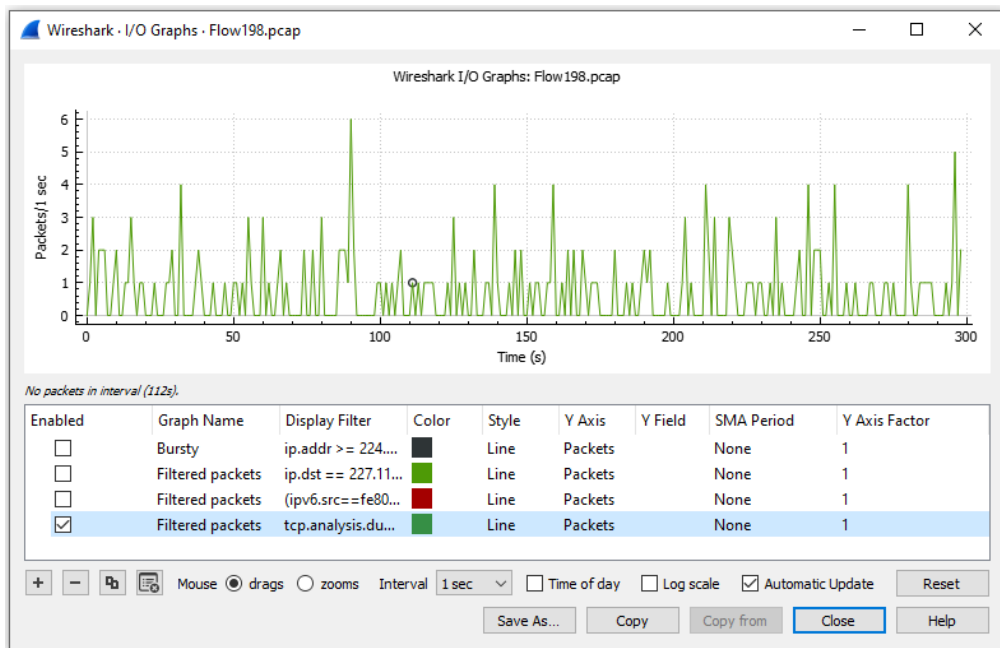


Figure 19.10 – I/O graph showing duplicate ACKs



While in the I/O graph, you'll see several areas where you can modify the settings.

## Modifying the settings

Across the top of the screen, you will see the generated graph. In the lower panel, you'll find a list of graphs and column headers that allow you to manipulate the way Wireshark presents the data. Below that, you will find an area where you can further configure the graph, along with providing the ability to add or remove an existing graph.

Let's start by reviewing the column headers as shown in an I/O graph.

## Understanding basic column headers

For each graph, you can modify and/or view the field values, as described here:

- **Enabled:** This is a checkbox where you can select whether or not to display the graph.
- **Graph Name:** By default, Wireshark generates a name; however, you can modify and personalize the field value by double-clicking on the name.
- **Display Filter:** This is a filter that you can use to narrow the results to a specific type of traffic. If you do not enter a display filter, all packets will be shown by default.
- **Color:** Indicates the color of the graph, which can be modified.
- **Style:** Provides a way to modify and present the data. To see all options, drop down the caret on the right-hand side of the field, which will display a list of options, as shown in the following screenshot:

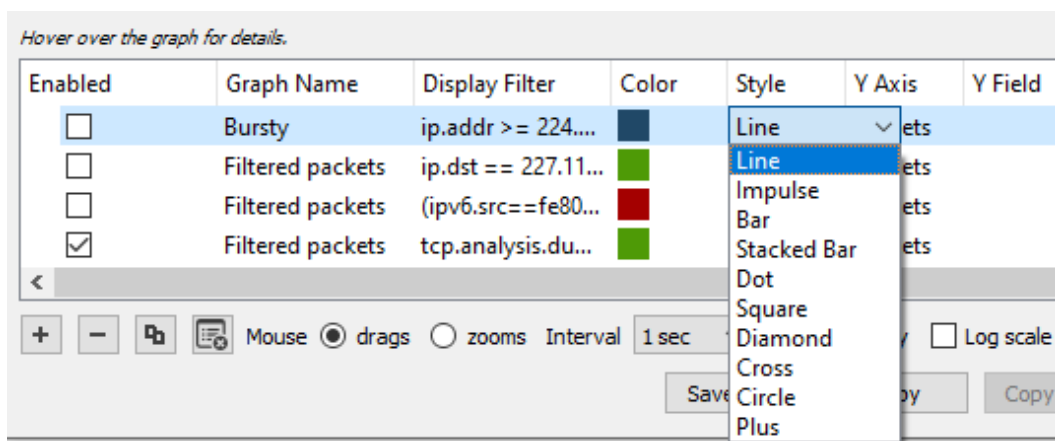


Figure 19.11 – I/O graph style options

- **Y Axis:** Offers the ability to modify the **Y Axis** to be either **Packets**, **Bytes**, **Bits**, or advanced selections, such as **SUM(Y Field)** and **COUNT(Y Field)**. If you drop down the caret, you will see the following choices:

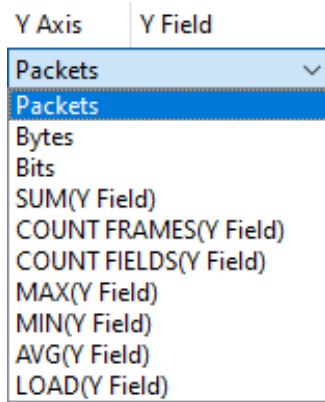


Figure 19.12 – I/O graph y-axis options

- **Y Field:** This provides the ability to specify a field value when using advanced selections in the **Y Axis**.
- **SMA Period:** This provides the ability to specify the interval of a **simple moving average** (SMA) time period.
- **Y Axis Factor:** Defaults at **1**, as changing this to another integer will modify how Wireshark presents the data.

In addition to the field values for each of the graphs, there is a section where you can add or remove graphs, along with completing other tasks.

## Adding, removing, and duplicating graphs

Underneath the list of graphs, we see several options including the following:

- **Add a new graph:** By selecting the **+** symbol, a new line will appear in the graph area. This will default as **All Packets**; however, once created, you can modify the newly created graph.
- **Remove a graph:** If you want to remove a graph, select the graph by clicking on the name of the graph and then select the **-** symbol.

- **Copy a graph:** If you want to copy a graph, select the graph by clicking on the name of the graph and select the *duplicate* symbol, which is the third symbol from the right.
- **Remove all graphs:** To remove all graphs, select the *clear all graphs* symbol, which is the fourth symbol from the left.

Along with having the ability to modify the settings, there are additional options to explore as well. Options include adding a new graph, changing the appearance of a graph, or zooming in on a specific data point.

## Exploring other options

Once in a graph, you can manipulate the appearance, add more graphs, and/or move around to better visualize the data. To explore some of these options, return to the duplicate ACKs graph you created for `Flow198.pcap`, as shown in *Figure 19.10*, and complete the following actions:

1. Select the **Add a new graph** icon, which will create an **All Packets** graph by default.
2. Enable the graph by clicking on the square to the right of the **Graph Name** option.

Once the graph is active, you can modify the appearance by changing the style. For example, change the style of the **All Packets** graph to dots using the drop-down menu, as shown here:

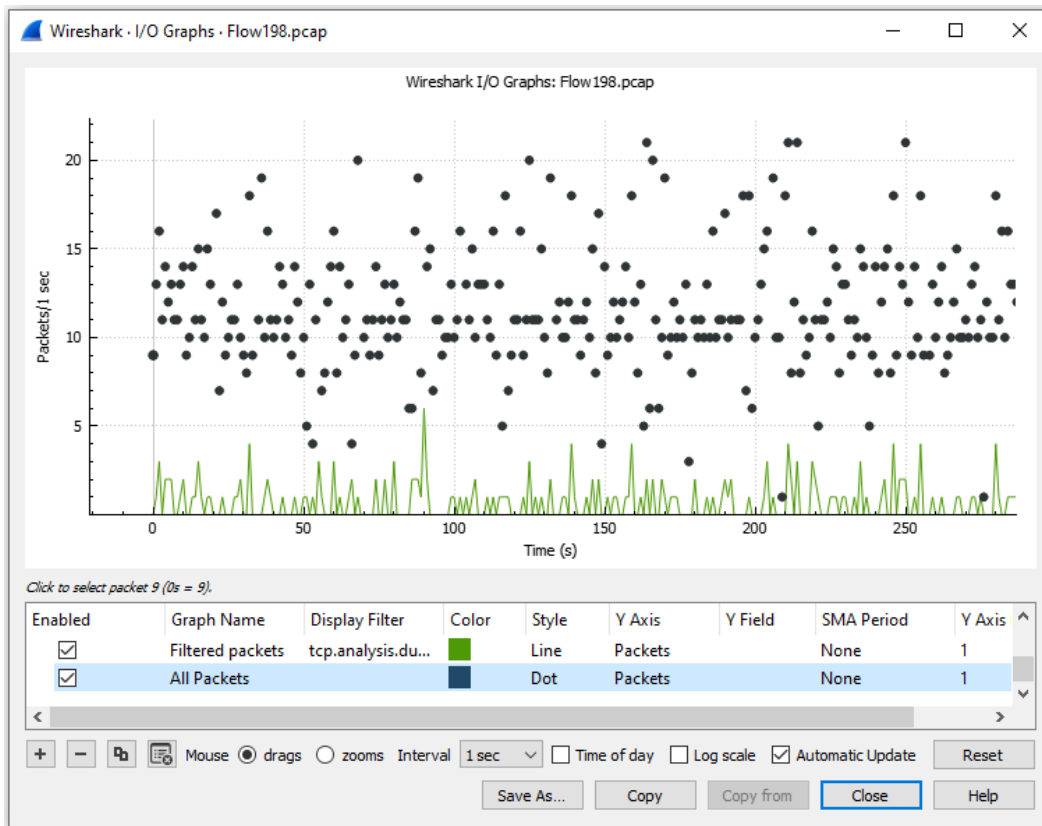


Figure 19.13 – Visualizing two I/O graphs

After you are satisfied with the appearance, you can complete the following actions:

- Select **drags** to generate a hand symbol and move about the screen.
- Lasso a section and select **zooms** to zero in on a specific area in the graph.
- Change the **Interval** option anywhere from 1 **millisecond (ms)** to 10 minutes.
- Display the **Time of day** the data was captured.
- Toggle to a logarithmic *y* axis instead of the (default) linear *y* axis by selecting **Log scale**.
- Continuously update the data during a live capture by selecting **Automatic Update**, as shown in the lower right-hand corner of the interface.

As with most options in Wireshark, you can save the graph in a variety of formats, such as an image or PDF file. In addition, once you have created a graph, Wireshark will save the results in your **Configuration Profile** settings, which can be exported so that you can share the graph with a co-worker.

As evidenced, there are a variety of ways we can view data with an I/O graph. In the next section, let's take a look at how we can create a TCP stream graph.

## Comparing TCP stream graphs

While an I/O graph provides the ability to visualize traffic flowing in *both directions*, there are times you want to focus on the traffic flowing in a *single direction*. That is where the value of **TCP stream graphs** come into play. The graphs will help provide ways to visualize the different streams in a capture.

To see what's available, go to **Statistics | TCP Stream Graphs**, as shown here:

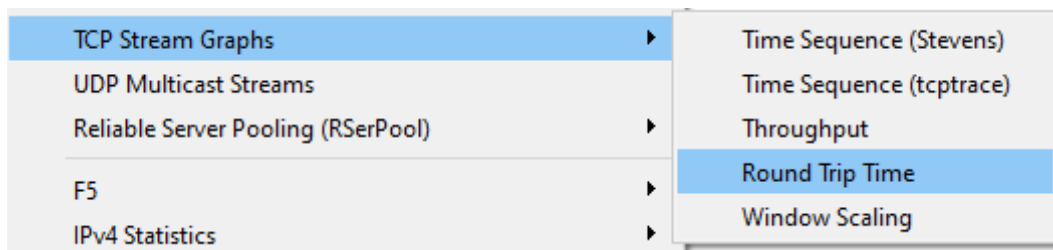


Figure 19.14 – TCP Stream Graphs menu

Once in the **TCP Stream Graphs** submenu, you can see there are multiple choices. You can select any of the graphs, which will bring up a single window. Once in the window, you can select any one of the five choices.

Let's start with viewing time sequence graphs.

## Using time sequence graphs

A time sequence graph will chart *sequence* numbers over *time*. When you first launch the graph, you will need to make sure you select the correct direction, as only one direction will be of value.

Within the **TCP Stream Graphs** submenu, you will see the following two choices:

- **Time/Sequence (Stevens)**
- **Time/Sequence (tcptrace)**

Although they have a similar function, there are subtle differences. Either one will help you spot delays or interruptions in the data transfer.

When viewing a graph of sequence numbers over time, we should see a steady flow of data. In this section, we'll step through an example of a steady stream of data using a Stevens graph. We'll then use a tcptrace graph, a more complex option that can show **selective ACKs (SACKs)** within the graph.

So that you understand the difference, let's compare the two types, starting with a Stevens graph.

## Understanding the Stevens graph

A Stevens graph allows us to view the sequence numbers over time on a single stream, using a simple format with no additional options.

To see an example, return to the `Flow198.pcap` file. Go to **Statistics | TCP Stream Graphs | Time Sequence (Stevens)**, which will result in the following output:

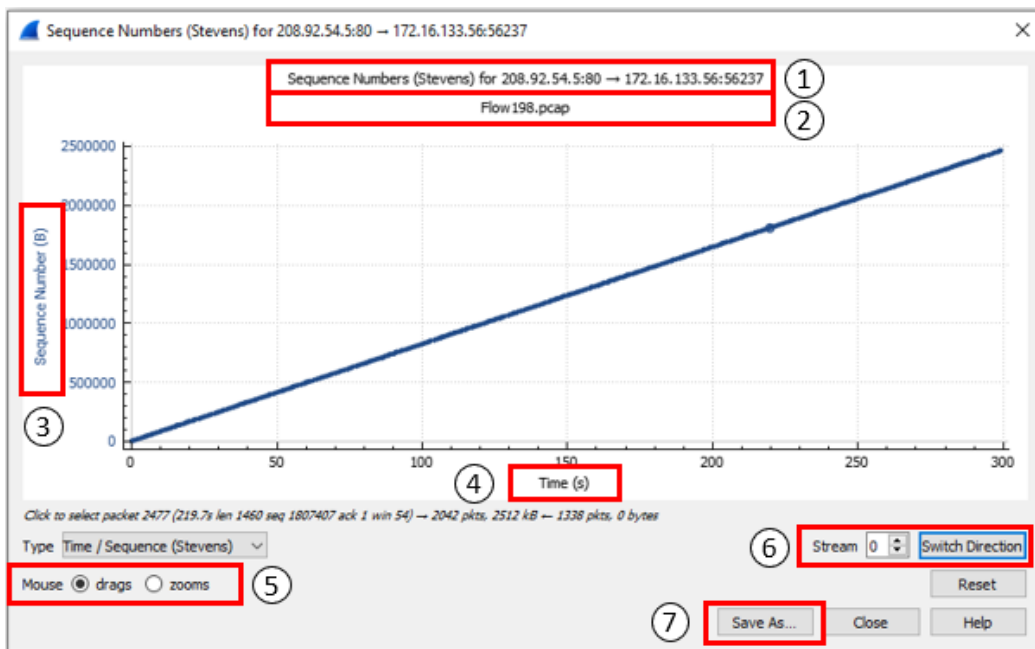


Figure 19.15 – A steady transfer of data

Once in the graph, you will see the following elements, which correspond to the numbers indicated in *Figure 19.15*:

1. This line indicates the details and direction of the stream.
2. This line identifies the name of the file.
3. Here, we find the  $y$  axis, which represents the sequence numbers in **bytes (B)**.
4. Here, we find the  $x$  axis, which represents the time shown in **seconds (s)**.
5. The mouse **drags** and **zooms** options, which work in the following manner:
  - **drags** allows you to move the graph. In addition, when you click on a specific packet, Wireshark will adjust the packet list to go directly to that packet.
  - **zooms** can be used to lasso an area of interest and *zoom in* on a particular packet(s).
6. This option will allow you to complete the following:
  - **Stream** identifies the stream number.
  - **Switch Direction** allows you to switch directions to see the traffic coming from either the client or the server.
7. **Save As...** can be used to save the graph in a PDF or image format, to use in a report, or share with a co-worker.

In addition to the Stevens graph, you can also run a `tcptrace` graph, which is similar, however presents different options when working with the graph.

## Outlining a `tcptrace` graph

A `tcptrace` graph is based on the TCP trace utility, which gathers traffic on a port and outputs the results.

Return to `bigFlows.pcap`. Once open, enter `tcp.stream eq 634` in the display filter. Select **Statistics | TCP Stream Graphs | Time / Sequence (tcptrace)** from the menu. A graph will appear, as shown in the following screenshot:

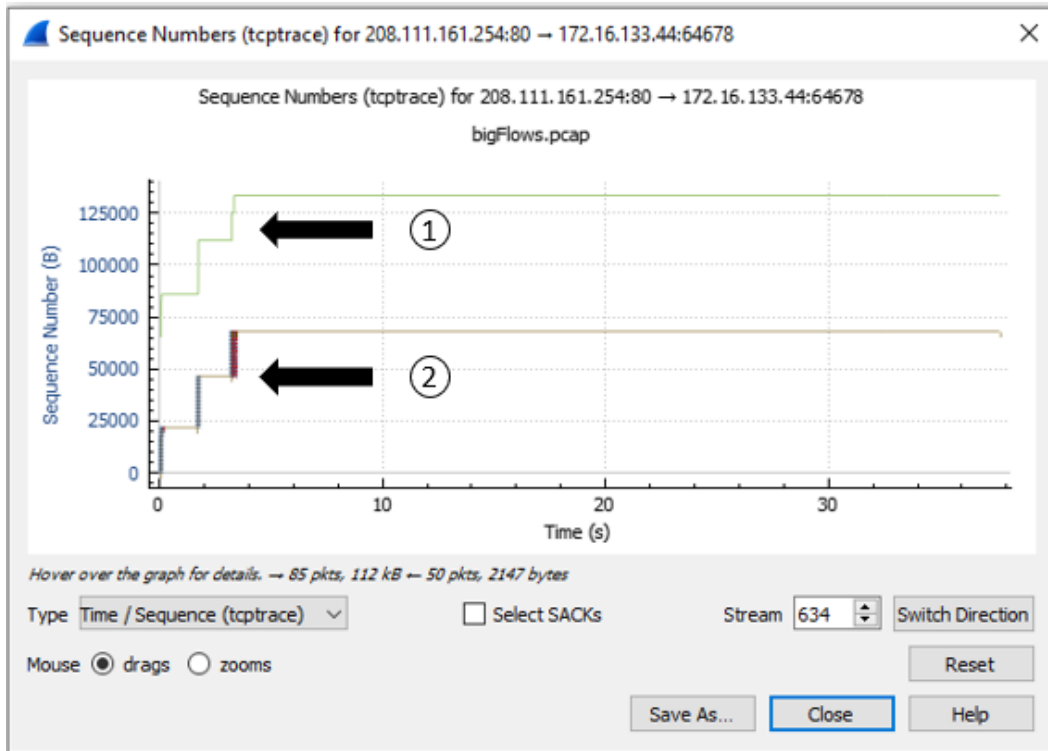


Figure 19.16 – Viewing a tcptrace graph for stream 634

Once in the graph, you will note how Wireshark keeps track of the following information:

- **Receive window**, shown as number 1 in *Figure 19.16*, is what the client can accept.
- **TCP segment**, shown as number 2 in *Figure 19.16*, is the data that is being transferred.

In addition, along the bottom of the window, you will see a checkbox labeled **Select SACKs**. Let's step through the process of enabling this option.



## Viewing SACKs

During a three-way handshake, the client and server agree on the parameters of the conversation. In most cases, each side will include TCP options to further define the variables used during the data transfer. One of the options is requesting to use **Select SACKs** during the data exchange. When used, the server will keep sending data, even if it's not in the correct order. The client will notify the sender *only* if there are missing packets, which helps prevent the need for retransmissions.

To drill down on a group of SACKs, select **zooms** in the lower left-hand corner of the screen. Once selected, lasso the grouping of packets on the left-hand side of the screen, as shown in the following screenshot:

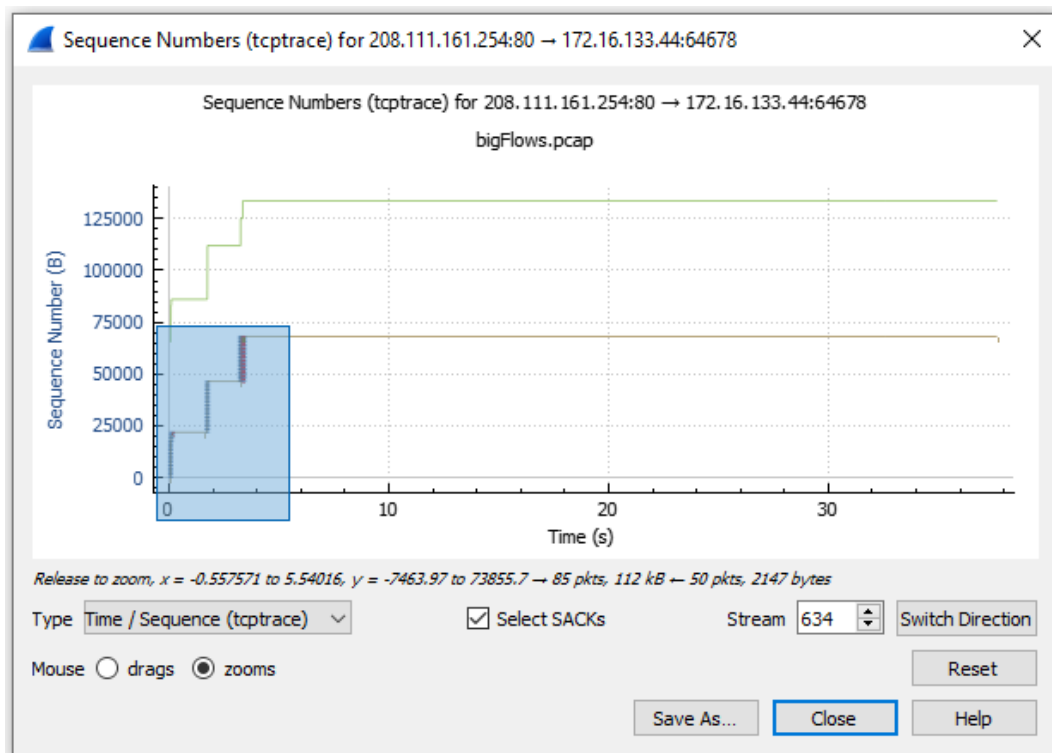


Figure 19.17 – Zooming in on SACKs

Alternate between **zooms** and **drags** with the mouse until you zero in on the following view:

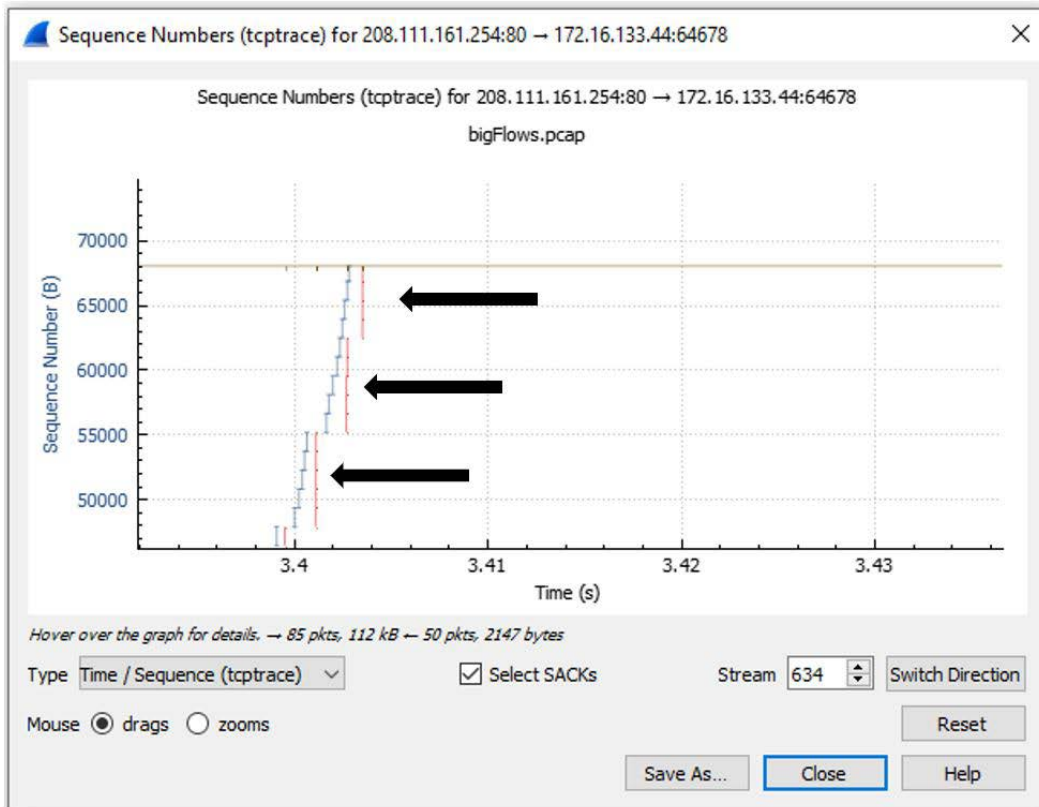


Figure 19.18 – Viewing SACKs

After zooming in on the graph, you will see several vertical lines below the TCP segment line. This represents the SACKs within the capture.

**Note**

If you do not get the desired results, you can select **Reset**, as shown on the lower right-hand side of the screen in *Figure 19.18*.

Once you have drilled down, place your cursor on one of the lines. When you select a data point, Wireshark will mirror your selection so that you can drill down on a single packet. As shown in the following image, I placed my cursor on a SACK line. Below the graph, you can see the text **Click to select packet 16479**, as highlighted in the following screenshot:

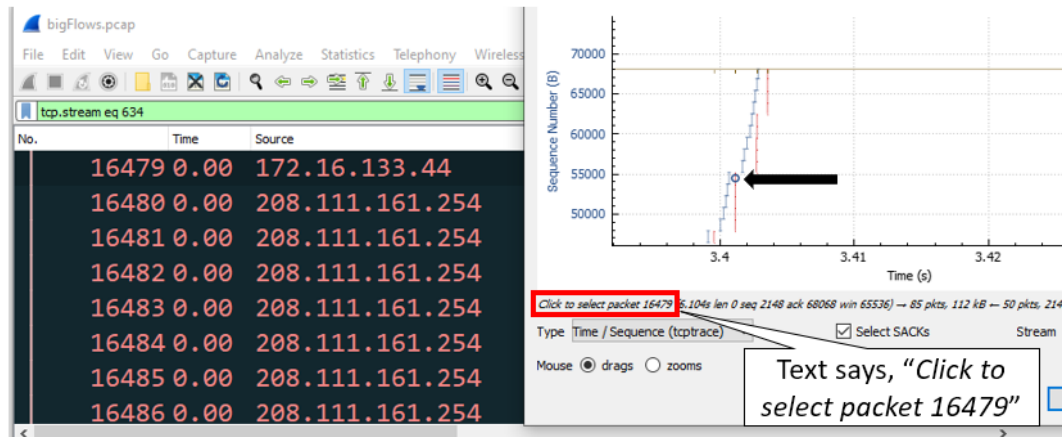


Figure 19.19 – Drilling down on a specific packet

Seeing a few SACKs is not uncommon; however, an excessive amount could mean trouble on the network. In addition, the receiving host will need to hold numerous segments within memory, which could result in the host bogging down or even crashing.

Along with providing the ability to view the transfer of data over time, another helpful tool is a throughput graph.

## Determining throughput

Throughput is how much data is sent and received (typically in **bits per second**, or **bps**) at any given time. In Wireshark, we can measure throughput as well as goodput, which is *useful* information that is transmitted.

Using `bigFlows.pcap`, enter `tcp.stream eq 634` in the display filter. Then, select **Statistics | TCP Stream Graph | Throughput**, which will generate a graph, as shown in the following screenshot:

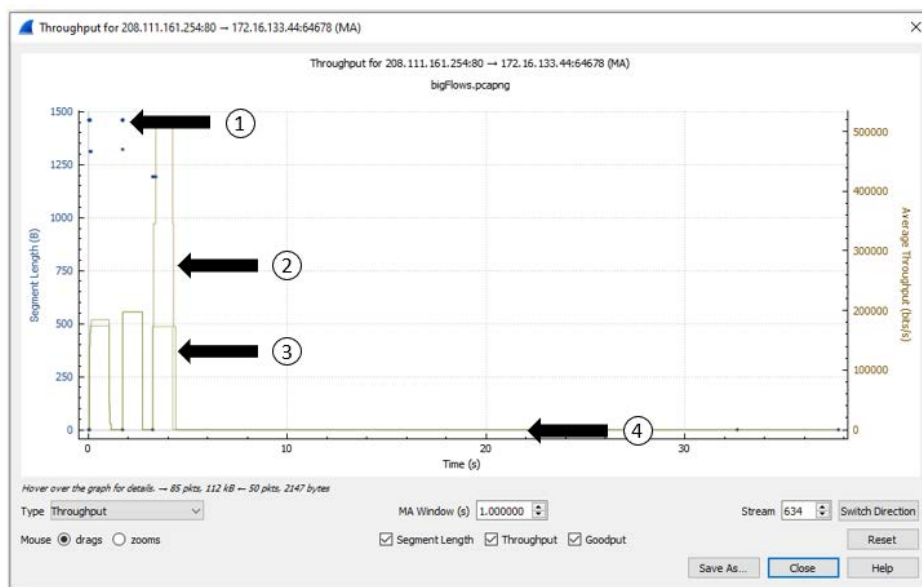


Figure 19.20 – Viewing throughput

This single graph tells us a great deal about what is happening in this capture. Once generated, I selected all three options shown along the bottom of the screen in *Figure 19.20*. Within the screenshot, I have identified several aspects of the graph, as follows:

1. The first identifier represents the segment length, which is the data that follows the TCP header, and any options.
2. The second identifier represents the throughput, which is the amount of data that is getting through.
3. The third identifier represents the goodput, which is the amount of *useful* data within the capture.
4. The fourth identifier shows all lines at 0, meaning no data is being transferred.

Another useful TCP stream graph is **round-trip time (RTT)**, which is how long it takes to make a complete round trip from A to B, and then from B to A.

## Assessing Round Trip Time

In an ideal world, the RTT will be steady, as outlined in *Chapter 17, Determining Network Latency Issues*, in the *Grasping latency, throughput, and packet loss* section. However, that is not always the case.

To view an RTT graph, return to the `Flow198.pcap` file. Select **Statistics | TCP Stream Graph | Round Trip Time**, which will generate a graph like the one shown here:

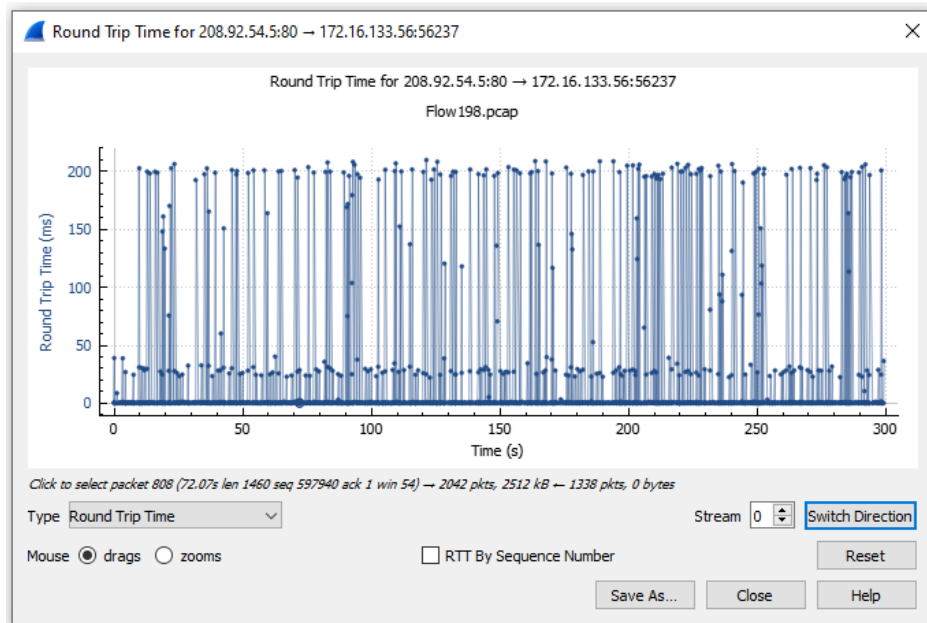


Figure 19.21 – Viewing a RTT graph

Once in the graph, you can view the data using either **RTT by Sequence Number** or **RTT by Time**. In this case, I used **RTT by Time**, which is the default. Within the graph we see the following elements:

- The  $y$  axis is the RTT in ms.
- The  $x$  axis is the sequence number.

To see the variance in this packet capture, zoom in on the beginning of the graph, as follows:

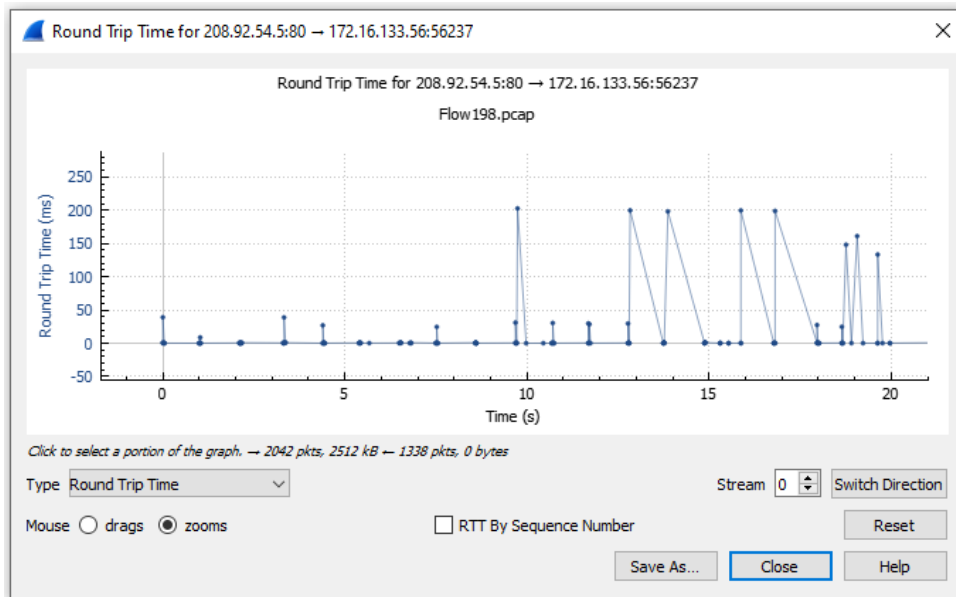


Figure 19.22 – Zooming in on RTT for Flow198.pcap

Once you zoom in, switch to **drags** and examine the variation in the RTT. In addition, you can also move the graph or zero in on a specific packet.

Next, let's investigate how we can visualize a window scaling graph.

## Evaluating window scaling

During the exchange of data, each endpoint continuously advertises a **window size (WS)** value (in bytes) that indicates how much data they can accept. Window scaling is a value that expands the *stated* WS by providing a multiplier that more accurately reflects the *true* WS.

Return to `bigFlows.pcap` and enter `tcp.stream eq 634` in the display filter. Select **Statistics | TCP Stream Graph | Window Scaling**, as shown here:

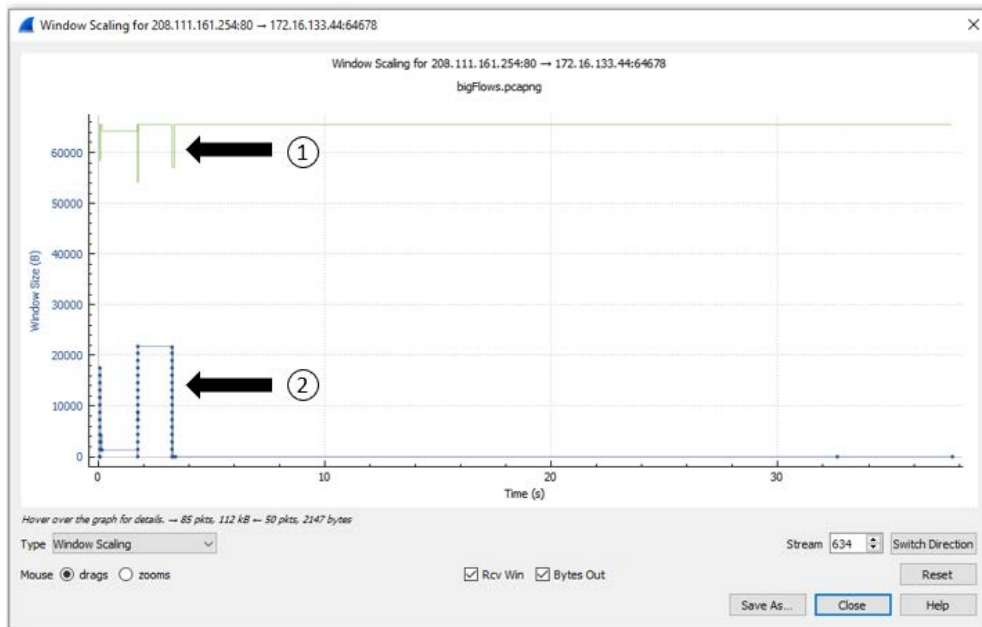


Figure 19.23 – Viewing a window scaling graph

This graph tells a few things about this capture. I selected both options shown along the bottom of the graph, as outlined here:

1. The first identifier represents the **Receive Window (Rcv Win)**, which represents the advertised WS of the endpoint receiving data.
2. The second identifier represents the **Bytes Out**, which indicates the bytes in flight.

After the peak in the **Bytes Out** line, we see a flattening of both the **Rcv Win** and **Bytes Out** line. This generally means that the receiver has not adjusted the WS and is easily able to accept a continuous stream of data.

This graph can be useful if you are experiencing bottlenecks. For example, you can monitor the receive window of a server to help you assess if the value is shrinking, growing, or staying the same over time.

## Summary

Wireshark has a number of useful tools and graphs that help network administrators visualize what's happening on the network, at any given time, using a variety of methods. In this chapter, we began with an overview of various options in the **Statistics** menu, such as general information on the packet capture, protocol hierarchy, and the ability to assess the health of numerous protocols.

We then investigated I/O graphs and learned how to utilize different filters and expressions, create differently colored graphs, and view several graphs concurrently. Additionally, we evaluated the power of using a TCP stream graph. By using examples, we first evaluated a time sequence graph and compared the differences between a Stevens and a tcptrace graph. We then summarized by learning the value of determining throughput, assessing RTT, and monitoring window scaling.

In the next chapter, you will discover **CloudShark (CS)**, an online packet analysis tool with which you can view captures in your browser, from anywhere there is internet access. We'll cover the benefits of using CS to share and analyze packet captures with your team. You'll get a good understanding of the filters, graphs, and analysis tools included within CS. In addition, we'll take a look at the many online repositories in order to locate sample captures, to enhance our packet analysis skills.

## Questions

Now, it's time to check your knowledge. Select the best response to the following questions and then check your answers, which can be found in the *Assessment* appendix:

1. To view a visual of a tiered list of protocols contained in the file, go to **Statistics** | \_\_\_\_\_.
  - A. **Capture File Properties**
  - B. **Resolved Addresses**
  - C. **Protocol Hierarchy**
  - D. **Endpoints**



2. Wireshark has several graphs you can run to visualize traffic. One type of graph you can use is a \_\_\_\_\_, which shows data exchanged between hosts.
  - A. BACnet graph
  - B. Flow graph
  - C. BOOTP graph
  - D. Display graph
3. To view a graph showing the traffic flowing in both directions use a(n) \_\_\_\_\_.
  - A. I/O graph
  - B. TCP stream graph
  - C. BOOTP graph
  - D. Display graph
4. Once you have created an I/O graph, Wireshark will save the results in your \_\_\_\_\_, which can be exported so that you can share the graph with a coworker.
  - A. Protocol chain
  - B. Stream repository
  - C. Boot file
  - D. Configuration profile
5. If, when viewing a graph of sequence numbers over time, you need to see any SACKs within the stream, you should use a \_\_\_\_\_ graph.
  - A. BOOTP
  - B. TCP stream (Stevens)
  - C. TCP stream (tcptrace)
  - D. Display g

6. \_\_\_\_\_ is how much data is sent and received (typically in bits per second) at any given time.
- A. Throughput
  - B. Throttle
  - C. SACK
  - D. Protocol chain
7. During the exchange of data, each endpoint continuously advertises a \_\_\_\_\_ value (in bytes), which indicates how much data they can accept.
- A. RTT
  - B. WS
  - C. SACK
  - D. BOOTP



# 20

# Using CloudShark for Packet Analysis

Although Wireshark is a powerful, versatile tool, there are times when you may need to involve your team in a packet analysis exercise. One site that makes it easy to share your packet captures with co-workers is **CloudShark (CS)**. CS does not have as many features as Wireshark; however, you can still complete a number of functional tasks during an analysis. In this chapter, we'll discover CS, a browser-based solution that offers several of the same benefits as Wireshark.

You'll learn that, in addition to the basic tasks, you can create an account and perform more advanced functions such as uploading and sharing captures. So that you can get the full benefit of CS, we'll step through basic packet capture analysis, such as applying filters to narrow the scope. In addition, we'll learn how to create graphs that provide a visual representation of the data. We'll also explore some of the built-in analysis tools. CS has several tools, which allow us to dissect a **Voice over Internet Protocol (VoIP)** call, complete a **Hypertext Transfer Protocol (HTTP)** analysis, and monitor for possible threats. Finally, so that you continue to improve your packet analysis skills, we will take a look at the many online repositories where you can obtain sample captures.

This chapter will address all of this by covering the following topics:

- Discovering CloudShark
- Outlining the various filters and graphs
- Evaluating the different analysis tools
- Locating sample captures

## Discovering CloudShark

Most of us would agree that Wireshark is a great tool for troubleshooting, along with identifying malware and other anomalies on a network. However, Wireshark has some limitations in that it must be installed on a local machine to gather traffic, and it can be resource-intensive. In addition, Wireshark is not designed to be used concurrently by multiple people, such as in a team.

CS is a browser-based **Software as a Service (SaaS)** that provides a way to upload packet captures and share them with co-workers or even the world. You can also do an analysis on the fly, or simply use it as a browser-based solution to learn about protocol behavior.

CS offers the following features:

- **Capture Index**, where you can store and share captures
- The main interface, where you can apply filters and add notes
- The ability to conduct advanced analysis and threat evaluation

You can find CS at <https://www.qacafe.com/analysis-tools/cloudshark/qa-cloudshark-personal-saas/>. On the main page, you can either log in or set up a free trial. To get the most out of CS, create a trial account. Once your account is active, you will begin on the welcome page, as shown in the following screenshot:

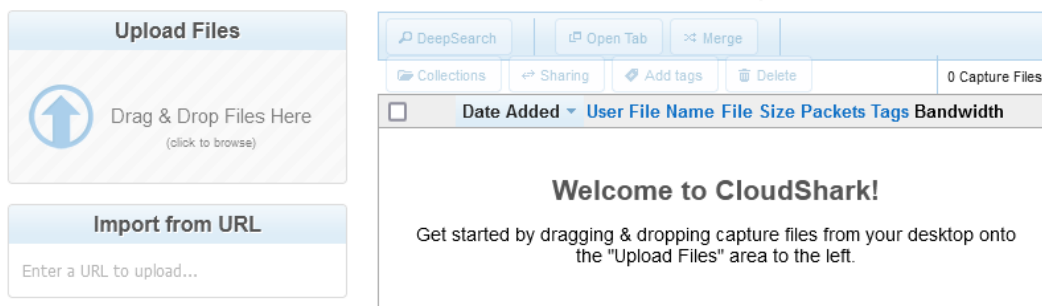


Figure 20.1 – CS welcome page

From the welcome page, you can upload files from your PC or laptop or import them from a **Uniform Resource Locator (URL)**, as shown on the left-hand side of *Figure 20.1*.

After you have uploaded your files, they will be listed on the right-hand side of the screen. Once you become familiar with the CS interface, you can adjust many aspects of your account, as we'll see next.

## Modifying the preferences

In CS, there are several areas that you can customize and fine-tune, such as account information, managing your uploads, creating collections, and enforcing quotas. To get to the **Preferences** menu, go to the top right-hand side of the screen, where there is a drop-down menu that allows you to modify the following variables:

- **Account:** Here, you can view the type of subscription information for your account. It also offers the option to start a subscription when you are ready.
- **Capture Index:** This is where you can customize your column headers. As shown in the following screenshot, you can use the default layout, or remove any of the visible fields. In addition, you can drag **Additional Columns** from the available options to where you would like them:

### Capture Index Preferences

Choose the columns for the capture table. Drag additional fields into place as well as reorder columns.

Show in Table:

Date Added	User	File Name	File Size	Packets	Tags	Bandwidth
------------	------	-----------	-----------	---------	------	-----------

Additional Columns:

Capture Start	Capture End	Duration	Group	Data Size	Type	Encapsulation
Byte Rate	Bit Rate	Avg Packet Size	Avg Packet Rate	SHA-1		

Options:

Show me  captures per page.

or

Figure 20.2 – CS Capture Index Preferences

When you are done with your selections, select **Save**, and CS will rearrange the columns according to your preferences.

- **Uploads:** CS is designed to be a collaboration tool, so it is assumed you will have interactivity, and other team members will have access to the files you have uploaded. The **Uploads** preferences is where you tell CS what group you want to assign the files you have uploaded, as shown here:

## Uploads

Cloudshark allows you to automatically assign uploaded files to one of your groups. This is useful if you're always sharing with a specific team.

Default Group:

Automatically assign any new uploads to

Group Members can:

☒ Read-Only ☐ Read/Write

Guest Access

☐ Share uploaded files with Guests

Figure 20.3 – CS Uploads preferences

Once the files are loaded, you can further restrict what group members can do with the files, by selecting either **Read-Only** or **Read/Write**. In addition, CS can provide guest access to your uploaded files.

- **API Tokens:** You may have your own tools that you want to interact with CS. This is where you can find the **Application Programming Interface (API)** token so that CS can interact with your software.
- **Usage Quotas:** Packet captures can consume a great deal of storage. This preference menu choice will list how much storage you have used out of your allocation (the trial provides 2 **gigabytes (GB)** of storage), along with how many uploads you have completed out of your allocation (the trial provides a limit of 500 files). If you are in danger of exceeding the limit, there is a link to upgrade.

- **Collections:** To organize your captures, you can create collections where you can house similar captures together. Once created, a collection will appear on a separate landing page, as shown in the following screenshot:

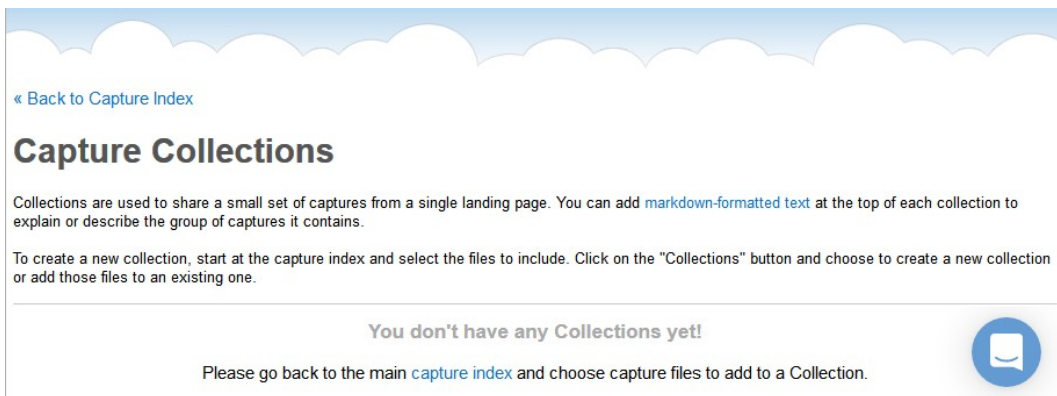


Figure 20.4 – CS capture collections

After modifying your preferences, you're ready to upload your captures, to share with your team or the world, as discussed next.

## Uploading captures

To upload and share your captures, go to the left-hand side of the CS welcome page, as shown in *Figure 20.1*. You can either drag them from your file manager and drop them in the **Upload Files** area, or you can click and browse to a file location. Once a file is uploaded, CS will display a summary, as shown here:

<input type="checkbox"/>	Date Added ▾	File Name	Byte Rate	Packets	Encapsulation	Bandwidth
<input type="checkbox"/>	Today 12:22 AM	TCP Example.pcapng	49.2 KB/sec	2073	Ethernet	

☐ Read-only  
☐ Packet annotations  
☒ File comments  
☒ Public  
☒ Saved graphs

Figure 20.5 – File upload summary



After uploading your files, click **Done** to return to the **Capture Index** main menu, which will display a list of the files in your repository. Once there, select the check mark to the left of a file and the menu choices along the top will become active, as shown in the following screenshot:

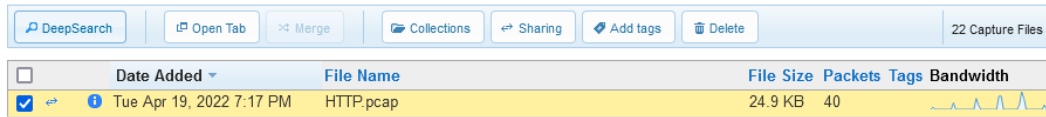


Figure 20.6 – File selected

The menu choices will allow you to complete several tasks with each file. Let's explore this concept next.

## Working with capture files

CS has several menu choices. For example, you can conduct a **DeepSearch** capture, which will allow you to search specific fields within a capture. In addition, you can also search using an **Index Filter**, found on the lower left-hand side of the **Capture Index** feature, as shown here:

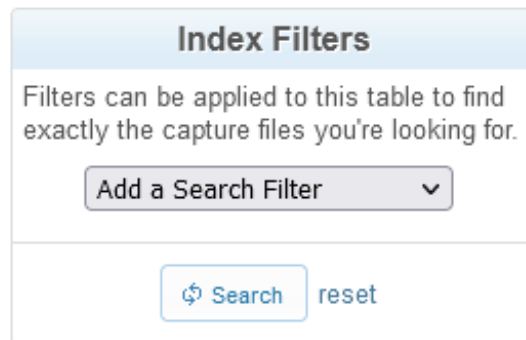
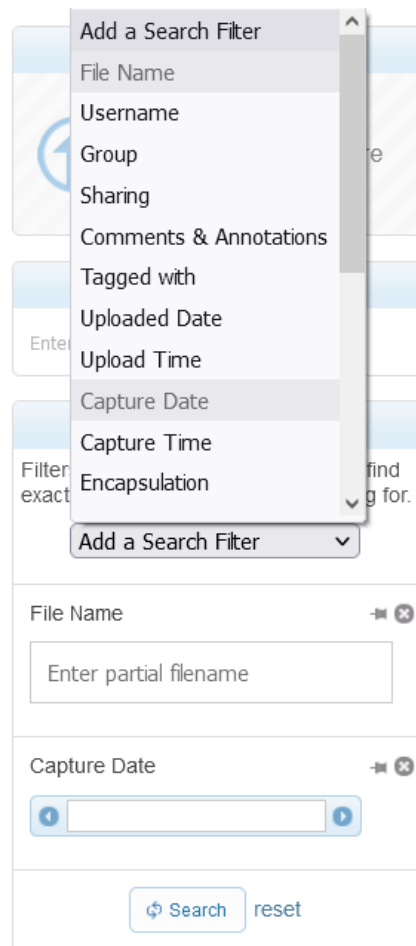


Figure 20.7 – Index Filters

To launch a search, click on the drop-down menu and select an option, as shown in the following screenshot:



The screenshot displays the CloudShark search interface. A dropdown menu is open, showing a list of search filters: Add a Search Filter, File Name, Username, Group, Sharing, Comments & Annotations, Tagged with, Uploaded Date, Upload Time, Capture Date, Capture Time, and Encapsulation. Below the dropdown, there are two input fields: 'File Name' with a placeholder 'Enter partial filename' and 'Capture Date' with a date picker. At the bottom, there is a 'Search' button and a 'reset' link.

Figure 20.8 – Using an index filter

Once you have selected an option, CS will generate a form below where you can fill in the requested variables.

You can improve the searching process by using the **Add Tags** feature to a file, to help identify the contents. For example, if you have a capture obtained from *Building 4: East Hall*, you can add a tag, as follows:

### Add Tags to 1 Capture

Please enter individual tags followed by commas. Existing tags will be suggested as you type. Press 'save' when you are done editing.

A screenshot of a text input field for adding tags. The field contains the text 'bld4\_east\_hall' followed by a small 'x' icon to remove the tag. The input field is light blue with a thin border.

Figure 20.9 – Adding a tag to a file

Over time, there may be many files in your repository. Adding tags helps you narrow your search.

When you are done with a file, you can select the **Delete** menu choice, which will permanently remove the file(s).

Next, we'll explore other tasks that you can do with your captures, including setting sharing permissions and adding to your collections.

## Sharing captures in CS

CS provides a way to securely share your captures and allow packet analysis from a wide array of devices. When you want to share a file, select **Sharing**, which will open a window as shown here:

### Update Sharing Settings for 1 Capture File

Share with one of your groups: (no change) ▼

Other members of this group can:

- ☐ View Only
- ☐ Modify & Delete

### Share with Guests

Public files are viewable by anyone who knows the URL for the file, without having to log-in.

- ☒ No Change
- ☐ Not shared
- ☐ Public


 Save or Cancel

Figure 20.10 – Updating sharing settings

You can share with one of your groups and define what they can do once they access the file using either **View Only** or **Modify & Delete**. You can also set **Share with Guests** to one of the following options: **No Change**, **Not shared**, or **Public**.

In addition to sharing, you can also create capture collections to group and organize similar captures together.

## Adding to a collection

Collections are like a folder and provide a handy way to organize similar types of files. To add a file to a collection, select the **Collections** button. If you do not have any collections, you can create a new collection from the drop-down menu, as shown here:

### Add 1 capture to a Collection

Collections are used to share small sets of capture files from a single page. Each collection is assigned a unique URL and can be made public along with descriptive text.

Your [Collections list](#) is available under the Preferences menu.

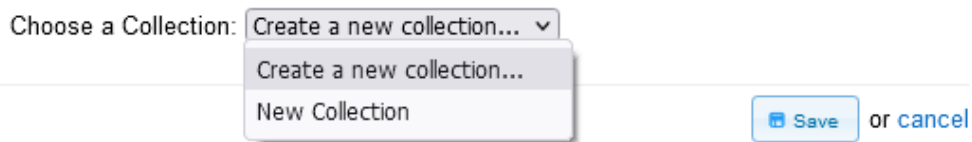


Figure 20.11 – Adding a file to a collection

To begin, drop down the menu choice, select **Create a new collection...**, and then select **Save**. This will then open a form where you can enter a name for your collection. In my example, I used the name **Basic Analysis**. After the name, you can provide a brief description, as shown in the following screenshot:

**Name:**

**Describe this Collection** [\[preview markdown\]](#)

Figure 20.12 – New collection

Below the form is where you can set the access privileges to either **Private** or **Public**. In addition, you can select individual file permissions, as shown in the following screenshot:


**Collection Access:** ☒ Private ☐ Public

Private collections are only visible to the owner. A public collection is only accessible to those who have been given the unique URL regardless if they are logged in to a CloudShark account. This setting does not affect the individual files.

**Individual File Permissions:**

**1 Capture File:**

Uncheck files to remove them from this collection.

	File name	Packets	Size	
<input checked="" type="checkbox"/>	TCP Example.pcapng	2073	1.2 MB	

[↔ Public File](#)

Figure 20.13 – Collection access

When done, select **Save** to return to **Capture Index**. Once at **Capture Index**, you can select a file and either double-click or select the **Open Tab** menu choice to open the file in the analysis window.

After you create an account, CS provides a way to customize the interface for you and your team. In the next section, we'll discover how to create a custom profile to personalize your workflow.

## Creating a profile

Once in CS, you'll find an interface that looks similar to the Wireshark interface. CS is flexible in that you can make some modifications. To show you some of the features, we'll use the HTTP.pcap file, located at <https://www.cloudshark.org/captures/0012f52602a3>.

After you open the file, you can adjust the interface. For example, to give you more room, you can move the packet bytes window to the right so that you can expand the protocol trees, as shown in this screenshot:

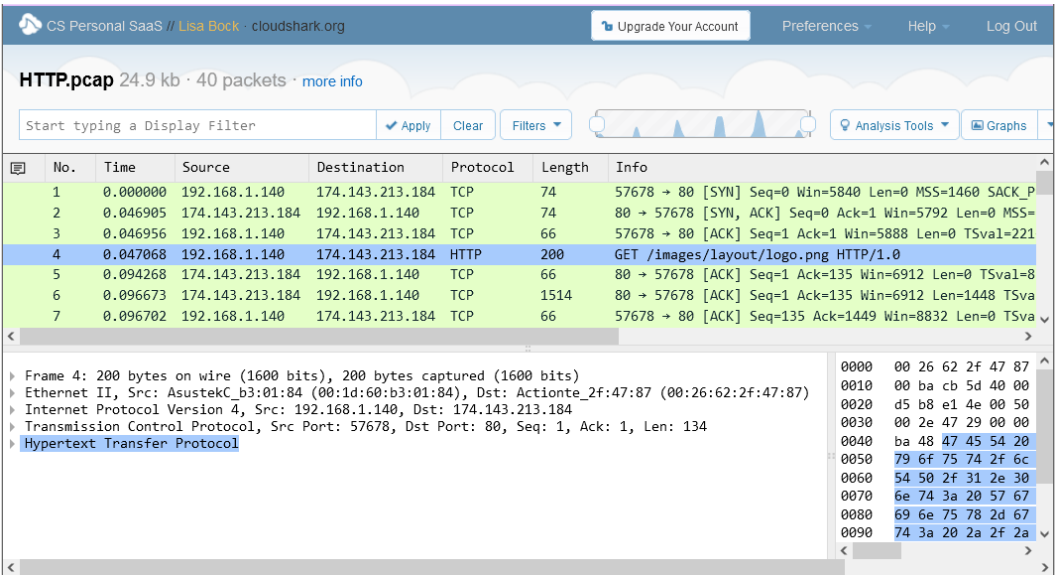


Figure 20.14 – Modified interface

In addition, you can make modifications to customize your workflow by creating a unique profile. Select the **Profiles** drop-down menu choice, and then select **New Profile**, which will launch the following window:

Profile Name

New Profile

Description (markdown allowed)

**Profile Sharing**

Sharing profiles across your team lets everybody start their analysis from the same point. Changes you make to this profile will affect all other users and capture files associated with it.

**Access Permissions**

Owner: Lisa Bock ▾

Group: -- No Group -- ▾

☐ Allow group to modify the profile

Create a NEW profile ▾ ★ Create or [cancel](#)

Figure 20.15 – Creating a new profile

Within the interface, you can select one of the tabs that include **Columns**, **Filters**, and **Protocol Preferences** to create your custom profile.

Now that your capture is open and you have modified and customized the interface, you are ready for your analysis. In the next section, we'll evaluate the choices for filtering and graphing traffic.

## Outlining the various filters and graphs

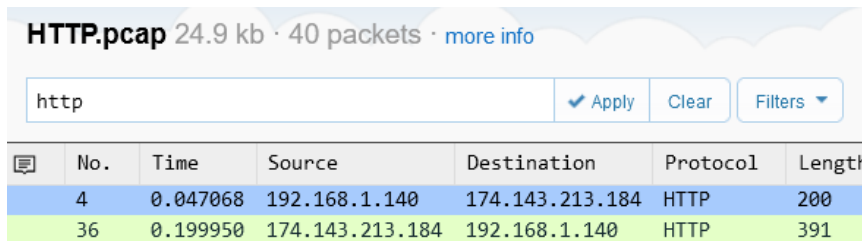
Within CS, there are several ways to view your captures. Filters narrow a capture to display only the traffic you want to see, and graphs provide a visual representation of the data.

One common task is to apply a display filter. CS's easy-to-use interface provides a way to narrow your scope. Let's learn more about this in the next section.

## Displaying data using filters

Display filters in CS are comparable to the way Wireshark filters data. Filters can be applied to identify packets with specific ports, IP addresses, or protocols by entering a filter in the upper left-hand side of the interface. After you enter a filter, select **Apply** to run the filter.

In the following screenshot, I used the `http` filter, which narrowed the capture to show only HTTP traffic:

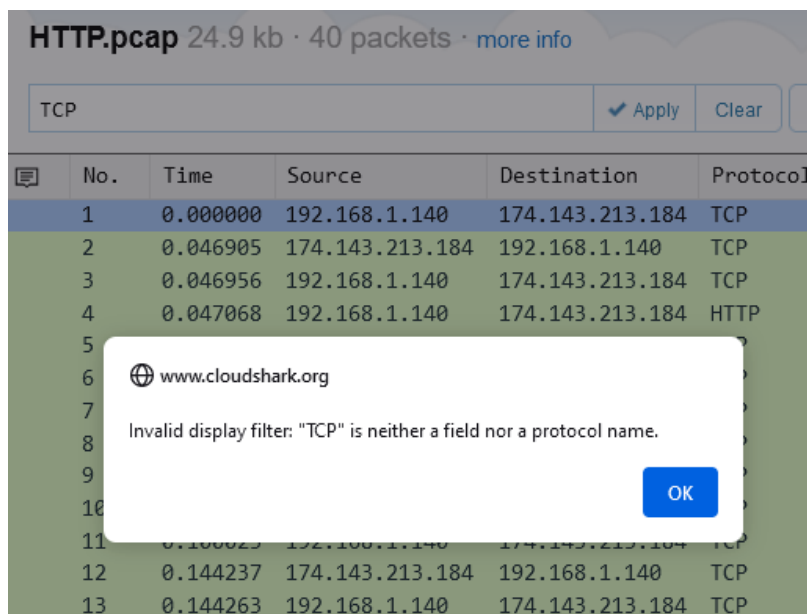


The screenshot shows the CloudShark interface with the filter `http` applied. The table below represents the data shown in the interface.

No.	Time	Source	Destination	Protocol	Length
4	0.047068	192.168.1.140	174.143.213.184	HTTP	200
36	0.199950	174.143.213.184	192.168.1.140	HTTP	391

Figure 20.16 – HTTP traffic

When using display filters, the syntax must be correct, or you will see an error. For example, the TCP filter (using capital letters) does not use the correct syntax and will generate an error, as shown here:



The screenshot shows the CloudShark interface with the filter `TCP` applied. An error dialog box is displayed over the packet list, indicating that the filter is invalid.

No.	Time	Source	Destination	Protocol
1	0.000000	192.168.1.140	174.143.213.184	TCP
2	0.046905	174.143.213.184	192.168.1.140	TCP
3	0.046956	192.168.1.140	174.143.213.184	TCP
4	0.047068	192.168.1.140	174.143.213.184	HTTP
5				
6				
7				
8				
9				
10				
11				
12	0.144237	174.143.213.184	192.168.1.140	TCP
13	0.144263	192.168.1.140	174.143.213.184	TCP

Error message: Invalid display filter: "TCP" is neither a field nor a protocol name.

Figure 20.17 – Syntax error



To effectively run this filter, you must enter `tcp` (using lowercase letters), which is similar to the way Wireshark uses filters.

In addition to the standard filters, you can also create a search by string or **hexadecimal** (**hex**) values, as outlined in the following examples:

- To look for a specific image, enter `frame contains "adc_pet_dog_336x280"` and use quotes around the string value.
- To search for a specific **media access control** (MAC) address, use `frame contains 28:e3:47:8c:02:60`.

CS will then search and present the results, if any.

Filters help to narrow the scope. Now, let's take a look at the various graphs you can quickly apply while in CS to help visually represent the data.

## Viewing data using graphs

Once in your capture, you may want to create a graph of either all traffic or of the filtered capture. In the upper right-hand corner, there is a drop-down **Graphs** menu, as shown here:

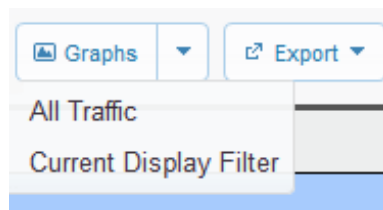


Figure 20.18 – CS Graphs menu

After you select the type of graph you would like, CS will display the graph. If you would like more interactivity, select **Open in new window** and then select **Open in Editor**, found in the upper right-hand corner, as shown in the following screenshot:

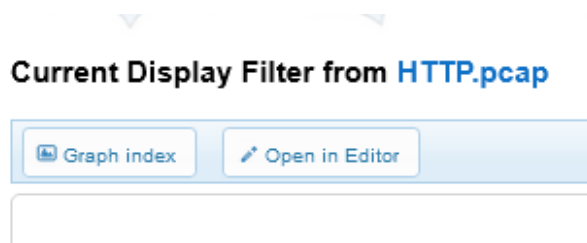


Figure 20.19 – Open In Editor option

This will open a window that displays the graph, as shown:

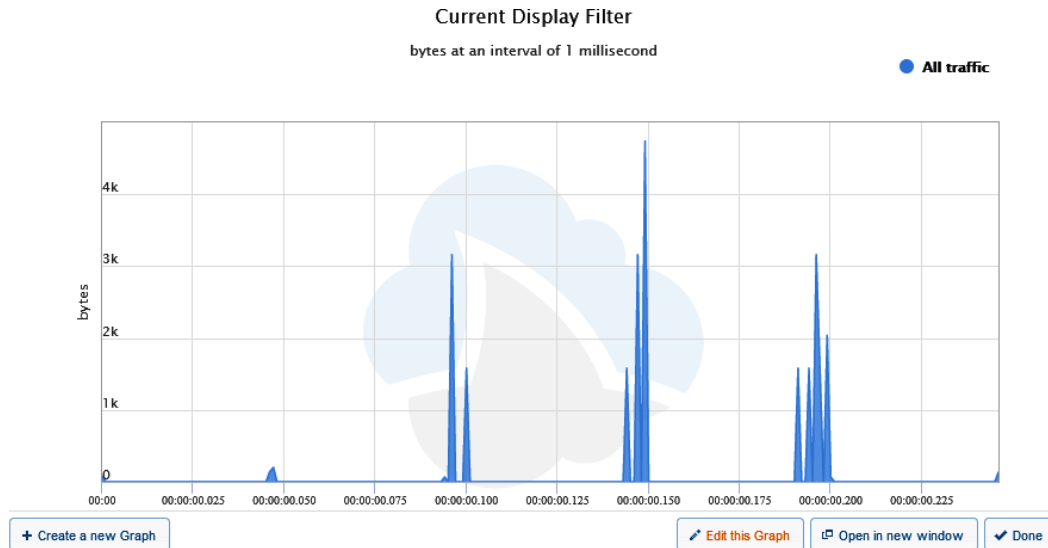


Figure 20.20 – Viewing a CS graph

Along the bottom right-hand side of the graph, select **Edit this Graph**, which will open a new window, as shown here:

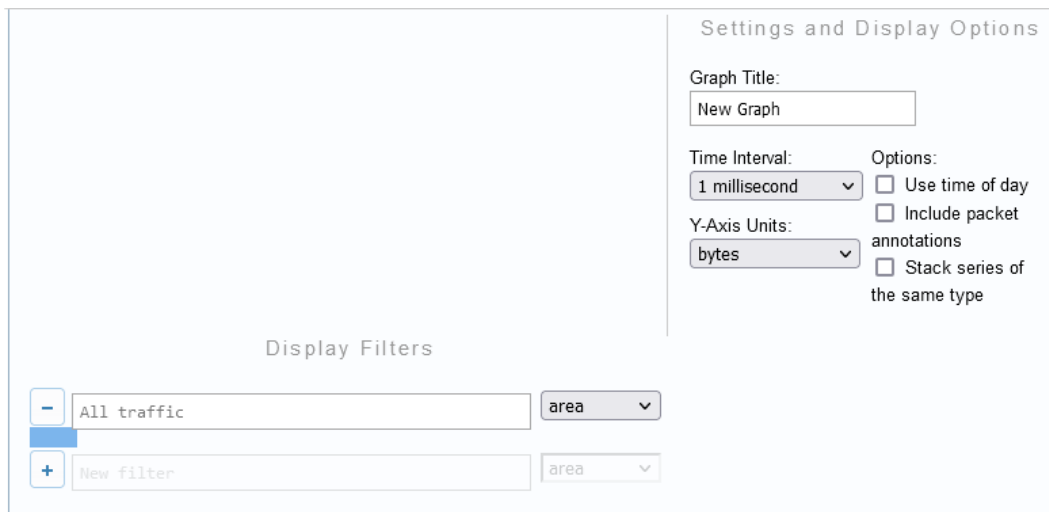


Figure 20.21 – Editing options in a CS graph

Once there, you can add or modify the following options:

- **Graph Title:** Add a title that is reflective of what the graph represents.
- **Time Interval:** Set in milliseconds, seconds, or minimum.
- **Y-Axis Units:** Set by packets, bytes, value, or by packets, bytes, or bits/second.
- **Options:** To further customize the chart, you can add additional variables that include the following:
  - **Use time of day**
  - **Include packet annotations**
  - **Stack series of the same type**
- **Display Filters:** This is where you would enter a display filter. You can also select a style for how you want the data to be represented, such as line, column, or spline.

Once you have completed the graph, you can either print the graph or export it as an image in one of the following formats:

- **Portable Network Graphics (PNG)**
- **Joint Photographic Experts Group (JPEG)**
- **Scalable Vector Graphics (SVG)**

In addition to graphs and filters, there are times when you need to perform more advanced analysis. The next section provides an overview of a variety of tools for quickly examining data.

## Evaluating the different analysis tools

CS offers a solution to share packet captures with your team. Several analysis tools are available via a web interface. In addition to the graphs in CS, there are many other built-in analysis tools. The drop-down menu for the **Analysis Tools** is located in the upper right-hand part of the screen, where you will find various menu choices, as shown here:

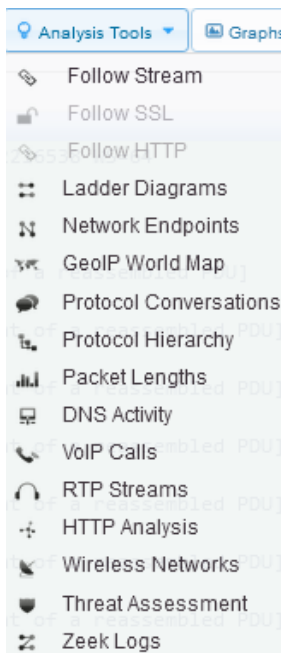


Figure 20.22 – Viewing analysis tools

If any of the options are dimmed, such as **Follow SSL** and **Follow HTTP**, as shown in *Figure 20.22*, that means the tool is not applicable to the current capture.

From the top of the list, you will find many tools to use in your analysis. Let's begin with viewing conversations, ladder diagrams, and filtering the stream.

## Following the stream and viewing conversations

Within Wireshark, we have many tools under the **Statistics** menu that help us make sense of a packet capture. While CS doesn't have as many features, you'll see that you can do a preliminary evaluation on the fly with the built-in analysis tools.

The following list outlines the first few selections in the **Analysis Tools** menu choice:

- **Follow Stream, Follow SSL, and Follow HTTP:** Similar to the **Follow the Stream** function in Wireshark, these provide a way to see the details of a single conversation between two endpoints.
- **Ladder Diagrams:** These are similar to flow graphs in Wireshark, showing endpoints communicating back and forth, as illustrated in the following screenshot:

### Protocol Ladder View: HTTP.pcap

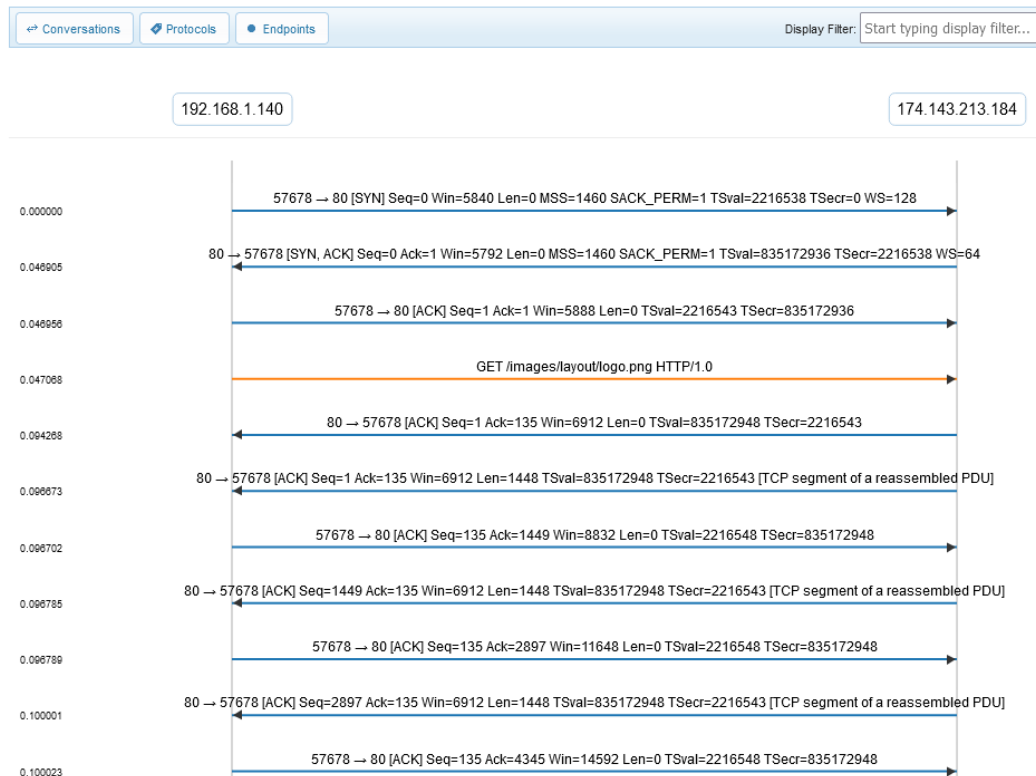


Figure 20.23 – Ladder diagram

- **Network Endpoints:** This will provide a list of endpoints. While in the window, you can filter by the type of endpoint you would like to see, such as **eth**, **ipv4**, **ipv6**, **tcp**, or **udp**, as shown in the following screenshot:

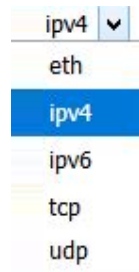


Figure 20.24 – Endpoints

- **GeoIP World Map:** When selected, it will show from where packets originate, as shown here:

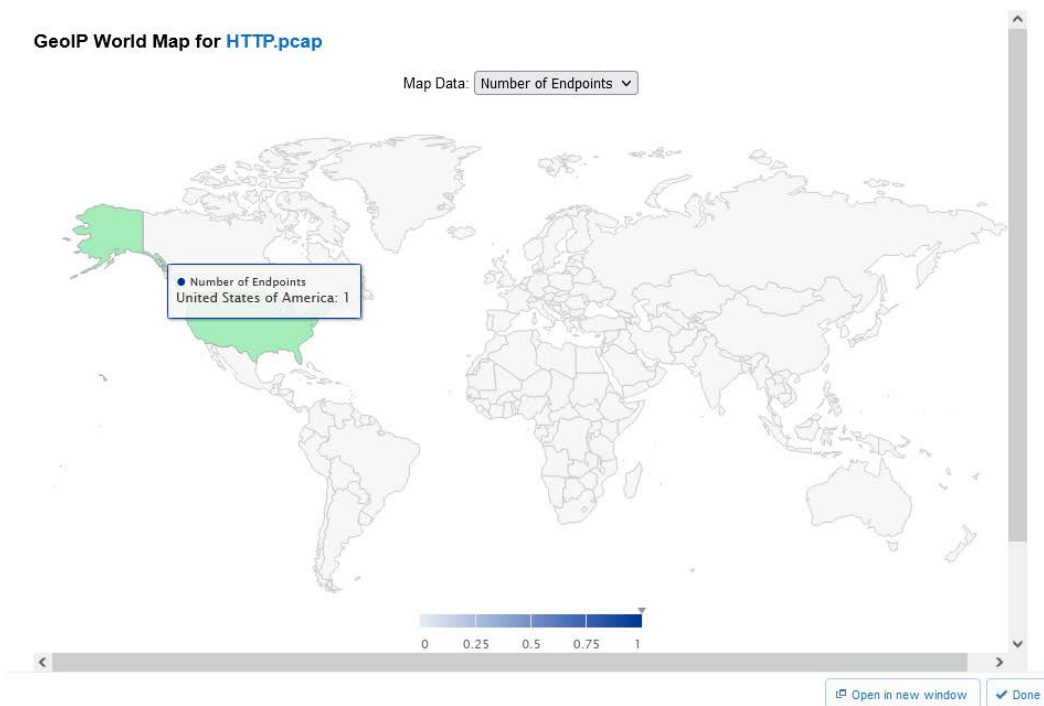


Figure 20.25 – GeoIP World Map

- **Protocol Conversations:** This will provide a list of conversations, similar to Wireshark. While in the window, you can filter by the type of conversation you would like to see, such as **eth**, **ipv4**, **ipv6**, **tcp**, or **udp**.

CS is populated with many tools that you can use to analyze data. The next section shows how we can take a look at the details of a VoIP call, graph packet lengths, and **Domain Name System** (DNS) activity.

## Viewing packet lengths and VoIP activity

Some of the analysis tools may not make sense when you look at them; however, they do provide value while troubleshooting, so it's worth running a few of the graphs to see the results.

The next grouping of analysis tools includes the following options:

- **Packet Lengths:** This provides an interactive graph of packet lengths and other information such as **Average**, **Min**, **Max**, and **Rate**, as shown in the following screenshot:

Packet Lengths in [HTTP.pcap](#)

Click on a bar to filter the capture file to only those packets.

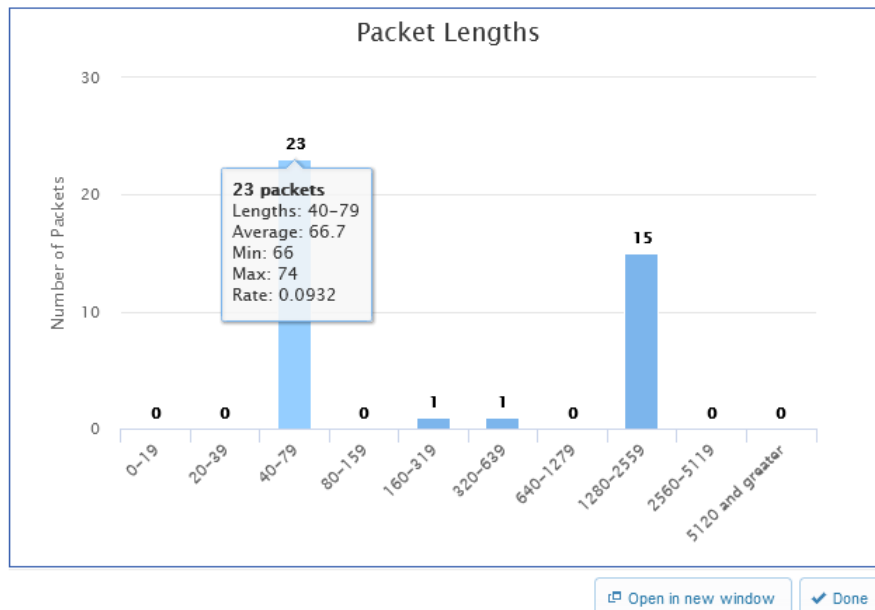


Figure 20.26 – Packet Lengths

- **DNS Activity:** This provides a count of DNS queries, responses, and **Resource Record (RR)** types.
- **VoIP Calls:** This provides a list of any VoIP calls found in the file. In addition, there is also a sortable summary of the following elements: **Call**, **Start Time**, **Stop Time**, **Initial Speaker**, **From**, **To**, **Protocol**, and **Packets**, as shown here:

Showing 1 VoIP Call from [voip-extension2downata.pcap](#)

Click on a row to open the SIP flow diagram for that conversation. If the conversation includes any RTP streams, they may be playable within CloudShark.

Call	Start Time	Stop Time	Initial Speaker	From	To	Protocol	Packets
0	7.477406	25.609087	192.168.5.10	"107"<sip:107@192.168.5.5>	<sip:84254978362@192.168.5.5>	SIP	18

[View entire call flow](#)
[SIP statistics](#)
[Open in new window](#)
[Done](#)

Figure 20.27 – CS VoIP calls

- **RTP Streams:** This lists **Real-time Transport Protocol (RTP)** streams found in the file.

Another grouping of tools can dissect HTTP traffic and help identify wireless network issues.

## Exploring HTTP analysis and wireless traffic

When either troubleshooting HTTP or wireless networks, CS analysis tools include the following options:

- **HTTP Analysis:** This will list all URLs requested in the capture file, along with a count of the requests.
- **Wireless Networks:** This provides a list of any wireless networks found in the file, along with a sortable summary of the following: **basic service set identifier (BSSID)**, SSID, vendor, `Signal_dBm`, channel, and security.

CS has many tools that are similar to those found in Wireshark. However, this last grouping offers the unique ability to quickly assess a packet capture for potential malicious activity.



## Monitoring possible threats

CS has two tools that allow you to determine if there is any suspicious activity within the capture, as follows:

- **Threat Assessments:** This is a more advanced option that will scan the capture for potentially malicious traffic within the file. If none is found, the report will come back with the notification: all clear!
- **Zeek Logs:** Zeek is an open source network analyzer used to monitor traffic for unusual or suspicious activity.

These tools will aid the analyst when assessing threats on the network. To see an example of a capture that contains malicious activity, go to <https://www.cloudshark.org/captures/f35aa6fcd160>. Select **Analysis Tools** | **Threat Assessment**, and CS will display the following output:

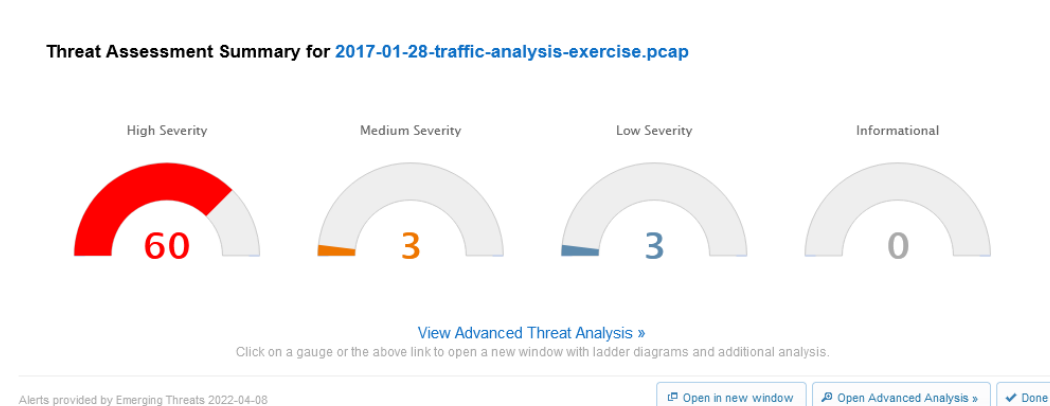


Figure 20.28 – Viewing a threat assessment report

By running a quick report, you can immediately see that there is a **High Severity** threat assessment level. Select **View Advanced Threat Analysis**, found in the middle of the screen, which will open another window, as shown here:

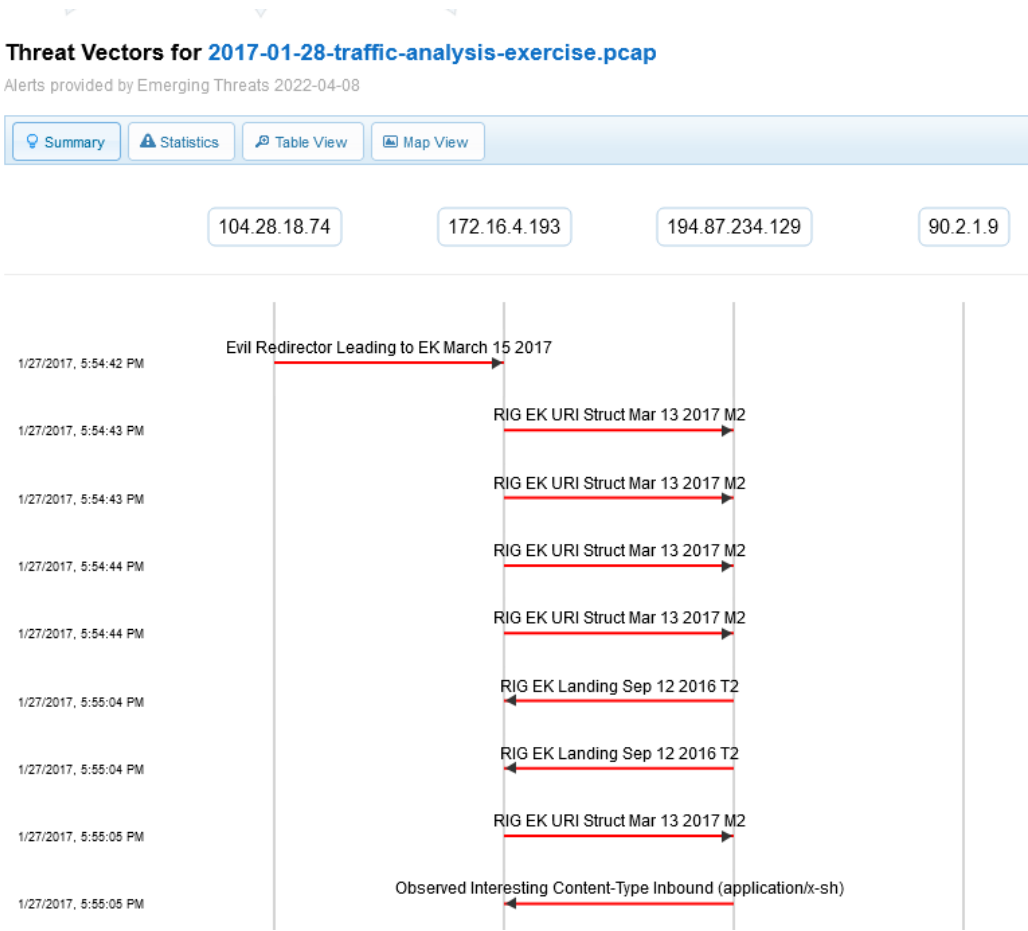


Figure 20.29 – Examining the advanced threat analysis

In addition, you can run a report on **Zeek Logs** found within the capture. Select **Analysis Tools | Zeek Logs**, and CS will bring up a summary of the log information found in the file, as shown here:

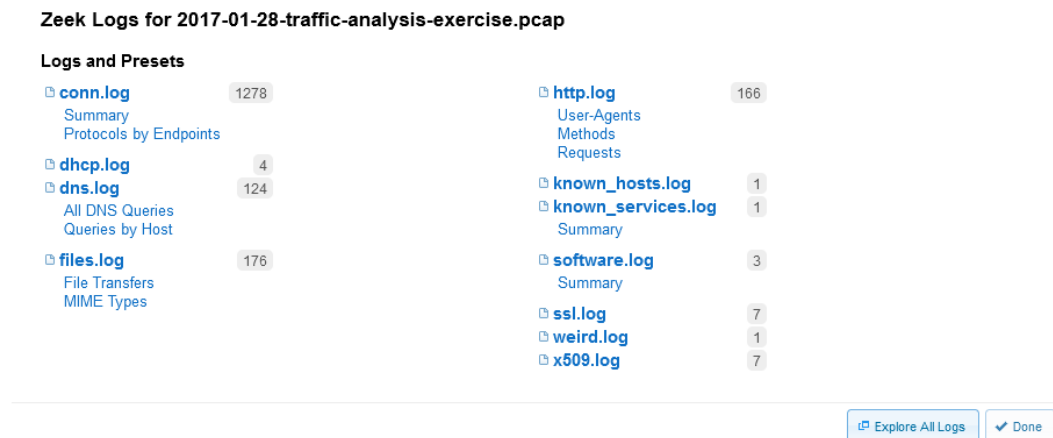


Figure 20.30 – Zeek Logs

Once in the summary, you can select **Explore All Logs**, as shown in the lower right-hand corner of *Figure 20.30*, to discover more details.

Now that you have seen the many ways in which you can analyze data using CS, let's take a look at where you can get packet captures to strengthen your analysis skills.

## Locating sample captures

While learning about packet analysis, it's important to study a variety of captures until you are proficient at knowing what to look for in a file. This may take a while, but it will be well worth the effort.

First, let's see how `PacketLife.net` provides a handy way to open and examine a packet capture, right in CS.

## Examining captures

When working with packet captures, you may want to learn about an unfamiliar protocol with your team. Today there are many places to obtain packet captures; one site I visit often is `https://packetlife.net/`.

Once at PacketLife.net, navigate to <http://packetlife.net/captures/>, where you can search for captures. For example, I found `snmp-ipv4.pcap`, as shown in the following screenshot:

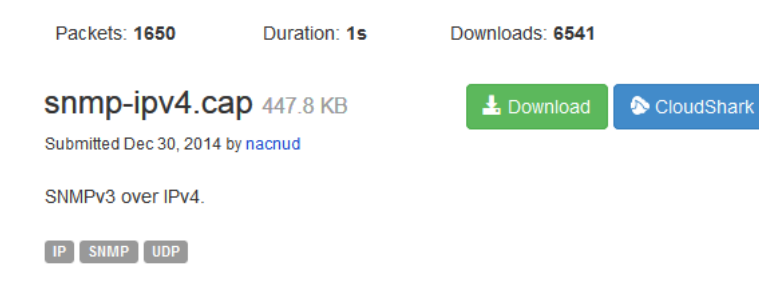


Figure 20.31 – Packet capture found at PacketLife.net

After you have found a packet capture, you can open it directly in CS, as shown here:

CloudShark Hosted // cloudshark.org      Guest upload is turned off      Log In

<http://packetlife.net/captures/snmp-ipv4.cap> 447.8 kb · 2100 packets · [more info](#)

Start typing a Display Filter    Apply    Clear

No.		Time	Source	Destination
1	□	0.000000	10.0.0.150	10.254.0.10
2	□	0.001071	10.254.0.10	10.0.0.150
3	□	0.002160	10.0.0.150	10.254.0.10
4	□	0.003304	10.254.0.10	10.0.0.150
5	□	0.003849	10.0.0.150	10.254.0.10
6	□	0.004854	10.254.0.10	10.0.0.150
7	□	0.005123	10.0.0.150	10.254.0.10

Figure 20.32 – Packet capture opened in CS

While in CS, you can use a variety of built-in tools to study the capture, or you can download the file and open it in Wireshark, for better visualization or a more advanced analysis of the data.

To continue with building your skills in packet analysis, I have listed a few more sites where you can find a variety of captures.

## Finding more captures

Here are a few websites that can give you a variety of real network traffic:

- To see a general list of unsorted packet captures, visit <https://wiki.wireshark.org/SampleCaptures>.
- For a small sampling of captures that are ideal for training purposes, visit <http://tcpreplay.appneta.com/wiki/captures.html>.
- NETRESEC, located at <https://www.netresec.com/?page=PcapFiles>, has a nice sampling of public repositories and their respective URLs.
- Chris Sanders is an author who has several capture files available on his GitHub page. Learn more here: <https://chrissanders.org/packet-captures/>.

This is only a partial list of where you can get samples to hone your skills. As you become more involved in analyzing traffic, you will most likely find many more online repositories with sample captures. Visit them, download the captures, and continue to improve your packet analysis skills.

## Summary

In this chapter, we took a look at CS, a tool that allows you to view and analyze packet captures in a browser. We learned some of the ways CS provides the ability to examine captures, many of which are similar to Wireshark. We started by discovering CS and learned ways to modify the preferences, work with captures, and create customized profiles. We then evaluated ways to filter a capture to show only a specific type of traffic, as well as creating a variety of graphs.

In addition, we learned that CS has a rich variety of analysis tools. Tools include **Follow Stream**, **Network Endpoints**, **GeoIP World Map**, **Packet Lengths**, **DNS Activity**, **VoIP Calls**, and **Wireless Networks**, with methods to assess threats. We discovered that, in general, there are many resources for packet captures that you can visit and download a capture file to study and improve your packet analysis skills. We then took a look at [PacketLife.net](http://PacketLife.net), which has an online repository of capture files for download, or an option to open them and analyze them in CS. Finally, we reviewed a few of the locations you can search for packet captures, to further your skills in packet analysis with Wireshark.

## Questions

Now, it's time to check your knowledge. Select the best response to the following questions and then check your answers, which can be found in the *Assessment* appendix:

1. In **Preferences**, \_\_\_\_\_ is where you can customize your column headers. You can use the default layout, remove any of the visible fields as shown, or drag additional columns to where you would like them.
  - A. **Account**
  - B. **Uploads**
  - C. **Decode Window**
  - D. **Capture Index**
2. In addition to the standard filters, you can also search for a specific image by using a \_\_\_\_\_ value.
  - A. Decode
  - B. String
  - C. Hex
  - D. Craft
3. \_\_\_\_\_ are similar to flow graphs in Wireshark, which show endpoints communicating back and forth.
  - A. Ladder diagrams
  - B. Step charts
  - C. VoIP ladders
  - D. Time intervals
4. At the bottom right of a **Network Endpoints** report, you will see a button to select \_\_\_\_\_, which will visually show from where packets originate.
  - A. **Step Charts**
  - B. **Craft**
  - C. **Time Interval**
  - D. **GeoIP World Map**

5. In CS, a \_\_\_\_\_ is a more advanced option that will scan a capture for potentially malicious traffic within the capture.
  - A. Malicious conversation
  - B. Malicious endpoint
  - C. Threat assessment
  - D. Threat map
6. Built within the CS analysis tools is \_\_\_\_\_, which is an open source network analyzer used to monitor traffic for unusual or suspicious activity.
  - A. **HTTP Analysis**
  - B. **Step Charts**
  - C. **Zeek**
  - D. **Decode.io**
7. In CS, you can make modifications to customize your workflow by creating a unique \_\_\_\_\_.
  - A. Profile
  - B. Step chart
  - C. Threat map
  - D. Decode window

## Further reading

Refer to the following links for more information:

- Visit <https://www.qacafe.com/support/cloudshark/user-guide/capture-file-index/#searching-the-capture-file-index> to learn more about running a **DeepSearch** on a capture file.
- For the original malware analysis file, visit <https://www.malware-traffic-analysis.net/2017/01/28/index.html>. Once there, you will find more information on this packet capture of a Windows host infected with malware.

# Assessments

## Chapter 1

1. D. 1990s
2. C. EINSTEIN
3. B. Chat
4. C. Baseline
5. A. Reactive
6. C. IoT devices
7. B. DORA process

## Chapter 2

1. C. sparkline
2. D. promiscuous
3. B. pcap
4. D. Dissectors
5. C. mergecap
6. A. capture engine
7. B. ipconfig

## Chapter 3

1. C. libpcap
2. C. 802.11
3. D. mmdbresolve
4. B. Npcap



5. A. editcap
6. B. Transum
7. C. **News**

## Chapter 4

1. A. Clear
2. D. previously displayed packet
3. D. Colorize Packet List
4. C. View
5. B. Edit
6. C. Status Bar
7. D. Show Packet in New window

## Chapter 5

1. A. LAN
2. C. `manuf.txt`
3. B. Output
4. D. single mode
5. A. `.pcapng`
6. C. UTP
7. C. conversation
8. D. Statistics

## Chapter 6

1. D. Default
2. B. **Edit | Appearance**
3. C. red
4. A. **Edit | Preferences**
5. B. **Edit**

6. B. profile
7. C. red

## Chapter 7

1. B. green
2. C. Value
3. D. udp port 53
4. A. Expression
5. D. ...and not Selected
6. B. BPF
7. A. Prepare a Filter

## Chapter 8

1. C. Session
2. A. Transport
3. D. Presentation
4. B. Well-known
5. D. Segment
6. C. IP
7. A. MAC

## Chapter 9

1. B. socket
2. B. 724 and 725
3. D. PSH
4. C. 20
5. C. DHCP
6. D. options
7. D. 512

## Chapter 10

1. C. `frame.marked==1`
2. B. Window scale
3. C. SACK
4. D. FIN
5. C. 936
6. A. EOL
7. D. Timestamp

## Chapter 11

1. C. Network control
2. B. Class B private IPv4 address
3. D. 128
4. D. Hop count
5. C. 3
6. C. 0xfbb74
7. A. 65,535

## Chapter 12

1. C. Unreachable
2. D. Deprecated
3. B. Parameter problems
4. A. Type 3 for ICMP and Type 1 for ICMPv6
5. B. Ping
6. C. Packet too big
7. D. Type 3 and Type 9

## Chapter 13

1. B. Root
2. C. authoritative

3. D. PTR
4. D. truncated
5. B. DoS
6. A. name
7. C. nslookup

## Chapter 14

1. C. Discover
2. B. Request
3. D. SLAAC
4. B. 547
5. D. hlen
6. B. Requested IP address
7. A. ciaddr

## Chapter 15

1. C. POST
2. D. Apache
3. D. 2.0
4. C. Keep-alive
5. B. 1 hour
6. B. MIME
7. A. Cache

## Chapter 16

1. C. Data link
2. D. 1 and 2
3. D. RARP
4. C. Gratuitous ARP
5. A. Storm

6. C. Opcode
7. B. IDS/IPS

## Chapter 17

1. A. Latency
2. D. Throughput
3. C. Intelligent Scrollbar
4. B. Keep-alive
5. C. Note
6. B. Packet loss
7. A. Previously Displayed Packet

## Chapter 18

1. C. Conversation
2. A. **Protocol Hierarchy**
3. B. .pcapng
4. D. **Mark**
5. C. **HTTP**
6. D. Asterisk
7. A. **Statistics**

## Chapter 19

1. C. **Protocol Hierarchy**
2. B. Flow graph
3. A. I/O graph
4. D. Configuration profile
5. C. TCP stream (tcptrace)
6. A. Throughput
7. B. WS

## Chapter 20

1. D. **Capture Index**
2. B. String
3. A. Ladder diagrams
4. D. **GeoIP World Map**
5. C. Threat assessment
6. C. **Zeek**
7. A. Profile



# Index

## Symbols

11-field TCP header  
examining 219, 220

## A

Access Control List (ACL)  
about 323  
using 6  
acknowledgment (ACK) flag set 396  
acknowledgment (ACK) packet  
about 213  
finalizing 251, 252  
acknowledgments (ACKs) 242, 443  
active attacks 15  
Address Resolution Protocol (ARP)  
about 157, 193, 195, 271, 298, 370  
purpose 418  
replacing, with NDP in IPv6 423  
reversing 427, 428  
role 418  
types, exam 427  
used, for resolving addresses 194  
working, on behalf of 430

Advanced Research Projects Agency  
Network (ARPANET) 330  
analysis phase 41  
analysis tools  
conversations, viewing 538-540  
evaluating 537  
HTTP analysis, exploring 541  
packet lengths, viewing 540, 541  
stream, following 538-540  
threats, monitoring 542-544  
VoIP activity, viewing 540, 541  
wireless traffic, exploring 541  
androiddump 61  
anti-ARP software 436  
anycast 286  
Apache 393  
Application layer  
about 183  
evaluating 185  
PDU 186  
protocols, exploring 185  
Application Programming  
Interface (API) 36, 50



- ARP attacks
  - defending, against 435, 436
  - tools 435
  - tools and techniques, to
    - prevent 435, 436
  - versus defense methods 432
  - versus tools 432
- ARP cache
  - investigating 421, 422
- ARP cache table 15
- ARP fields
  - exploring 423
- ARP header fields
  - breaking down 425-427
- ARP headers
  - exploring 423
- arpoinson 435
- ARP reply
  - evaluating 424, 425
  - identifying 423
  - viewing 420
- ARP request
  - identifying 423
  - inspecting 424
  - viewing 420
- arpspoof 435
- ARP spoofing
  - about 15
  - cache poisoning 15, 16
  - discovering 432, 433
- ARP storm
  - reviewing 433, 434
- artificial intelligence (AI) 5

- Asynchronous Transmission
  - Mode (ATM) 429
- authoritative 333
- Automatic Private IP Addressing (APIPA) 282

## B

- baseline
  - about 123
  - planning 123
  - saving 125, 126
- baselining
  - significance 123
- basic service set identifier (BSSID) 541
- Berkeley Internet Name Domain (BIND) 331
- Berkeley Packet Filter (BPF) syntax
  - about 165
  - recognizing 165
- bigFlows.pcap
  - reference link 243, 397, 467, 493
- bigFlows.pcap packet capture
  - reference link 271
- Blaster worm virus alert
  - reference link 176
- Bluetooth 112
- bookmark icon 156
- bookmarks
  - capture filter, saving to 168
  - using 161, 162
- bounded signals 110
- broadcast 119

Building Automation and Control  
  Network (BACnet) 496

buttons

  crafting 139, 140

bytes

  exporting 78-80

  sequencing 222

## C

Campus Area Networks (CANs) 108, 109

capinfos 60

capture, comments

  adding 482

  file comments, preserving 482-484

  packet comments, preserving 482-484

  saving 484-486

  viewing 484-486

capture, components

  exporting, ways 477

  specified packets, selecting 478-480

  various objects, exporting 480-482

captured data

  displaying 38

captured traffic

  analyzing 124, 125

capture engines

  libpcap 54

  using 36

  WinPcap, examining 54

capture files

  Save as option, using 476, 477

  saving, options 474, 475

  collection, adding 529, 530

  profile, creating 531, 532

  sharing, in CS 528, 529

capture filters

  bookmarking 168

  crafting 166, 167

  creating 162-164

  examining 158

  modifying 164-165

  using 176

capture issues

  graphing 497, 498

capture options

  about 113

  input, providing 114

  output, directing 114-116

  selecting 116-118

capture, traffic

  breaking down, by protocol 472, 473

  dissecting, by IP address 467-470

  minimizing, by port number 471, 472

  narrowing down, by

    conversations 470, 471

  subsetting, ways 466, 467

CapType 60

Change mode Berkeley Packet

  Filter (ChmodBPF) 51

Ciscodump 61

client 393

Client Hardware Address (chaddr) 367

CLI tools

  tshark, exploring 43, 44

  using, with Wireshark 42

Cloudflare 393

- CloudShark (CS)
  - about 522
  - capture files, working with 526-528
  - captures, uploading 525, 526
  - discovering 522
  - features 522
  - preferences, modifying 523-525
  - reference link 522
- coaxial 111
- codec plugins 60
- code values
  - defining 317
  - reviewing 315, 316
- coloring rules 447-449
- colors
  - adjusting 141
  - refining 145-147
- columns
  - adding 141-143
  - adjusting 141
  - deleting 141-143
  - editing 141-143
- Command and Control (C&C) 351
- command-line interface (CLI) 422
- Comma Separated Values (CSV) 80, 122
- comments
  - adding 148
  - attaching, to files 148
  - saving 149, 150
  - viewing 149, 150
- congestion control 213
- Congestion Experienced (CE) 276
- congestion window (CWND) 213, 255
- Congestion Window Reduced (CWR) 276

- connection methods
  - tracking 394
- connection types
  - data transfer, pipelining 396
  - evaluating 395
  - non-persistent connection, using 395
  - persistent connection, offering 395, 396
- Constrained Application Protocol (CoAP) 185
- content-addressable memory (CAM) 433
- conversations
  - types 121
  - versus endpoints 119
  - viewing 120
- cookies
  - about 397
  - authentication 397
  - used, for maintaining state 396, 397
  - user experience (UX), personalizing 397
- Coordinated Universal Time (UTC) 94
- copper 111
- copy
  - options 85-87
- Cyclic Redundancy Check (CRC) 197, 200

## D

- data
  - acknowledging 225
  - displaying, with filters 533, 534
  - viewing, with graphs 534-536
- data link connection identifier (DLCI) 429

- Data Link layer
  - about 183
  - address, describing 197
  - examining 196
  - PDU, investigating 196
  - protocols, investigating 196
- Data Over Cable Service Interface Specification (DOCSIS) 111
- data payload
  - commands, sending covertly 305
  - error, reporting 303
  - ICMP echo request, viewing 302
  - investigating 302
  - watermark, creating 304, 305
- decoders 37
- defense methods
  - versus ARP attacks 432
- Denial of Service (DoS)
  - attack 15, 351, 433
- developer tools
  - using 99, 100
- DHCP Acknowledgment (DHCP ACK) 369
- DHCP example
  - about 377
  - discover packet, broadcasting 379, 380
  - IP address, releasing 377-379
  - IP address, requesting 382, 383
  - offer, acknowledging 383, 384
  - offer, delivering to DHCP client 380, 382
- DHCP field values
  - Boot File (file) 374
  - Client Hardware Address (chaddr) 374
  - Client IP Address (ciaddr) 374
  - DHCP options 374
  - examining 373, 374
  - Hardware Length (hlen) 373
  - Hardware Type (htype) 373
  - Hops (hops) 373
  - Operation Code (op) 373
  - Preboot Execution Environment (PXE) 374
  - Relay Agent IP Address (giaddr) 374
  - Seconds (secs) 374
  - Server Host Name (sname) 374
  - Server IP Address (siaddr) 374
  - Transaction Identifier (XID) 373
  - Your IP Address (yiaddr) 374
- DHCP header
  - dissecting 372
- DHCP lease time
  - defining 370, 371
- DHCP messages 375, 376
- DHCP options
  - comparing 376, 377
- DHCP relay agent
  - using 361, 362
- DHCP server
  - searching for 368
- DHCP states
  - about 366
  - bound 366
  - initiate 366
  - rebind 367
  - renew 366
  - request 366
  - select 366
  - transaction ID, defining 367

- DHCPv6
  - options 364
- Differentiated Services Code Point (DSCP) 275
- Differentiated Services (DiffServ) field
  - about 271
  - Explicit Congestion Notification (ECN) 276, 277
  - Quality of Service (QoS) 274, 275
- Digital Subscriber Line (DSL) 197
- Disk Image (DMG) 68
- display filter (dfilter) test 60
- display filters
  - bookmarks, using 161, 162
  - comprehending 159
  - editing 160, 161
  - investigating 157
  - using 175, 176
- dissector plugins 59
- dissectors 37
- Distributed Denial of Service (DDoS) attacks 318, 393,
- DNS packet
  - header, examining 339, 340
  - packet structure, dissecting 343
  - query section, outlining 344, 345
  - reviewing 338
- DNS packet, header structure
  - flags, comparing 340, 341
  - opcode 342
  - recursion, requesting 343
- DNS queries
  - evaluating 345
- DNS responses
  - caching 346
  - evaluating 345
  - times, calculating 347-351
- DNS Security Extensions (DNSSEC) 353
- DNS servers
  - acting, as authority 333
  - recursive server, using 333, 334
  - testing, with nslookup 351, 352
  - types 333
- DNS Stateful Operations (DSOs) 342
- Domain Name Service (DNS) 360
- Domain Name System (DNS)
  - about 139, 189, 234, 497
  - cache, poisoning 353
  - defending 353
  - IP address, mapping 330
  - purpose, recognizing 330
  - reflecting on 330, 331
  - root 332
  - securing 353
  - transporting 335
  - types 336, 337
- Domain Name System (DNS),
  - transport layer
    - complete transfer, ensuring 335
    - speedy resolution, providing 335
- DORA process
  - about 366-368
  - DHCP states 366
  - IP address, leasing 370
  - IP address, obtaining 367

- DORA process, phases
  - DHCP Acknowledgment (ACK) 368
  - DHCP Discover 367
  - DHCP Offer 367
  - DHCP Request 368
- download options
  - evaluating 67, 68
- dsniff 435
- Duplicate Acknowledgement (Dup ACK)
  - about 258
  - observing 450-452
- Dynamic Host Configuration Protocol (DHCP)
  - about 12, 156, 192, 234, 282, 334, 428, 496
  - client's IP address, configuring 361
  - IPv6 addresses, working with 363
  - purpose, recognizing 360
  - security issues, addressing 365
- Dynamic Host Configuration Protocol (DHCP), elements
  - DHCP clients 360
  - DHCP relay agent 360
  - DHCP servers 360

## E

- ECN-Capable Transport (ECT) 276
- ECN Echo (ECE) 276
- editcap 60
- Edit menu
  - discovering 84
- Electromagnetic Interference (EMI) 111

- encapsulation process
  - data, viewing 199
  - exploring 198
  - frame, forming 200
  - packet, characterizing 200
  - segment, identifying 199
- endpoints
  - versus conversations 119
- Enhanced Interior Gateway Routing Protocol (EIGRP) 119
- Enhanced Packet Analyzer (EPAN) 118, 155
- Enhanced Packet Analyzer (EPAN), components
  - display filters 38
  - dissector-plugins 38
  - dissectors 37
  - protocol tree 37
- EOL option
  - grasping 254
- Ethereal 32, 37
- Ethernet II 196
- Ethernet II header
  - expanding 216, 217
- Ettercap 137, 435
- etwdump 61
- Event Trace Log (ETL) 61
- evil bit 289
- expectational acknowledgment 226
- expert information
  - column headers, viewing 456, 457
  - data, sorting 458, 459
  - discovering 454-456
  - organizing 458

- severity, assessing 457, 458
- values, searching for 459-461
- expert system 10
- Explicit Congestion Notification (ECN) 274
- expression builder
  - about 168-170
  - custom filter, building 170, 171
- extensions
  - selecting 59
- External Data Representation (XDR) 187

## F

- fake Ethernet packets 55
- fiber optic
  - multimode (MMF) 112
  - single mode (SMF) 112
  - using 112
- field occurrence
  - using 143-145
- File menu
  - exploring 76
  - file, closing 77
  - file, opening 77
  - file, saving 77, 78
  - packets, exploring 78, 80
  - packets, printing 82, 83
  - bytes, exporting 78, 80
  - objects, exporting 81, 82
  - Wireshark, closing 82, 83
- files
  - comments, attaching to 148
- File Transfer Protocol (FTP) 165, 390
- file type plugins 59
- filters
  - applying 175
  - data, displaying with 533, 534
  - files, comparing 156, 157
- filter shortcuts
  - additional options, viewing 175
  - discovering 172
  - embracing 172-174
- finis (FIN) packets 191, 211
- finish (FIN) 412
- firewall
  - about 6, 435
  - with ACL 7
- firewall rules
  - configuring, to prevent malicious activity 318
  - malicious ping sweeps, sending 318-320
  - service, denying 322, 323
  - traffic, redirecting 320, 321
- flags
  - about 374
  - Broadcast flag 374
  - client 340
  - comparing 340
  - server 341
  - Reserved flag 374
- flow control 213, 256
- font
  - adjusting 141
  - refining 145-147
- footprinting 14
- four-field UDP header
  - discovering 236
- fragmentation, fields
  - flags 278

fragment offset 278  
identifier 278  
Frame Check Sequence (FCS) 200, 300  
frame formation  
    demonstrating, in Wireshark 201, 202

## G

general information  
    viewing 493, 494  
General Public License (GPL) 32  
Generic Routing Encapsulation  
    (GRE) 292  
global unicast 286  
GNU General Public License 58  
Graphical User Interface (GUI) 50, 155  
graphs  
    data, viewing with 534, 536  
gratuitous ARP  
    issuing 430

## H

header values  
    calculating 233, 234  
    TCP options, adding 233  
    viewing 232, 233  
hexadecimal (hex) 534  
High-Level Data Link Control  
    (HDLC) 196  
host  
    obscuring 431  
HTTP.cap  
    reference link 531  
HTTP header  
    checking 219

HTTP methods  
    recognizing 394  
HTTP.pcap  
    reference link 403  
HTTP request  
    MIME header, viewing 399  
    request line 399  
    viewing 398  
HTTP response  
    about 400  
    format 400  
    MIME header 401, 402  
    status codes 401  
HTTP Secure (HTTPS) 392  
HTTP stream  
    cache, controlling 410  
    client, responding to 408-410  
    conversation, beginning 406  
    conversation, ending 412  
    data, requesting 406, 407  
    following 402-405  
    object, exporting 411  
HTTP versions  
    about 393  
    HTTP 1.0 393  
    HTTP 1.1 393  
    HTTP 2.0 393  
HyperText Markup Language  
    (HTML) file 139, 390, 497  
HyperText Transfer Protocol (HTTP)  
    about 389, 390  
    state, maintaining with cookies 396-398  
Hypertext Transport Protocol  
    (HTTP) 81, 452



**I**

- ICMP echo request
  - viewing 302
- ICMP header 299-301
- ICMP header, fields
  - checksum (16-bit) 300
  - code (8-bit) 300
  - type (8-bit) 300
- ICMP messages
  - about 301
  - providing, information with
    - ICMPv6 312-315
  - query messages, issuing 311
  - sending 307, 308
- ICMP packets
  - required types, allowing 323
- ICMP ping sweep 318
- ICMP report errors
  - examples, viewing 308-310
  - packet size, monitoring 310
  - reviewing 308
- ICMP type
  - reviewing 315, 316
- ICMPv6
  - dissecting 305
  - used, for providing information 312
- ICMPv6 type
  - defining 317
- IEEE 802.3 197
- input/output (I/O) graphs
  - adding 503
  - column headers, reviewing 502, 503
  - copying 504
  - creating 499
  - duplicate ACKs, graphing 501, 502
  - errors, examining 500, 501
  - other options, exploring 504-506
  - removing 503, 504
  - settings, modifying 502
- Integrated Services Digital Network (ISDN) 197
- Intelligent Scrollbar
  - about 10, 30
  - exploring 449, 450
  - reference link 449
- International Organization for Standardization (ISO) 182
- International Telegraph and Telephone Consultative Committee (CCIT) 182
- Internet Assigned Numbers Authority (IANA) 253, 426
- Internet Control Message Protocol (ICMP)
  - about 143, 193, 195, 271, 279, 298, 347, 418
  - dissecting 305
  - reference link 194
  - reviewing 305, 306
  - tasks 299
  - versions 299
- Internet Control Message Protocol (ICMP), protocols
  - Address Resolution Protocol (ARP) 298
  - Internet Protocol (IP) 298
- Internet Control Message Protocol version 6 (ICMPv6) 370, 423
- Internet Group Message Protocol (IGMP) 312-315

- Internet of Things (IoT) 4, 108, 185, 442
- Internet Protocol Flow Information Export (IPFIX) 264
- Internet Protocol Identification 142
- Internet Protocol (IP)
  - about 298, 393, 467
  - purpose 271
  - used, for routing packets 193, 194
- Internet Protocol (IP) addresses 493
- Internet Protocol (IP) Maximum Transmission Unit (MTU) 255
- Internet Service Provider (ISP) 352
- Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) 292
- intrusion detection system (IDS) 6, 323, 435
- intrusion prevention systems (IPSs) 435
- Inverse Address Resolution Protocol (InARP)
  - evaluating 428, 429
- IP address
  - checking for, duplication 370
  - DHCP server, searching for 368
  - leasing 370
  - mapping 330
  - obtaining 367
  - offering 368
  - parameters, confirming 369
  - requesting 369
  - supplying, for packet 195, 196
- IP header
  - viewing 217
- IPv4
  - about 194, 272
  - data, fragmenting 277, 278
  - DiffServ field 274
  - header checksum field 280
  - modifying options 282
  - protocol field 280
  - Time to Live (TTL) field 279
- IPv4 addresses
  - about 280, 497
  - classes 281
  - private 282
  - special 282
- IPv4 header
  - about 273
  - length 274
  - version 274
- IPv4 preferences
  - adjusting 290
  - reviewing 287-289
- IPv6
  - about 194, 272, 497
  - addressing, requirements 286
  - address types, comparing 286
  - enhancements 282
  - exploring 282
  - flow, managing 284, 285
  - NDP, used for replacing ARP in 423
- IPv6 addresses
  - working with 363
- IPv6 header
  - about 283
  - fields 283
  - flow label 284
  - hop limit 285
  - next header 285

- payload length 285
- traffic class 284
- version 284

- items
  - copying 84

## J

- JavaScript Object Notation (JSON) 80
- Joint Photographic Experts Group (JPEG) 390, 536

## K

- keep-alive packet
  - observing 452, 453
  - reference link 452

## L

- latency
  - about 442, 443
  - computing 443-445
- latency issues
  - analyzing 442
- layout
  - appearance, altering 130-132
  - changing 133, 134
  - packet list settings, adjusting 134, 135
  - personalizing 130
  - status bar settings, adding 135
- libpcap 54
- Light-Emitting Diode (LED) 112

- Lightweight Directory Access Protocol (LDAP) 337
- link local 286
- Linux
  - Wireshark, deploying on 51, 52
- Local Area Network (LAN) 108, 109, 194, 216, 272, 298, 353, 361
- logical addresses 195
- Loki tool
  - using 305

## M

- macOS
  - Wireshark, installing on 51
- malicious ping sweeps
  - sending 318-320
- Man-in-the-Middle (MitM) attack 15
- maximum segment size (MSS)
  - about 277
  - defining 255
- maximum transmission unit (MTU) 272, 277, 310
- Media Access Control (MAC) address
  - about 216, 271, 298, 363, 534
  - resolving 419
- mergcap 60
- Message Queuing Telemetry Transport (MQTT) 109, 185
- Meta Analysis and Tracing Engine (MATE) 59
- MMDBResolve 60
- multicast 119 286
- Multiprotocol Label Switching (MPLS) 110

Multipurpose Internet Mail  
Extensions (MIME) header  
about 399  
viewing 399

## N

name server (NS) 333  
Negative Acknowledgement  
(NAK) 367, 369  
Neighbor Discovery Protocol (NDP)  
about 312, 418  
used, for replacing ARP in IPv6 423  
Neighbor Solicitation (NS) 423  
network address translation (NAT) 292  
network architectures  
reviewing 108  
network bindings  
examining 202  
Network Driver Interface  
Specification (NDIS) 55  
Network Interface Card (NIC) 7,  
118, 197, 280 351, 363  
network layer  
about 183, 193, 270  
PDU 193  
protocols 193  
Network Time Protocol (NTP) 90, 360  
network traffic  
analyzing 154, 155  
capture engine, using 36  
capturing 155  
filtering 154  
gathering 34, 35  
promiscuous mode, capturing 35  
viewing 155

network traffic, filtering options  
capture filters 154  
display filters 154  
expressions 154  
shortcuts 154  
network types  
comparing 108  
nginx 393  
non-persistent connection  
using 395  
No-Operation (NOP)  
about 233  
using 254, 255  
Npcap 36, 55  
Npcap features  
recognizing 55  
nslookup  
used, for testing DNS server 351, 352

## O

objects  
exporting 81, 82  
offset  
calculating 227  
Open Systems Interconnection  
(OSI) model  
about 111, 418  
framework, developing 182  
framework, using 183  
network layer 270  
overview 182  
PDUs, discovering 183-185  
protocols, discovering 183-185  
purpose of layer, discovering 183-185  
operating system (OS)  
about 75, 302, 421  
support, discovering for 50

**Operation Code (Opcode)**

- about 340, 342, 423

**Organizationally Unique**

- Identifier (OUI) 363

**outdated type values**

- usage, discouraging 316, 317

## P

**packet analysis**

- about 34
- beneficiaries, recognizing 8
- benefits 4, 8
- devices, evaluating 6
- network traffic, capturing 7, 8
- network traffic, sniffing 18
- packet sniffers 5
- real world example 18
- reviewing 4
- sniffing tools 5
- sniffing tools, functionality 5
- traffic analysis, on LAN 17
- usage, identifying 17
- usage, outlining 19

**packet analysis, benefits**

- comments, subsetting 11
- developers, assisting 8
- export, subsetting 10
- hackers, arming with information 14
- Intelligent Scrollbar 10
- network monitoring, by network administrators 9
- save, subsetting 10
- security analysts, alerting to threats 13
- students, educating on protocols 12
- traffic, subsetting 10

**packet analysis, phases**

- about 34
- captured data, displaying 38
- network traffic, gathering 34
- packet capture, analyzing 41
- raw bits, decoding 37

**packet analysis, uses**

- active threat monitoring 20
- IoT devices, testing 20
- latency issues, troubleshooting 19, 20
- network baseline process 21
- proactive threat monitoring 20
- reactive threat monitoring 20
- threat monitoring 20
- threats monitoring 21

**packet bytes panel 40****packet capture (pcap)**

- about 4, 36, 54
- examining 544, 545
- finding 546
- locating 544
- reference link 546

**packet comments**

- inserting 148, 149

**packet details panel 39****PacketLife.net**

- reference link 544

**packet list panel 39****packet loss**

- about 442, 443
- experiencing 446

**packets**

- colorizing 97, 98
- exporting 78, 80
- finding 84
- ignoring 89
- locating 87
- marking 88

- printing 82, 83
- reloading 100
- routing, with IP 193
- packet structure
  - dissecting 343
- Packet Too Big error
  - managing 311
- panel layout
  - modifying 41
- passive attacks
  - outlining 14
- Path MTU Discovery (PMTUD) 311
- persistent connection
  - offering 395
- Personal Area Networks (PANs) 108
- Physical layer
  - about 183
  - exploring 197
  - PDU, describing 198
  - protocols, exemplifying 197
- pipelining 396
- Plain Old Telephone Service (POTS) 110
- plugins
  - selecting 59
- Point-to-Point Tunneling Protocol (PPTP) 189
- portable application (app)
  - utilizing 53, 54
- Portable Document Format (PDF) 83, 498
- Portable Network Graphics (PNG) 536
- port addressing
  - dynamic port 192
  - ephemeral port 192
  - private port 192
  - registered ports 192
  - well-known ports 192

- Precision Timing Protocol (PTP) 90
- premade virtual images
  - downloading 52
- Presentation layer
  - about 183
  - exploring 186, 187
  - PDU, investigating 187, 188
  - protocols, investigating 187, 188
- profile
  - customizing 136-139
- promiscuous mode
  - about 7
  - capturing 35
- Protocol Data Unit (PDU)
  - about 222
  - Application layer 186
- protocol effectiveness
  - assessing 494-497
  - assessing, options 496
- protocol preferences
  - editing 287
- protocols
  - dissecting 44, 45
- proxy 430
- proxy ARP
  - about 430
  - acting 431, 432
- Public Switched Telephone Network (PSTN) 197

## Q

- Quad A 337
- Quality of Service (QoS) 274
- query section
  - outlining 344, 345

## R

- radiotap headers
  - troubleshooting 56, 57
- random packet generator (randpkt) 61
- raw bits
  - decoding 37
- rawshark 60
- Real-time Transport Protocol (RTP) 189, 541
- Receiver Window (Rwnd) 213
- recursion
  - requesting 342, 343
- recursive 333
- recursive server
  - using 333
- reliable data protocol (RDP) 210
- Remote Procedure Call (RPC) 189
- reordercap 60
- Request for Comment (RFC) 5, 37
  - 194, 253, 299, 331, 393, 457
- request messages
  - versus response messages 398
- Reset (RST) 266
- Resource Records (RR)
  - DNS types 336, 337
  - structure, examining 337, 338
  - types and classes, comparing 336
- resources, Wireshark
  - help options 66
  - news 65, 66
- retransmissions
  - issuing 454
  - recognizing 454
- Reverse Address Resolution Protocol (RARP) 427
- ring buffer
  - using 116

- Round-Trip Time (RTT)
  - about 234, 443-445
  - assessing 514, 515
- router 6
- Router Solicitation 306
- Routing Information Protocol (RIP) 192

## S

- Sample Captures
  - reference link 546
- Sampled Flow (sFlow) 347
- Scalable Vector Graphics (SVG) 536
- Secure/Multipart Internet Mail Extensions (S/MIME) 187
- Secure Neighbor Discovery (SEND) 436
- Secure Shell (SSH) 61
- Secure Socket Layer (SSL) 187
- Security Operations Center (SOC) 21
- Selective Acknowledgements (SACK) 212
  - permitting 257-259, 507
  - viewing 510-512
- sequence field values
  - reviewing 225
- sequence numbers
  - about 222-225
  - synchronizing 222, 223
- server 393
- server identifier 369
- Server Message Block (SMB) 496
- service
  - denying 322-323
- Session Initiation Protocol (SIP) 337
- Session layer
  - about 183, 188, 189
  - PDU, recognizing 189
  - protocols, recognizing 189

- Session layer, services
    - authentication 189
    - authorization 189
    - checkpointing 189
  - Shielded Twisted Pair (STP) 111
  - Simple Authentication and Security Layer (SASL) 185
  - Simple Mail Transfer Protocol (SMTP) 390
  - simple moving average (SMA) 503
  - Simple Network Monitor Protocol (SNMP) MIBs 60
  - single sign-on (SSO) 397
  - single TCP frame
    - Ethernet II header, expanding 216, 217
    - exploring 214
    - frame details, describing 216
    - HTTP header, checking out 219
    - IP header, viewing 217
    - TCP header, navigating 218
  - single UDP frame
    - elements 235
  - sliding window 230
  - sniffing 5
  - Software Development Kit (SDK) 61
  - SPeeDY (SPDY) 393
  - spurious retransmission
    - sending 454
  - Sshdump 61
  - StateLess Address AutoConfiguration (SLAAC)
    - about 306, 363
    - employing 363
  - static ARP entries 435
  - Statistics menu
    - capture issues, graphing 497, 498
    - discovering 492, 493
    - general information, viewing 493, 494
    - protocol effectiveness, assessing 494-497
  - status code
    - 1xx Informational 401
    - 2xx Success 401
    - 3xx Redirection 401
    - 4xx Client Error 401
    - 5xx Server Error 401
    - 200 OK 401
    - 301 Moved Permanently 401
    - 400 Bad Request 401
    - 404 Not Found 401
    - about 401
  - Stevens graph 507, 508
  - stream
    - about 39, 118
    - isolating 243-245
  - stream control transmission
    - protocol (SCTP) 210
  - stream index 39
  - Structured Query Language (SQL) 15
  - Switched Port Analyzer (SPAN) 18
  - Synchronization Acknowledgement (SYN, ACK)
    - about 223, 242
    - returning 250, 251
  - synchronization (SYN) 242, 406
  - synchronization (SYN) flag 222
  - synchronization (SYN) packet
    - sending 248-250
  - syntax checker
    - working 156
- ## T
- tailored configuration profile
    - creating 136
  - TCP handshake
    - packet, marking 246, 248



- TCP header
  - about 39
  - navigating 218
  - reference link 214
- TCP options
  - about 252-254
  - EOL option, grasping 254
  - MSS, defining 255
  - NOP, using 254, 255
  - SACK, permitting 257-259
  - timestamps, using 259, 260
  - Window size (WS), scaling 256, 257
  - Selective Acknowledgements (SACK) 212
  - window scaling 212
- TCP ports
  - about 392
  - conversation completeness 221
  - destination port 16-bit 220
  - exploring 220
  - source port 16-bit 220
  - stream index 221
  - TCP segment length 222
- TCP protocol preferences
  - about 260, 262
  - modifying 262-264
- TCP Segmentation Offload (TSO) 289
- TCP stream 122
- TCP teardown
  - overview 264-266
- TCP three-way handshake
  - dissecting 242, 243
- TCP three-way handshake, packets
  - ACK packet, finalizing 251, 252
  - identifying 248
  - SYN-ACK packet, returning 250, 251
  - SYN packet, sending 248-250
- tcptrace graph
  - about 508
  - outlining 508, 509
  - selective ACKs (SACKs), viewing 510-512
- Teredo 292
- Terminal Ethereal (Tethereal) 42
- Terminal Wireshark (tshark)
  - example, running 43
  - exploring 43
- text2pcap 60
- throughput
  - about 442, 443
  - determining 512, 513
  - measuring 446
- time
  - shifting 90
- time reference
  - setting 89
- time sequence graphs
  - using 506
- timestamps
  - using 259, 260
- Time to Live (TTL) 308, 336
- time values
  - significance 446
- Top-Level Domain (TLD) 332
- traffic
  - capturing 123, 124
  - redirecting 320, 321
- traffic analysis
  - reference link 542
- Traffic Class field 271
- Transaction Identifier (XID) 367
- Transmission Control Protocol
  - Synchronization (TCP SYN) 139

## Transmission Control Protocol (TCP)

- about 158, 189, 191, 271, 334, 467
- bytes, sequencing 222
- connection, establishing 211-213
- connection, maintaining 211-213
- data, acknowledging 225
- flags 228, 229
- flow, managing 213, 214
- header values, viewing 232
- outlining 306, 307
- states, listing 191
- window size, dissecting 229

## Transmission Control Protocol (TCP) features

- connection-oriented 211
- describing 211
- end-to-end reliability 211

## Transmission Control Protocol

- (TCP), stream graphs
- comparing 506
- Round-Trip Time (RTT),
  - assessing 514, 515
- throughput, determining 512
- time sequence graphs, using 506
- window scaling graph,
  - evaluating 515, 516

## transmission errors, indications

- about 450
- duplicate ACKs, observing 450-452
- keep-alive packets, observing 452, 453
- retransmissions, issuing 454

## Transport layer

- about 183, 190
- port addressing, providing 192
- protocols versus PDU 190
- protocols 210
- reviewing 210, 211

- reliability, providing with TCP

- protocol 191, 192

- timely delivery, ensuring with UDP 192

## Transport Layer Security

- (TLS) 80, 187, 392

## Transport Options for UDP

- reference link 211

## transum 59

## tree statistics plugins 59

## Trivial File Transfer Protocol

- (TFTP) 192, 234

## tunneling protocols 291, 292

## twisted pair cabling 111

# U

## UDP header fields

- analyzing 236

- Checksum 16-bit 237

- Destination Port 16-bit 236

- Length 16-bit 237

- Source Port 16-bit 236

## unbounded signals 110

## unicast 119, 286

## Uniform Resource Identifier (URI) 391

## Uniform Resource Locator (URL) 391

## Uniform Resource Name (URN) 391

## Unix

- Wireshark, running on 50

## Unshielded Twisted Pair (UTP) 111

## User Access Control (UAC) 58

## User Account Control (UAC) 55

## User Datagram Protocol (UDP)

- about 61, 122, 190, 234, 243,

- 303, 334, 366, 468, 497

- single UDP frame elements 235

- used, for ensuring timely delivery 192

- user guide
  - obtaining 61

## V

- view
  - refreshing 98
- View menu
  - display, modifying 96, 97
  - exploring 91
  - interface, enhancing 91, 92
  - names, resolving 95, 96
  - time, displaying 93, 94
- Virtual Local Area Networks (VLANs) 123
- Virtual Machine (VM) 52
- VMware network interface 28
- Voice over Internet Protocol (VoIP)
  - 189, 234, 271, 330, 471
- volumetric DDoS 321

## W

- web page
  - client role, versus server role 392, 393
  - connection, creating 392
  - dissecting 390, 391
  - hostname 391
  - path 391
  - target, identifying 391
- Wide Area Networks (WANs)
  - 108, 110, 429
- Wi-Fi 112
- window scaling 212
- window scaling graph
  - evaluating 515, 516
- window size (WS)
  - about 255
  - calculating 230
  - dissecting 229
  - flow, controlling 230
  - scaling factor 231, 232
  - scaling 256, 257
- WinPcap
  - about 36, 54
  - URL 54
- Wireless Local Area Network (WLAN) 155
- wireless networks 112
- Wireless Personal Area Network (WPAN) 108
- Wireshark
  - about 22, 27
  - CLI tools, using with 42
  - closing 82, 83
  - deploying, on Linux 51, 52
  - features 23
  - frame formation, demonstrating
    - 201, 202
  - installing, on macOS 51
  - resources 65, 66
  - running, on Unix 50
  - tools, selecting 60
  - using, on Windows 50
  - working, on other systems 52
- Wireshark authors
  - recognizing 32, 33
- Wireshark installation, on Windows
  - about 58
  - capture engine, checking 63
  - components, selecting 58, 59
  - install location, selecting 62

- shortcuts, creating 62
- starting 58
- USB interface, using 64
- using 64
- Wireshark setup, completing 64
- Wireshark interface
  - examining 28
  - features 30
  - information, finding 34
  - keyboard shortcuts, discovering 31, 32
  - notes information, reading 31
  - streamlining 28, 30
- Wireshark, starting from command line
  - reference link 44
- Wireshark welcome screen
  - file, selecting 74, 75
  - opening 74
  - traffic, capturing 75
- work area
  - personalizing 90

## Y

- YAML Ain't Markup Language
  - (YAML) 86
- Yet Another Markup Language
  - (YAML) 122





Packt . com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

## Why subscribe?

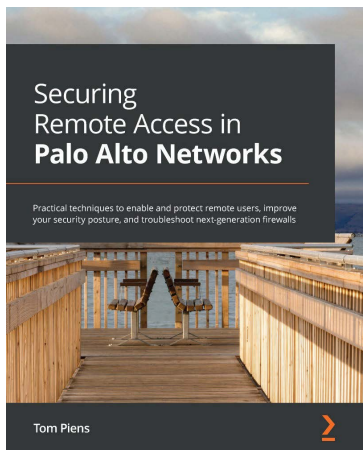
- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [packt . com](http://packt.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [customercare@packtpub . com](mailto:customercare@packtpub.com) for more details.

At [www . packt . com](http://www.packt.com), you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

# Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

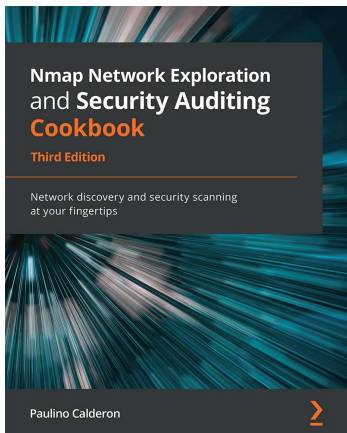


## **Securing Remote Access in Palo Alto Networks**

Tom Piens aka 'reaper'

ISBN: 9781801077446

- Understand how log forwarding is configured on the firewall
- Focus on effectively enabling remote access
- Explore alternative ways for connecting users and remote networks
- Protect against phishing with credential detection
- Understand how to troubleshoot complex issues confidently
- Strengthen the security posture of your firewalls



## **Nmap Network Exploration and Security Auditing Cookbook - Third Edition**

Paulino Calderon

ISBN: 9781838649357

- Scan systems and check for the most common vulnerabilities
- Explore the most popular network protocols
- Extend existing scripts and write your own scripts and libraries
- Identify and scan critical ICS/SCADA systems
- Detect misconfigurations in web servers, databases, and mail servers
- Understand how to identify common weaknesses in Windows environments
- Optimize the performance and improve results of scans



## Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit [authors . packtpub . com](https://authors.packtpub.com) and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

## Share Your Thoughts

Now you've finished *Learn Wireshark - Second Edition*, we'd love to hear your thoughts! If you purchased the book from Amazon, please [click here](#) to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.



