Learn Wireshark

Confidently navigate the Wireshark interface and solve real-world networking problems

Packt
www.packt.com

Lisa Bock

Learn Wireshark

Confidently navigate the Wireshark interface and solve real-world networking problems

Lisa Bock

Packt

BIRMINGHAM - MUMBAI

Learn Wireshark

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.

Commissioning Editor: Vijin Boricha
Acquisition Editor: Rahul Nair
Content Development Editor: Drashti Panchal
Senior Editor: Rahul Dsouza
Technical Editor: Dinesh Pawar
Copy Editor: Safis Editing
Project Coordinator: Vaidehi Sawant
Proofreader: Safis Editing
Indexer: Priyanka Dhadke
Production Designer: Joshua Misquitta

First published: August 2019

Production reference: 1220819

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78913-450-6

`www.packtpub.com`

I dedicate this book to my husband, Michael Bock, who encouraged me, believed in me, and told me long ago that I should be a teacher. Writing my first book was one of the most significant challenges I have had to face in my life. Thank you for supporting me through the years, for your advice, sense of humor, and kindness while helping me succeed.

Contributors

About the author

Lisa Bock is an associate professor in the IT department at Pennsylvania College of Technology, in Williamsport, PA. Some of the courses she has taught since 2003 include networking, security, biometrics, protocol vulnerabilities using Wireshark, CCNA security, and unified communications. In addition to this, she is a LinkedIn learning instructor and has published over 30 courses, mainly in cybersecurity and networking. She holds an MS from UMUC along with numerous other certifications. She has had training in forensics, biometrics, networking, steganography, and network security. She is involved with various volunteer activities, has evaluated professional journals, and is an award-winning, nationally known speaker.

> *I would like to thank my friends Denise Leete, Missy Miller, and Pat Coulter who have been a part of my fabric for many years and have been supportive in so many personal ways. You each deserve all the kindness and patience the world has to give, as you so freely give it from yourselves. And also, to my colleagues Jeffrey Weaver and Edward Henninger. Thank you both for sharing your gift of time to mentor and guide me throughout my journey.*

About the reviewer

Dario Lombardo is a computer engineer. He graduated in 2001 from Politecnico di Torino, Italy. Dario specializes in computer networks and security. His main interests include network protocol security and Linux OS. He has contributed to numerous open source projects, such as tcpdump, libpcap, and the Linux kernel, just to name a few. Moreover, he has co-authored several scientific papers published in many scientific conferences. He also has a PMP certification from the PMI institute. He started working on the Wireshark project in 2013 and became a core developer in 2016. His areas of expertise are the development of external capture sources (extcaps), support for Elasticsearch, and continuous integration.

> *I'd like to thank my wife, Valentina, for always supporting me, my kids, Pietro and Sara, because they remind me everyday that we need to strive for the best. A special acknowledgment goes to Peter Wu, a fellow core developer: he's the youngest and most talented developer I've ever met. Finally, thanks to Gerald and all the Wireshark developers for the fun I have had while working on this project.*

Packt is searching for authors like you

Table of Contents

Section 2: Getting Started with Wireshark

Section 3: The Internet Suite TCP/IP

Section 4: Working with Packet Captures

Preface

This book provides a solid overview of basic protocol analysis using Wireshark. This book will show you how to navigate the Wireshark interface so that you can confidently examine common protocols such as **Transmission Control Protocol** (**TCP**), **Internet Protocol** (**IP**), and **Internet Control Message Protocol** (**ICMP**). We'll begin by outlining the benefits of traffic analysis. We'll then walk through the evolution of Wireshark, and step through the phases of packet analysis. We'll review some of the command-line tools and outline how to download and install Wireshark on either a PC or Mac.

Then, we'll gain a better understanding of what happens when you tap into a data stream. You'll learn how to personalize the Wireshark interface, and then we'll compare display and capture filters and summarize the **Open Systems Interconnection** (**OSI**) model and data encapsulation. We'll then take a closer look at some of the protocols that move data in the TCP/IP suite, and dissect the TCP handshake and teardown process. We'll conclude with advanced ways to work with packet captures, including how to troubleshoot network latency issues; how to subset, save, and export; and how to use and share captures with colleagues using CloudShark.

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 understanding of different topologies, will be beneficial.

What this book covers

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

`Chapter 2`, *Using Wireshark NG*, 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 on a PC or macOS,* illustrates how Wireshark provides support for different **operating systems** (**OSes**). We'll compare the different capture engines, 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 onto 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 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. We'll conclude with a good look at the expression builder, and discover the shortcuts used to build filters.

`Chapter 8`, *Outlining the OSI Model,* provides an overview of the 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 TCP and 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 solid understanding of the purpose of the **Internet Protocol** (**IP**). We'll outline IPv4 and the header fields and then explore IPv6 along with the streamlined header. We'll take a look at 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 and compare query and error messages. We'll look at 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`, *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 14`, *Troubleshooting Latency Issues,* outlines how even a beginner can diagnose network problems. We'll explore the coloring rules and the Intelligent Scrollbar, and then conclude with an overview of the expert system, which subdivides the alerts into categories and guides you through a more targeted evaluation.

`Chapter 15`, *Subsetting, Saving, and Exporting Captures,* helps you to discover 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 16`, *Using CloudShark for Packet Analysis,* covers CloudShark, which is 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 show you where you can find sample captures so that you can continue improving your skills.

To get the most out of this book

With Wireshark, you can capture data live from a network interface or open and examine pre-captured packets. Once you start working with a packet capture, you will want to make sense of the file and what the packets are telling you.

This book will teach you 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.

To prepare for working with Wireshark, download and install the latest version on your system. Detailed instructions are listed in `Chapter 3`, *Installing Wireshark on a PC or macOS,* on how to install on a Windows OS.

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. We will download the capture files from the many 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/diagrams used in this book. You can download it here: `https://static.packt-cdn.com/downloads/9781789134506_ColorImages.pdf`.

Conventions used

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

`CodeInText`: 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 example, words in menus or dialog boxes appear in the text like this. Here is an example: "Once, you're on CloudShark, select the **Export | Download File** drop-down menu."

Warnings or important notes appear like this.

Tips and tricks 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, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

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.packt.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

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` 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`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1
Section 1: Traffic Capture Overview

This section will enable you to understand the benefits of traffic analysis, learn about the evolution of Wireshark, step through the phases of packet analysis, review Wireshark CLI tools, outline how to download and install Wireshark on either PC or macOS, and explore the Wireshark interface.

This section is comprised of the following chapters:

- `Chapter 1`, *Appreciating Traffic Analysis*
- `Chapter 2`, *Using Wireshark NG*
- `Chapter 3`, *Installing Wireshark on PC or macOS*
- `Chapter 4`, *Exploring the Wireshark Interface*

1
Appreciating Traffic Analysis

Today's networks are complex, and if there are issues, then, on many occasions, the only way you can solve the problem is if you can see the problem. Packet sniffers such as Wireshark have been around for that very reason 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 the contents to determine whether the data should be allowed on the network.

This chapter will help you recognize the many benefits of using Wireshark for packet analysis. You'll learn about Wireshark and its history as an exceptional open source software product that includes many rich features. You'll see how everyone can benefit from using packet analysis, including network administrators, students, and security analysts. You'll be able to identify the many places to conduct packet analysis, including on a LAN, on a host, or in the real world. Finally, you will gain a better understanding of the many ways in which Wireshark can provide a key role in troubleshooting, testing, baselining, and monitoring for threats.

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

- 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 is the process of examining packets to understand the characteristics and structure of the traffic flow.

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

The analyst can complete packet analysis by either studying one packet at a time or as a complete capture. Packet analysis can be done during a live capture or by using a previously captured packet.

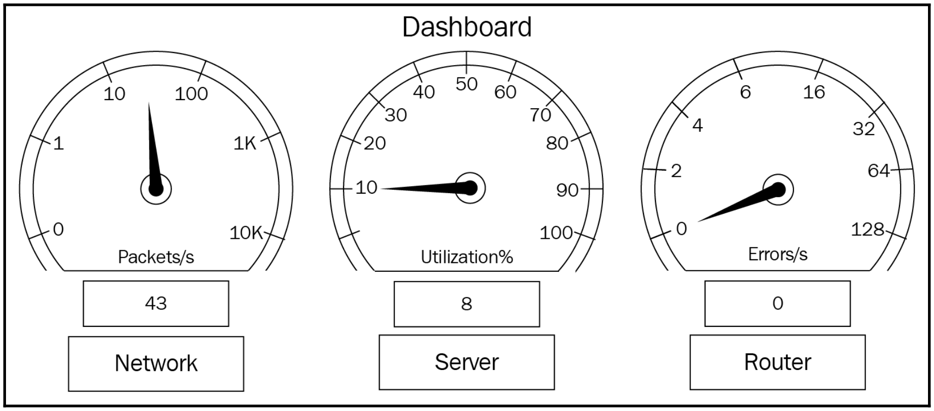
Network administrators use packet analysis to gain information about current network conditions. Security analysts use packet analysis to determine whether there is anything unusual or suspicious about the traffic when carrying out a forensic investigation. Students use packet analysis as a learning tool, to better understand the protocols. In addition, packet analysis is also used by hackers to sniff network traffic in order to gain valuable information about the network while conducting footprinting and reconnaissance.

We use packet analysis in many places, including on a LAN, on a host, or in the real world. We also use packet analysis when troubleshooting latency issues, testing **Internet of Things** (**IoT**) devices, and as a tool to baseline 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, there were various tools that enabled analysts to carry out packet analysis on the network to troubleshoot errors and to monitor server and network 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 a dashboard feature, as shown in the following diagram:



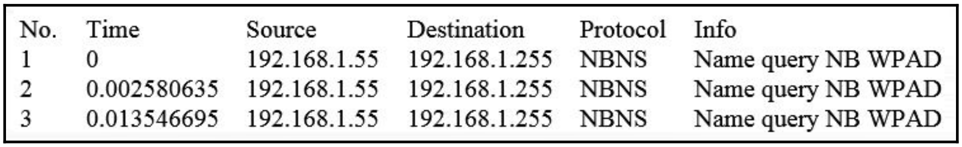
LANalyzer interface

At the same time, Microsoft introduced its network monitor. Over the last 20 years, there have been many other packet analyzers and tools to sniff traffic that include the following:

Tool	**Description**
Cain and Abel	Can gather passwords and can record VoIP conversations
NarusInsight	Formerly Carnivore, can monitor all internet traffic
dSniff	Passively monitors a network for interesting traffic
Ettercap	Eavesdrops to capture passwords, emails, and files
Tcpdump	Protocol analyzer that runs from the command line
Security Onion	Open source tool that combines packet capture with an **Intrusion Detection System** (**IDS**)
Wireshark	Packet sniffer used to analyze network traffic

Most packet analyzers have similar features. They capture the data, decode the raw bits in the headers to field values according to the appropriate **Request for Comment** (**RFC**) or other specifications, and present the data in a meaningful fashion.

The packet analysis tools range from very simple text-based analysis, such as terminal based Wireshark (tshark), as shown in the screenshot below, or tools that have a rich graphical UI with advanced AI-based expert systems that guide the analyst through a more targeted evaluation:



```
No.  Time         Source        Destination    Protocol  Info
1    0            192.168.1.55  192.168.1.255  NBNS      Name query NB WPAD
2    0.002580635  192.168.1.55  192.168.1.255  NBNS      Name query NB WPAD
3    0.013546695  192.168.1.55  192.168.1.255  NBNS      Name query NB WPAD
```

Sample output from Tshark

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

Evaluating devices that use packet analysis

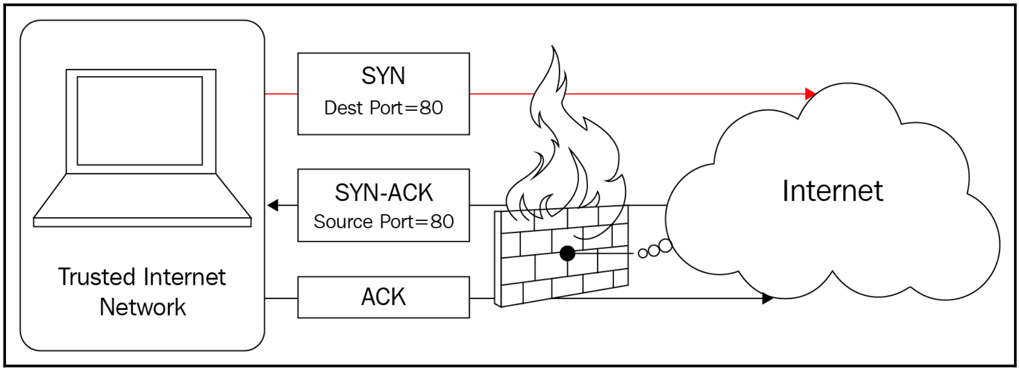
Packet analysis and packet sniffing are used by many devices on the network, including routers, switches, and firewall appliances. As data flows across the network, it passes through various network devices, which interpret the packet's raw bits and examine the field values in each packet to decide on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

A router captures the traffic and examines the IP header to determine where to send the traffic, as a part of the routing process. An IDS will capture the traffic and examine the contents and alert 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. For example, this ACL has the following entries:

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

As shown in the following diagram firewall with an ACL, in order to decide whether to allow or deny a packet, the firewall must evaluate the packet header and check to see what TCP flags are set and what port numbers are in use. If the packet does not meet the ACL entry, then the firewall will drop the packet:



Firewall with an ACL

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

As we can see, packet sniffing and analysis have been influential for many years as elements of managing networks. The first step in analysis is capturing traffic, which we will explore in the next section.

Capturing network traffic

On today's networks, a **Network Interface Card** (**NIC**) will *only* monitor traffic that is addressed to that host. We can, however, put the card into a state called **promiscuous mode**. Promiscuous mode is when the network adapter gathers not only traffic that is destined to that host, but all the traffic that is on the network, and is commonly used to monitor network activity. Therefore, to capture 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 are as follows:

1. Install Wireshark and the appropriate packet capture engine.
2. Launch Wireshark and select the appropriate capture options.
3. Start the capture and run until you capture 1,000 – 2,000 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 may 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 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

Everyone can benefit from using packet analysis, including developers, network administrators, students, and security analysts. Let's look at each and explore the benefits that can be reaped through packet analysis. We'll start with developers, who 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.

Developers typically 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 carefully check the application, however, 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 in determining the root cause of the delayed response times.

By using Wireshark, the developer can uncover common problems in transmissions, such as round-trip time and signs of congestion within an organization, which can occur in a network and impact response time.

Developers will understand that simply optimizing an application is not enough, and all development life cycles should include seeing 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.

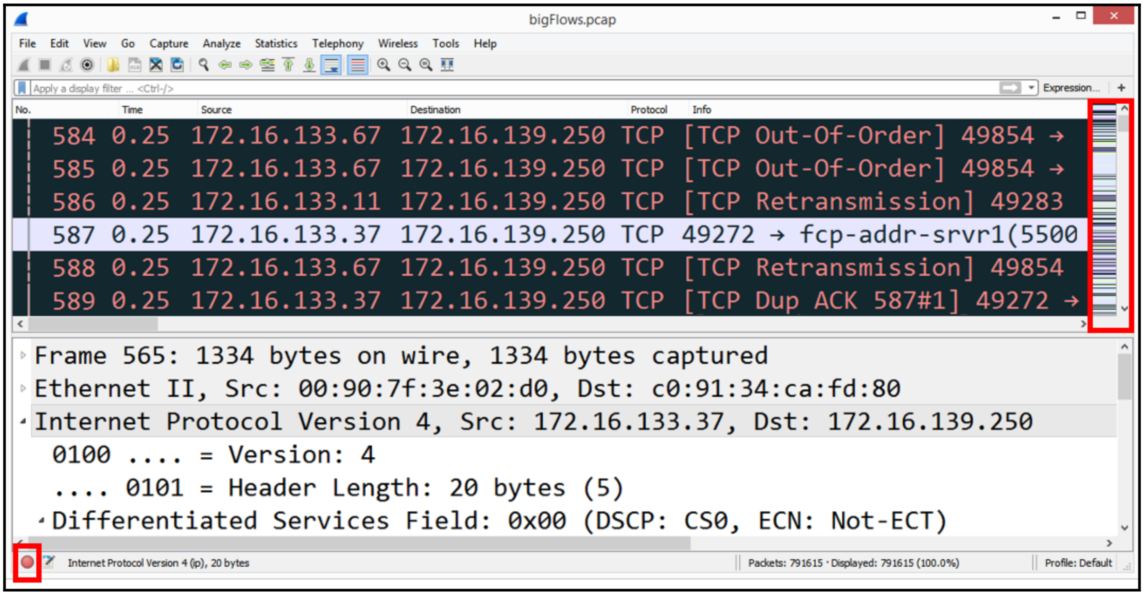
A powerful feature in Wireshark is the ability to quickly see issues in the capture. The network administrator can use both the expert system and the intelligent scrollbar, which color codes potential problems and helps with analysis, as we'll see in the next section.

Expert system and intelligent scrollbar

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 general workflow and possible critical events:

- **Chat color**: Gray provides information about typical workflows, such as a TCP window update or connection finish
- **Note color**: Cyan indicates items of interest, such as duplicate acknowledgments and TCP keep-alive segments
- **Warn color**: Yellow indicates a warning, such as a TCP zero window or connection reset
- **Error color**: Red is the highest level as there may be a serious problem, such as a retransmission or a malformed packet

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


bigFlows.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No.  Time  Source         Destination     Protocol Info
584  0.25  172.16.133.67  172.16.139.250  TCP  [TCP Out-Of-Order] 49854 →
585  0.25  172.16.133.67  172.16.139.250  TCP  [TCP Out-Of-Order] 49854 →
586  0.25  172.16.133.11  172.16.139.250  TCP  [TCP Retransmission] 49283
587  0.25  172.16.133.37  172.16.139.250  TCP  49272 → fcp-addr-srvr1(5500
588  0.25  172.16.133.67  172.16.139.250  TCP  [TCP Retransmission] 49854
589  0.25  172.16.133.37  172.16.139.250  TCP  [TCP Dup ACK 587#1] 49272 →
```

```
Frame 565: 1334 bytes on wire, 1334 bytes captured
Ethernet II, Src: 00:90:7f:3e:02:d0, Dst: c0:91:34:ca:fd:80
Internet Protocol Version 4, Src: 172.16.133.37, Dst: 172.16.139.250
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
 Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
```

Internet Protocol Version 4 (ip), 20 bytes | Packets: 791615 · Displayed: 791615 (100.0%) | Profile: Default

Expert system and intelligent scrollbar

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

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

Once problems are identified, you can then subset traffic, add comments, save, and export the packet captures.

Subsetting traffic, comment, save, and export

At times, the network administrator may want to share the packet capture with other members of the team. Wireshark can subset traffic to break apart large packet captures and focus on the problem areas.

For example, a large packet capture will most likely have several different types of traffic in addition to data, such as management traffic and 802.11 control frames. You can easily apply a filter using the **and NOT** option to exclude traffic that you don't want to see.

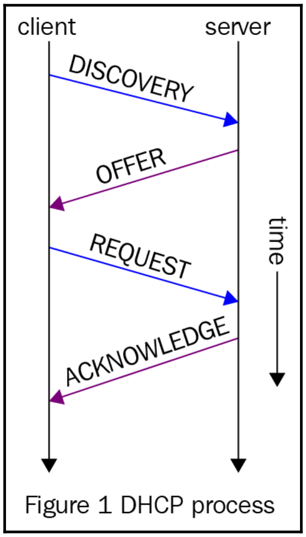
Within the subset, you can include comments. You can find comments either by selecting the comments icon in the lower left-hand corner that looks like a pad and pencil, or go to **Statistics** | **Capture file properties** and include your comments in the space below marked comments. If you do add comments, then you must save the file in the PCAPNG format as not all file formats will support the use of comments.

Once you have created a smaller file and added any (optional) comments, you can export the specified packets and save in a wide variety of formats. Formats include the default PCAPNG, along with PCAP, Sun Snoop, DMP, and many others.

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

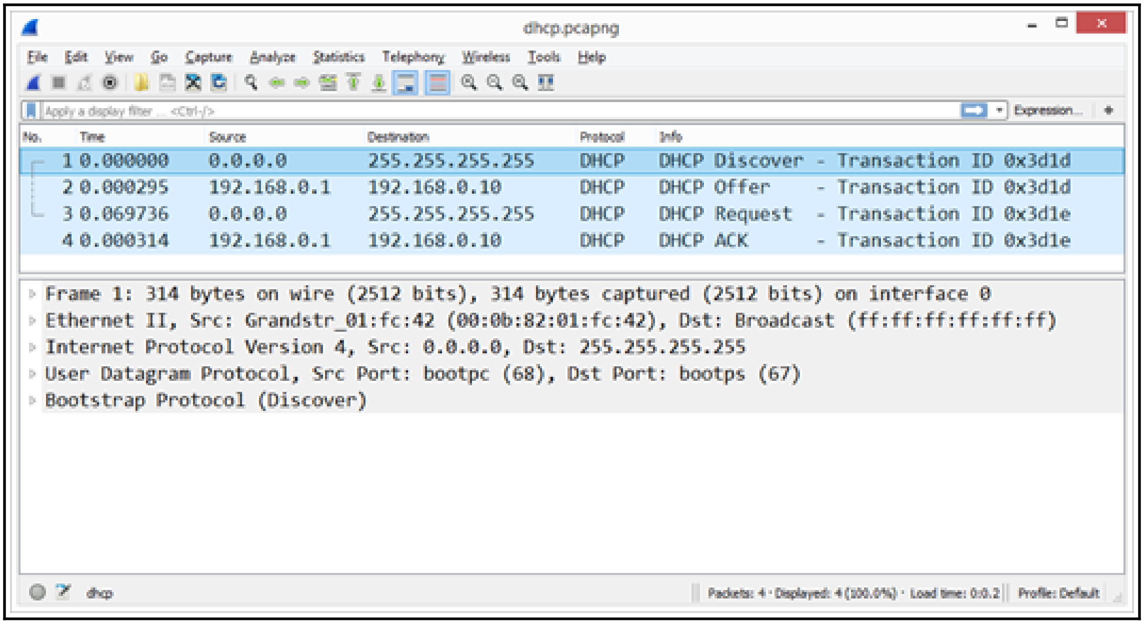
Educating students on protocols

Let's learn about how students can use packet analysis as a learning tool to better understand protocols. For example, when reviewing the DHCP process, the student might see the **DORA** process, as shown in the following diagram. While the diagram displays each of the four-part transaction, it does not show the details of each part of the four-packet exchange:



Dora process

In the following screenshot, we can see an actual DHCP transaction. The student can easily identify each of the four stages of the DORA process: **D**iscover, **O**ffer, **R**equest, and **A**cknowledge. In addition, the student can see the specifics of each exchange, including the transport protocol, the IP and MAC addresses, and the DHCP header flags:


dhcp.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
dhcp
Packets: 4 · Displayed: 4 (100.0%) · Load time: 0:0.2
Profile: Default

```
No.  Time      Source       Destination      Protocol Info
   1 0.000000  0.0.0.0      255.255.255.255  DHCP     DHCP Discover - Transaction ID 0x3d1d
   2 0.000295  192.168.0.1  192.168.0.10     DHCP     DHCP Offer    - Transaction ID 0x3d1d
   3 0.069736  0.0.0.0      255.255.255.255  DHCP     DHCP Request  - Transaction ID 0x3d1e
   4 0.000314  192.168.0.1  192.168.0.10     DHCP     DHCP ACK      - Transaction ID 0x3d1e
```

```
Frame 1: 314 bytes on wire (2512 bits), 314 bytes captured (2512 bits) on interface 0
Ethernet II, Src: Grandstr_01:fc:42 (00:0b:82:01:fc:42), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
User Datagram Protocol, Src Port: bootpc (68), Dst Port: bootps (67)
Bootstrap Protocol (Discover)
```

DORA process in Wireshark

By learning the normal behavior and purposes of common protocols, students will be able to troubleshoot problems that may 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 are security analysts.

Alerting security analysts of threats

Security analysts use packet analysis to determine whether there is anything unusual or suspicious about the traffic or discover what transpired on the network by completing a forensic investigation. To effectively discover potential problems, the security analyst must be an expert at packet analysis.

Wireshark can help the security analyst to better understand specific types of attacks so 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 found 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.

For example, if you go to `https://www.honeynet.org/node/906`, then you will see a completed challenge entitled *Forensic Challenge 12 – Hiding in Plain Sight.* Read the details on the challenge, which are outlined so you have a better understanding of the challenge. 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 valuable insight into what is happening on the network. Because of the ability to have so much insight on what is happening on the network, Wireshark is also used by hackers for reconnaissance to gather and analyze traffic—many times prior to an attack, or during an active attack, which we will discuss next.

Arming hackers with information

Hackers use packet analysis to sniff network traffic in order to gain valuable information about the network as a precursor to an attack. Sometimes called a passive attack, a hacker can use Wireshark to sniff network traffic with the goal of obtaining sensitive information. In addition, hackers can use the information gathered to launch an active attack.

As a precursor to an attack, hackers gather information during reconnaissance, which is also called footprinting. The goal of reconnaissance is to gather as much information about the target as possible. 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, a hacker will try to obtain confidential information, such as usernames and passwords exchanged, while traveling through the network. Using packet analysis to sniff network traffic can achieve the following goals:

- **Footprinting and reconnaissance**: As a precursor to an active attack, hackers use Wireshark to capture unencrypted traffic in order to gather as much information about the target as possible. In addition, Wireshark can also be used to gather additional information such as IP and MAC address, open ports and services, and possible defense methods in place.
- **Sniffing plain text passwords**: Another use of packet sniffing by hackers is looking for passwords that are sent in plain text. Common protocols that are susceptible to packet sniffers are the protocols that are in plain text, such as SNMP, HTTP, FTP, Telnet, and VoIP.

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

Hackers can launch many different types of attacks on the network, such as **Denial of Service** (**DoS**) attacks, phishing attacks, or **Structured Query Language** (**SQL**) injection attacks. Hackers can also use Wireshark to passively gather information so they can launch a more effective attack. One example is an ARP spoofing attack.

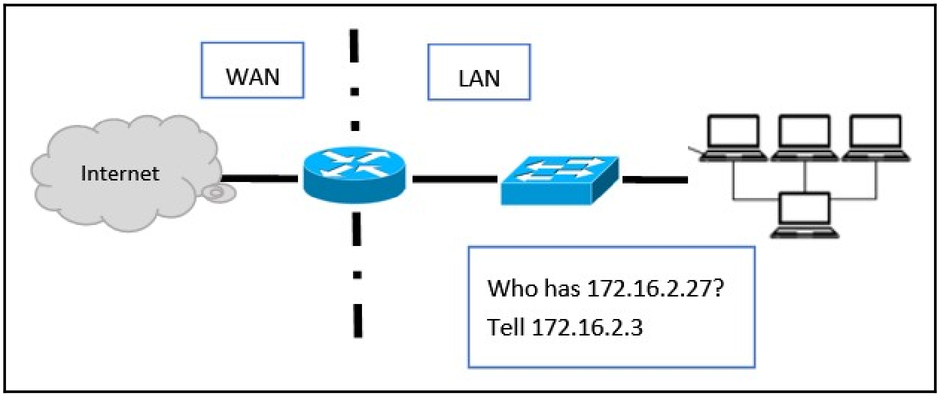
Poisoning the cache

ARP spoofing, also known as ARP cache poisoning, is used in a man-in-the-middle attack. In order to understand why this is an effective attack, let's step 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 the MAC address. The device will first 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 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:



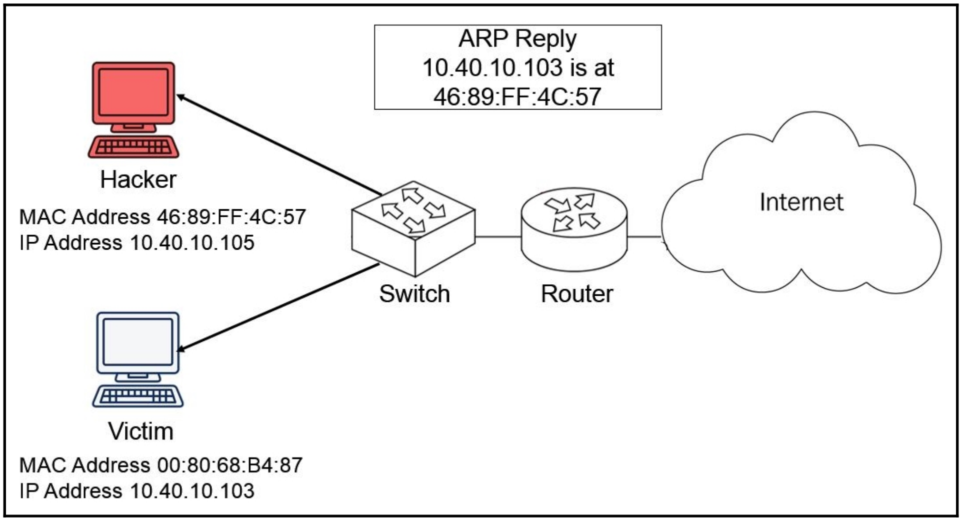
ARP broadcast on a network

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, an attacker will do the following:

- Send an unsolicited, spoofed ARP reply message that contains a spoofed MAC address for the attacker's machine to all hosts on the LAN.
- 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.
- Once the ARP tables are poisoned, this will allow an intruder to impersonate another host to gain access to sensitive information.

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



ARP spoof attack

Once the attacker begins to receive the traffic destined to another host, they will use *active* sniffing to gather the misdirected traffic in an attempt to gain sensitive information.

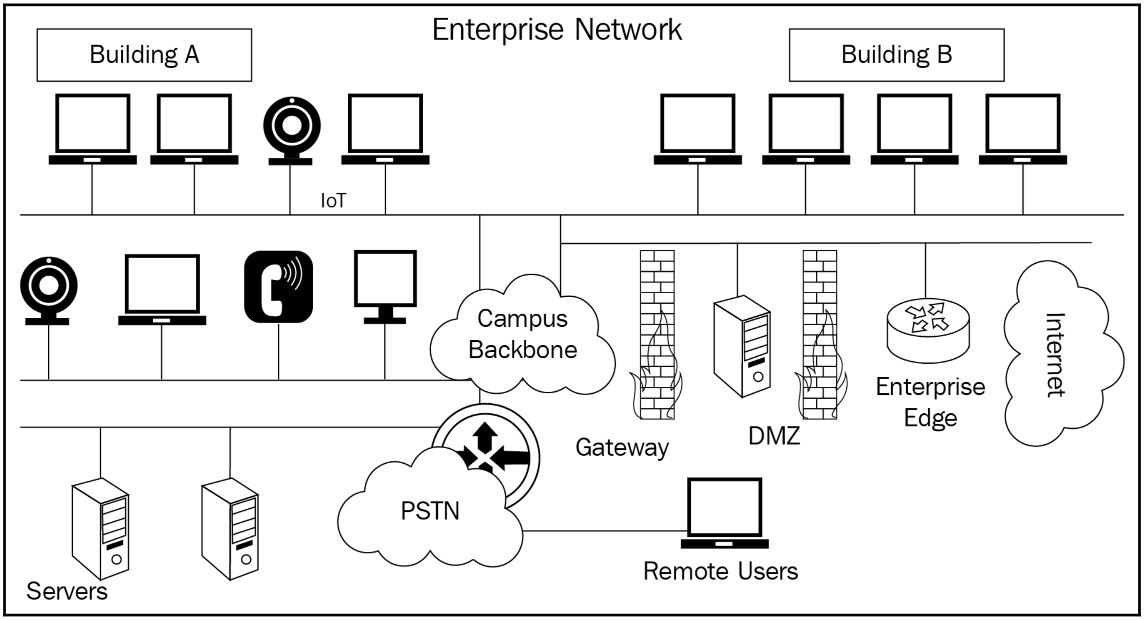
We now see the many individuals who can benefit from using packet analysis. The next section covers where packet analysis is most effective.

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 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, as the following diagram shows. An enterprise network provides connectivity, data applications, and services to the clients on the network:



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 business requirements, the network may include wide area network connectivity along with telephony.

To effectively use packet analysis, placement is key. All traffic is not created equally. Depending on placement, you may 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 may need to perform packet analysis on an individual host, such as a PC, to only monitor traffic destined to that host, or on a switch to see the traffic as it passes through the switchports.

Sniffing traffic on a host

Packet analysis can be done on an individual host. If the protocol analyzer is sniffing traffic on a switch, then the view of network traffic is limited as each switchport has its own collision domain. Therefore, on a switch on a specific port, you will only see broadcasts, multicast, and your own Unicast traffic.

To see all traffic on a switch, the network administrator can use port monitoring or **SPAN** (short for **Switched Port Analyzer**). 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 may need a special adapter. 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.

In addition to using packet analysis on a LAN or on 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. The two main functions are as follows:

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

The EINSTEIN system provides the situational awareness 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.

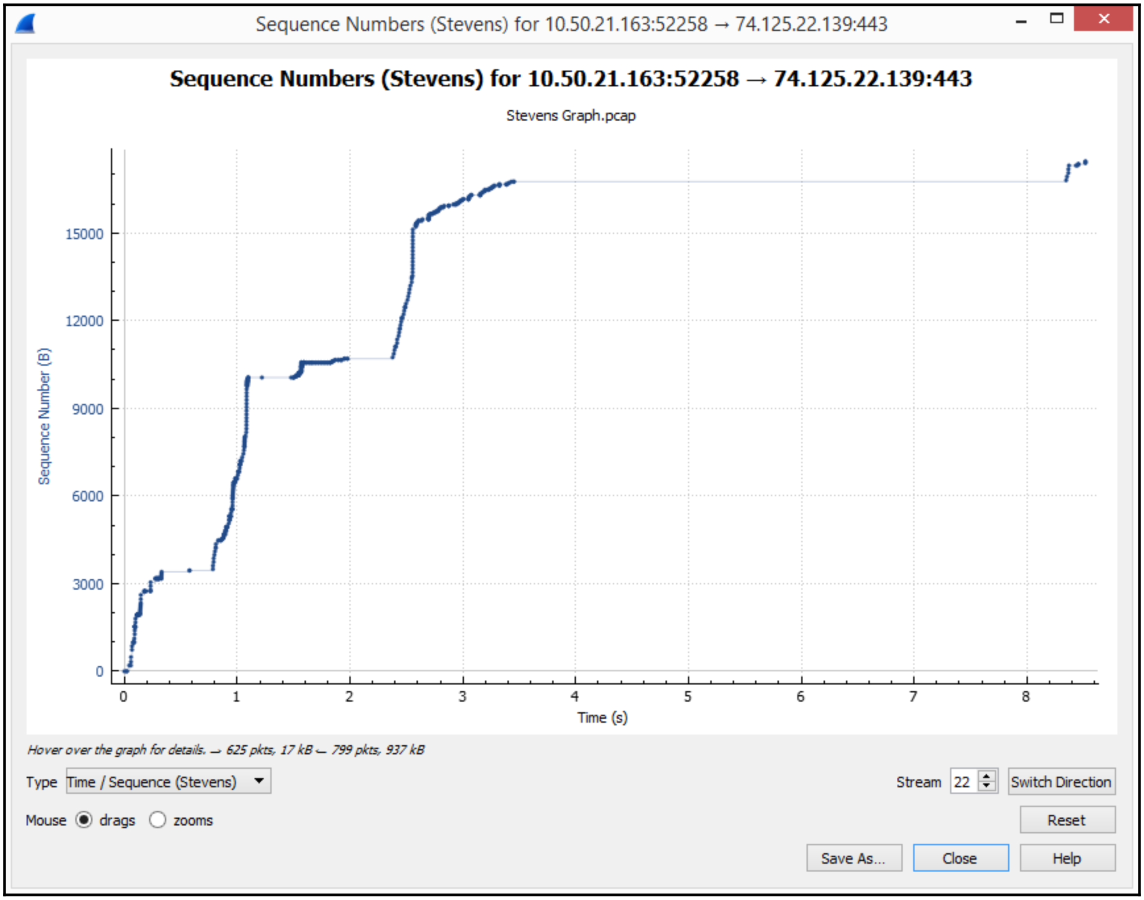
Outlining when to use packet analysis

We use packet analysis to troubleshoot latency issues, test IoT devices monitoring for threats, and as a tool to baseline the network. Let's start with troubleshooting, which is a common use of packet analysis.

Troubleshooting latency issues

Wireshark can be a valuable tool for 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 networks, unearthing problems that determine where performance can be fine-tuned. For example, if you suspect latency, then you can obtain a capture in the area where you suspect trouble and then run a Stevens graph, as shown in the following screenshot:



Sequence Numbers (Stevens) for 10.50.21.163:52258 → 74.125.22.139:443

Sequence Numbers (Stevens) for 10.50.21.163:52258 → 74.125.22.139:443

Stevens Graph.pcap

Hover over the graph for details. → 625 pkts, 17 kB ← 799 pkts, 937 kB

Type: Time / Sequence (Stevens)

Stream: 22

Mouse ● drags ○ zooms

Stevens graph

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. The IoT has become a billion-dollar industry as consumers, along with industries, are seeing the benefits of the IoT.

Even with all of the benefits, prior to connecting an IoT device to the network, it's best to test the device. Using Wireshark can help you see what happens when you plug the device into the network. The following are some of the questions 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
- **Reactive**: A system has fallen victim to an attack and the incident response team manages the attack, followed by a forensic exercise
- **Active**: Proactively seeking threats by conducting packet analysis and monitoring log files

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 may 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 may 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 see the value in using Wireshark for threat monitoring as well. For example, in Cisco's CCNA Cyber Ops 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 is 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. Characteristics to 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 on the health of the network, and possibly identify current network problems. In addition, 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 to identify possible issues in the trace and to fine-tune the configuration.

As you can see, there are many ways we can use packet analysis to monitor, test, baseline, and troubleshoot. However, you should also be aware of when you shouldn't use packet analysis.

As you can see, we can use packet analysis in many ways. However, because of the ability to obtain sensitive information or as a precursor to an attack, packet analysis should *only* be done on a network you own or where you have explicit permission to conduct packet analysis for security scans or to troubleshoot network connectivity issues. In addition, consideration should be given to maintaining the privacy of the data collected during capture and have a proper method to obtain, analyze, and retain the packet captures.

As shown in the chapter, we have now learned about 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 form by using decoders or dissectors.

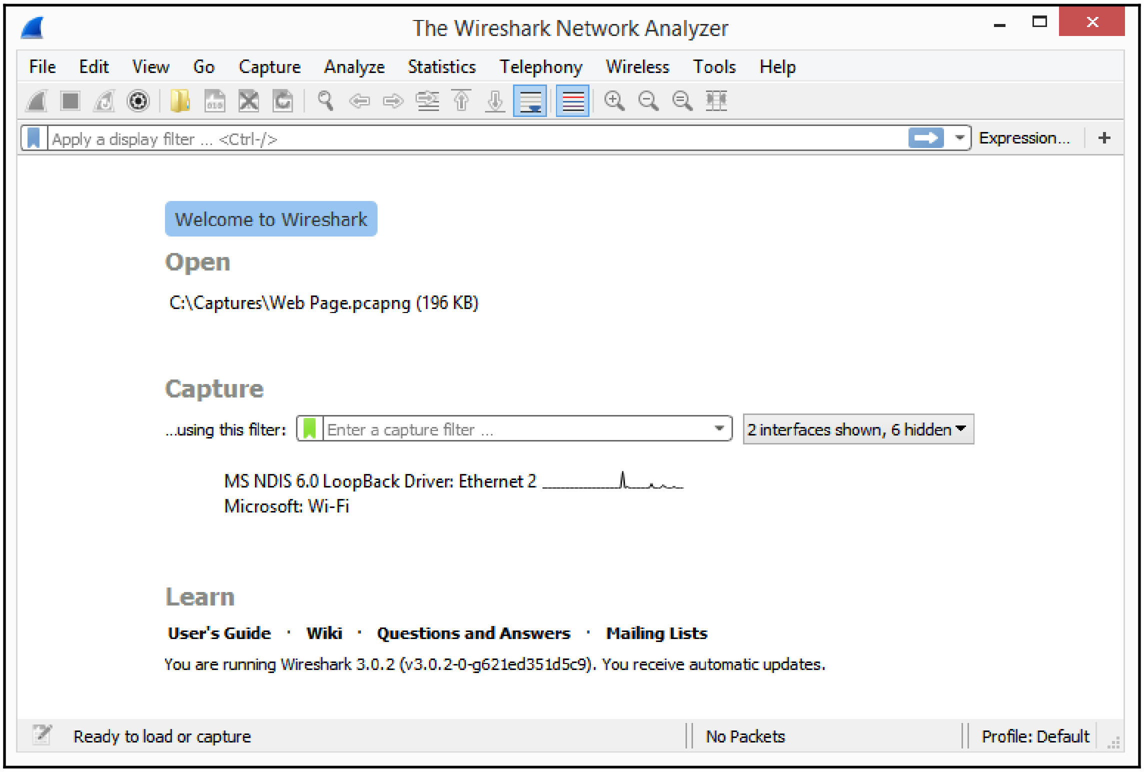
Dissectors provide information on how to break down the protocols into the proper format according to the appropriate RFC, or other specifications.

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

Wireshark is compatible with many other sniffers and has a wide range of file formats for import and export. 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. In the following graphic, we can see the simple and streamlined Wireshark welcome screen on a Windows OS:


The Wireshark Network Analyzer
File
Edit
View
Go
Capture
Analyze
Statistics
Telephony
Wireless
Tools
Help
Apply a display filter ... <Ctrl-/>
Expression...
+
Welcome to Wireshark
Open
Capture
...using this filter:
Enter a capture filter ...
2 interfaces shown, 6 hidden
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists

C:\Captures\Web Page.pcapng (196 KB)

MS NDIS 6.0 LoopBack Driver: Ethernet 2
Microsoft: Wi-Fi

You are running Wireshark 3.0.2 (v3.0.2-0-g621ed351d5c9). You receive automatic updates.

Ready to load or capture | No Packets | Profile: Default

The Wireshark interface

After using Wireshark for any length of time, you can see 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 see 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 forwarding to today, 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 about how developers, network administrators, students, and security analysts can all benefit from using packet analysis. We saw the many places where we conduct packet analysis: on a LAN, on a host, and in the real world. In addition, we have learned about how packet analysis has a variety of uses on today's networks, including troubleshooting, testing IoT devices, monitoring threats, and baselining. We have learned about how Wireshark is an exceptional open source software product that includes many rich features, has many tools available to easily solve visual problems, and provides one of the best ways to analyze network traffic.

In the next chapter, we will learn about Wireshark's predecessor, Ethereal, and how it evolved to become Wireshark. We will then compare and contrast Legacy with Wireshark Next Generation, and learn about the many improvements to the software. Because Wireshark can be resource intensive, we will learn about how Tshark can provide a lightweight alternative to Wireshark. At the end of the chapter, you will embrace the benefits of Wireshark Next Generation.

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*:

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.
    1. 1950s
    2. 1960s
    3. 1970s
    4. 1990s

2. Packet analysis is used in the real world in many forms. One is the DHS _____system, which monitors for threats.
    1. CARVER
    2. Packet
    3. EINSTEIN
    4. DESTINY3

3. In the expert system, _____ provides information about typical workflows such as TCP window updates or connection finishes.
    1. Note
    2. Chat
    3. Error
    4. Warn

4. A ____ provides a snapshot of network traffic during a window of time using Wireshark or Tshark. Characteristics can include utilization, network protocols, and effective throughput forwarding rates.
    1. Round Robin
    2. DORA process
    3. Baseline
    4. 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.
    1. Proactive
    2. Reactive
    3. Active
    4. Redactive

2
Using Wireshark NG

In this chapter, we will see how it all began by learning about Ethereal, and how over time it become Wireshark. During this journey, you'll gain insight into the many enhancements that improve the overall functionality of Wireshark. In addition, you will appreciate the work of the many authors that contribute to this project, and who help make Wireshark an exceptional tool. So that you can navigate the interface and embrace all of the improvements of Wireshark, we will take a look at the interface so that you can confidently capture and analyze packets.

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

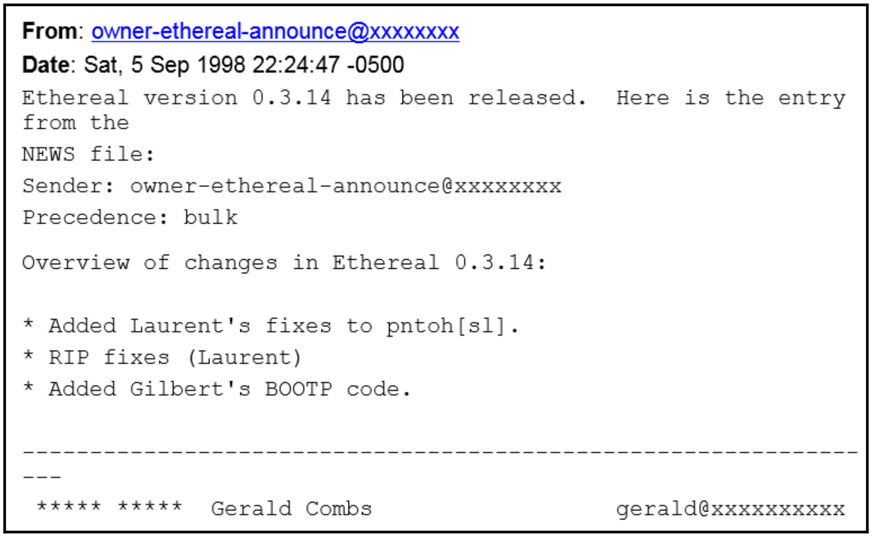
This chapter will address all of this by covering the following topics:

- Discovering the beginnings of today's Wireshark
- Examining the Wireshark interface
- Understanding the phases of packet analysis
- Learning Wireshark CLI tools

Discovering the beginnings of today's Wireshark

The term ethereal is defined as meaning delicate, airy, elegant, and exquisite; almost magical. The definition seems fitting, as Ethereal, and then later Wireshark, almost magically allows us to understand what is happening on a network.

In the late 1990s, Gerald Combs developed Ethereal as a tool to analyze network problems. In July 1998, Ethereal version 0.2.0. was released. In the following image, we can see an email sharing news of the release of Ethereal version 0.3.14 (`https://www.wireshark.org/lists/ethereal-announce/199809/msg00000.html`). Even in the beginning, developers joined the effort and collectively sought to create a solid application:



From: owner-ethereal-announce@xxxxxxxx

Date: Sat, 5 Sep 1998 22:24:47 -0500

```
Ethereal version 0.3.14 has been released.  Here is the entry
from the
NEWS file:
Sender: owner-ethereal-announce@xxxxxxxx
Precedence: bulk

Overview of changes in Ethereal 0.3.14:


* Added Laurent's fixes to pntoh[sl].
* RIP fixes (Laurent)
* Added Gilbert's BOOTP code.


---------------------------------------------------------------
---
 ***** *****  Gerald Combs                    gerald@xxxxxxxxxx
```

Email about the release of Ethereal 0.3.14

Combs invested a great deal of his time and money to keep the project alive. He developed and shared information about Ethereal during a time when hosting a website was difficult and expensive, and the internet was so new that nearly every website was under construction. However, despite the odds, the project continued to grow and expand, as we'll see in the next section.

Developing Ethereal

Early iterations of Ethereal provided basic functionality and could run on Unix, Linux, and macOS. At that time, Ethereal could not run on Windows, primarily because there was not a capture engine for Windows at the time. In the early 2000s, Loris Degioanni and Gianluca Varenni released WinPcap. One of the early Ethereal developers, Gilbert Ramirez, used WinPcap to grab traffic using Windows. Some time after that, the developers added a Windows installer.

With the addition of a Windows installer, the Ethereal community responded positively and grew significantly, as many saw the need to do packet analysis on a Windows machine. As a result, Ethereal expanded from the academic world, which was predominantly Unix and Linux, to the rest of the world, where the Windows OS was quickly becoming the predominant player.

In 2001, a significant early development in Ethereal's history was the ability to follow a stream. This was a powerful improvement over sniffers, many of which at the time could not reconstruct a stream.

In 2006, Gerald Combs began working for CACE Technologies, the developers of WinPcap, and had to leave the name Ethereal and any active development behind due to trademark issues.

Ethereal had a new name, yet the functionality of Wireshark remained the same. At that point, the Ethereal project officially became Wireshark. In 2008, the developers released Wireshark 1.0. After the release of Wireshark 2.0, developers referred to Wireshark 1.0 as Wireshark Legacy.

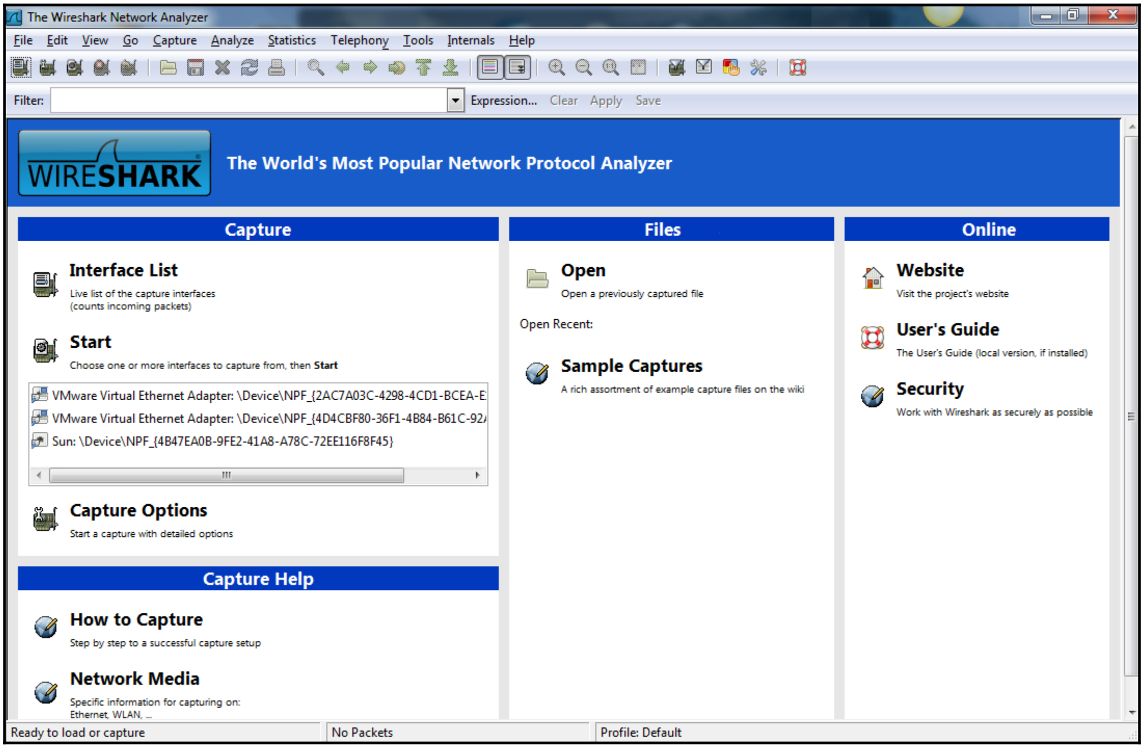
This early development—until today—grew quickly as more and more people began to see the benefits of packet analysis using Wireshark.

If you used Wireshark in the past, you know that the interface was different. The next section gives an overview of the graphical user interface of the past and the current interface of today's Wireshark.

Examining the Wireshark interface

As shown in the following screenshot, Wireshark Legacy has been around for over 10 years. In 2015, Wireshark 2.0 was released, which featured a new user interface. This evolution saw Wireshark moving from the **GIMP** (short for **GNU Image Manipulation Program**) Toolkit, or GTK, to the Qt framework.

Ethereal's graphical user interface was developed using GTK+ (`https://www.gtk.org/`) or the GIMP Toolkit. Because of the versatile nature of GTK+, and the ability to work on a wide variety of platforms, it was a logical choice for developing a user interface for Ethereal. The following screenshot represents Legacy interface:


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
WIRESHARK
Capture
Files
Online
Capture Help

The World's Most Popular Network Protocol Analyzer

Interface List
Live list of the capture interfaces (counts incoming packets)

Start
Choose one or more interfaces to capture from, then **Start**

VMware Virtual Ethernet Adapter: \Device\NPF_{2AC7A03C-4298-4CD1-BCEA-E
VMware Virtual Ethernet Adapter: \Device\NPF_{4D4CBF80-36F1-4B84-B61C-92
Sun: \Device\NPF_{4B47EA0B-9FE2-41A8-A78C-72EE116F8F45}

Capture Options
Start a capture with detailed options

How to Capture
Step by step to a successful capture setup

Network Media
Specific information for capturing on: Ethernet, WLAN, ...

Open
Open a previously captured file

Open Recent:

Sample Captures
A rich assortment of example capture files on the wiki

Website
Visit the project's website

User's Guide
The User's Guide (local version, if installed)

Security
Work with Wireshark as securely as possible

Ready to load or capture | No Packets | Profile: Default

Legacy interface

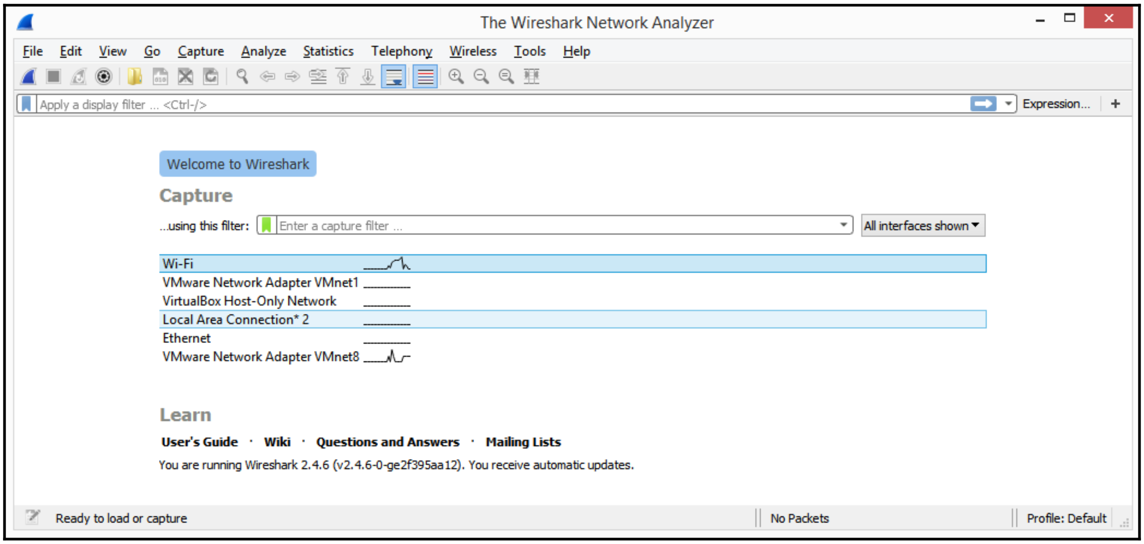
The original GTK+—or GTK—allowed developers to provide a highly usable and feature-rich graphical user interface for Wireshark. However, GTK doesn't effectively support all operating systems, which over time became problematic.

Although Ethereal, and then later, Wireshark, maintained the original appearance for many years, developers knew it was time to change.

Because of GTK's limitations in providing support for the various platforms, the developers moved to the Qt framework. Qt is a comprehensive framework, is more Windows friendly, and performs well on most operating systems.

Introducing Wireshark next generation

In 2015, developers released Wireshark 2.0. The next generation of Wireshark featured a new user interface, as well as many functional enhancements. The streamlined Qt framework is shown in the following screenshot:


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
All interfaces shown
Wi-Fi
VMware Network Adapter VMnet1
VirtualBox Host-Only Network
Local Area Connection* 2
Ethernet
VMware Network Adapter VMnet8
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists

You are running Wireshark 2.4.6 (v2.4.6-0-ge2f395aa12). You receive automatic updates.

Ready to load or capture | No Packets | Profile: Default

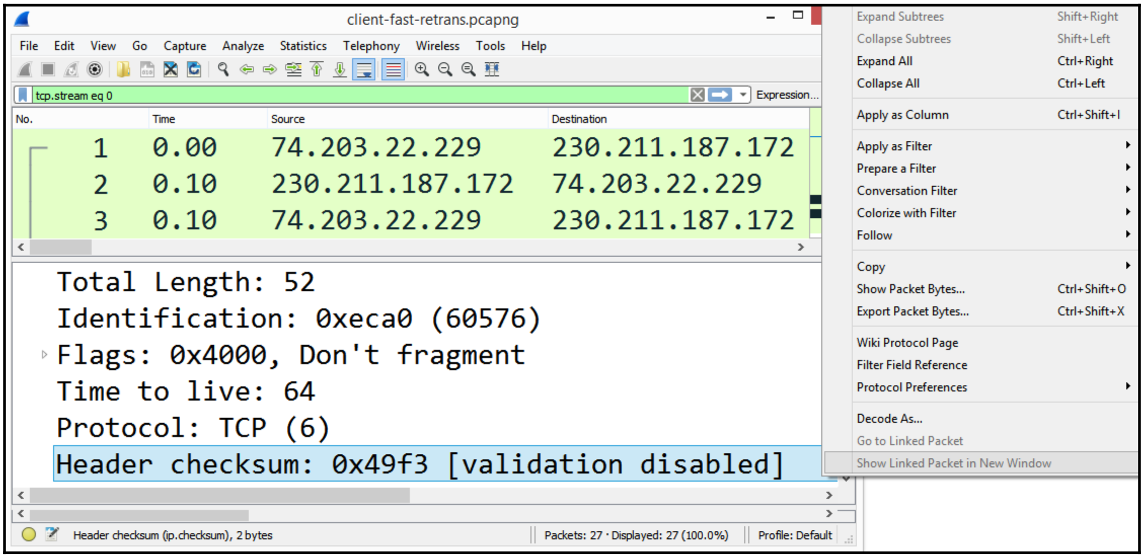
Wireshark NG

Although the interface looks as if there is not as much going on, there are many improvements, as we will see in the next section.

Enhancements

The Wireshark interface has significant improvements to get you up and running with your analysis. The interface is intuitive, with shortcuts and methods to make navigation easier. The following is a list of some of the many ways the interface improves your experience:

- Quickly begin capturing traffic by selecting an active sparkline, as shown in the previous screenshot.
- Easily add columns—simply right-click on a value in the packet details area and select **Apply as a Column**:



tcp.stream eq 0

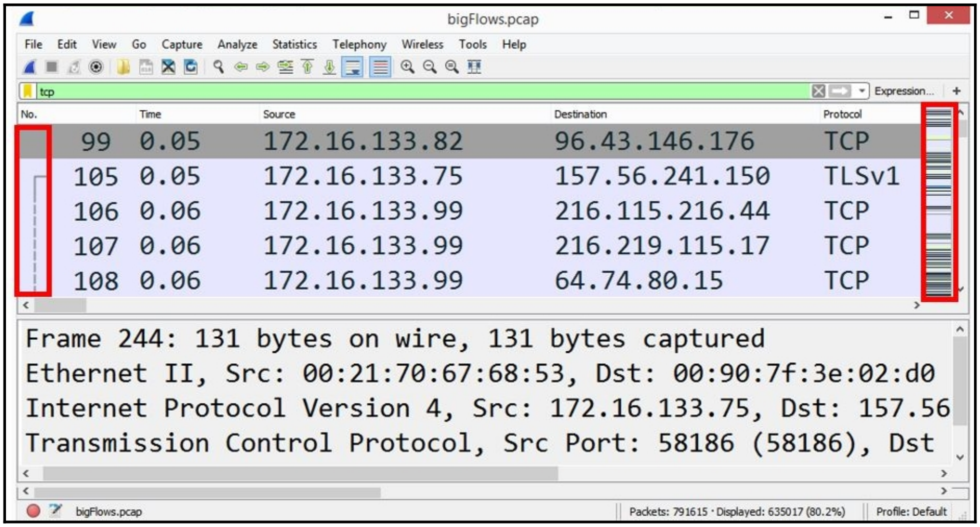
```
No.   Time   Source            Destination
1     0.00   74.203.22.229     230.211.187.172
2     0.10   230.211.187.172   74.203.22.229
3     0.10   74.203.22.229     230.211.187.172
```

```
Total Length: 52
Identification: 0xeca0 (60576)
Flags: 0x4000, Don't fragment
Time to live: 64
Protocol: TCP (6)
Header checksum: 0x49f3 [validation disabled]
```

Header checksum (ip.checksum), 2 bytes | Packets: 27 · Displayed: 27 (100.0%) | Profile: Default

Apply as a column

- Intelligent scrollbar coloring—found on the right-hand side of the packet list, as shown in the following screenshot. When coloring rules are on, you can see indication of any problems and quickly go to trouble spots, as follows:


bigFlows.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp
Expression...
Packets: 791615 · Displayed: 635017 (80.2%)
Profile: Default

No.	Time	Source	Destination	Protocol
99	0.05	172.16.133.82	96.43.146.176	TCP
105	0.05	172.16.133.75	157.56.241.150	TLSv1
106	0.06	172.16.133.99	216.115.216.44	TCP
107	0.06	172.16.133.99	216.219.115.17	TCP
108	0.06	172.16.133.99	64.74.80.15	TCP

```
Frame 244: 131 bytes on wire, 131 bytes captured
Ethernet II, Src: 00:21:70:67:68:53, Dst: 00:90:7f:3e:02:d0
Internet Protocol Version 4, Src: 172.16.133.75, Dst: 157.56
Transmission Control Protocol, Src Port: 58186 (58186), Dst
```

Wireshark Interface with Enhancements

- Enhanced graphs—flow graphs and IO graphs are easier to use.
- Coloring rules are easier to create and edit.
- Related packets—you can simply click to see related packets (shown in the preceding screenshot).
- Capable of translating to several different languages.

With approximately 1.5 million downloads per month, Wireshark has become a significant tool. It has proven to be flexible as an open source utility that encourages developers to add functionality, along with improving the overall appearance.

Each new version improves the application, adding things such as fixing a simple visual or display issue, to more significant problems that can cause an application to crash, such as 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

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

Authors

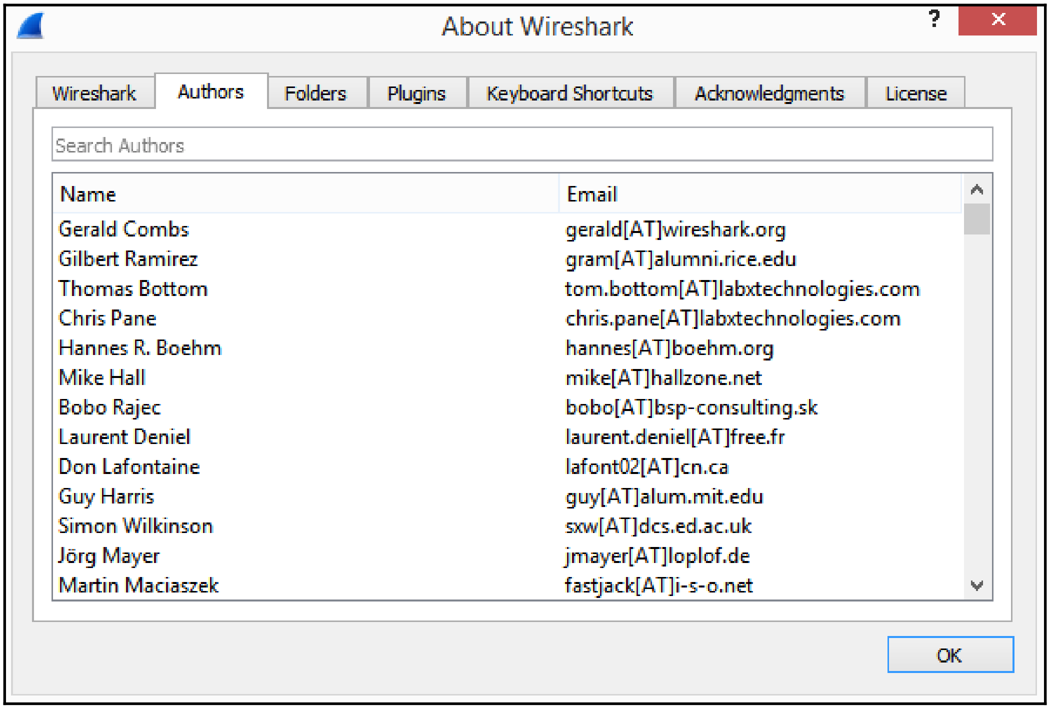
Wireshark is open source and distributed under the GNU (GNU's not Unix) **General Public License** (**GPL**). The success is attributed to the many contributing developers over the years.

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 with ongoing development and maintenance of the application. Many will jump in to add their expertise, and some contribute 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 enhancement with the Wireshark team.

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



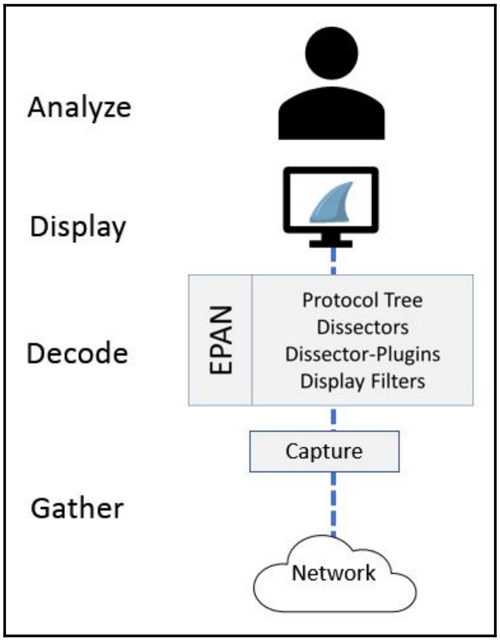
Name	Email
Gerald Combs	gerald[AT]wireshark.org
Gilbert Ramirez	gram[AT]alumni.rice.edu
Thomas Bottom	tom.bottom[AT]labxtechnologies.com
Chris Pane	chris.pane[AT]labxtechnologies.com
Hannes R. Boehm	hannes[AT]boehm.org
Mike Hall	mike[AT]hallzone.net
Bobo Rajec	bobo[AT]bsp-consulting.sk
Laurent Deniel	laurent.deniel[AT]free.fr
Don Lafontaine	lafont02[AT]cn.ca
Guy Harris	guy[AT]alum.mit.edu
Simon Wilkinson	sxw[AT]dcs.ed.ac.uk
Jörg Mayer	jmayer[AT]loplof.de
Martin Maciaszek	fastjack[AT]i-s-o.net

List of authors

Next, let's take a look at packet analysis, the process of gathering traffic on the network, decoding and dissecting the raw bits, and presenting it in a human-readable format for analysis.

Understanding the phases of packet analysis

Regardless of the software, there are four main phases of packet analysis: gather, decode, display, and analyze, as shown in the following diagram:



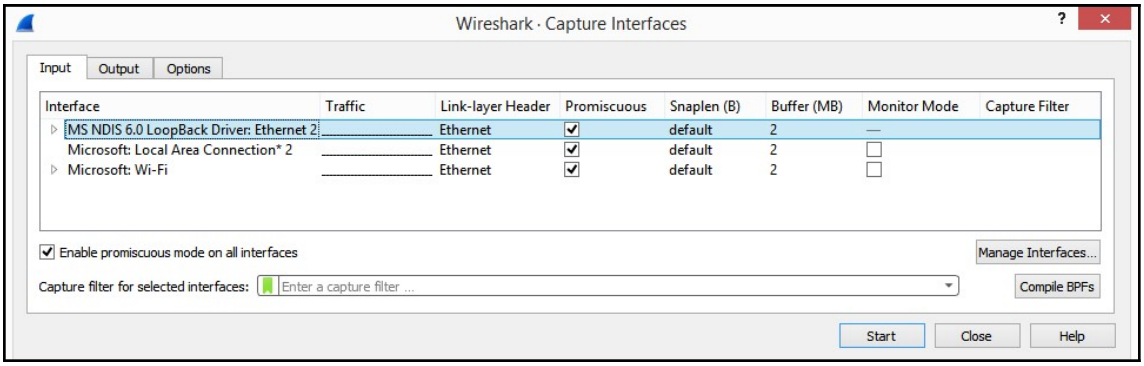
Phases of packet analysis

The first step in packet analysis is to obtain network traffic in some way. The following steps go through the gather process of packet analysis, which involves capturing the network traffic. We'll start 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, shown as *A* in the *Wireshark NG* screenshot, and begin the capture. Alternatively, you can go to the **Capture** menu, and then go to **Options**. This will open the following window. Once in, there are a few key areas that will enable you to more effectively capture traffic: capturing in promiscuous mode, and using a capture engine, which we will discuss in the next section:



Interface	Traffic	Link-layer Header	Promiscuous	Snaplen (B)	Buffer (MB)	Monitor Mode	Capture Filter
▷ MS NDIS 6.0 LoopBack Driver: Ethernet 2		Ethernet	☑	default	2	—	
Microsoft: Local Area Connection* 2		Ethernet	☑	default	2	□	
▷ Microsoft: Wi-Fi		Ethernet	☑	default	2	□	

☑ Enable promiscuous mode on all interfaces

Capture filter for selected interfaces: Enter a capture filter ...

Enabling promiscuous mode

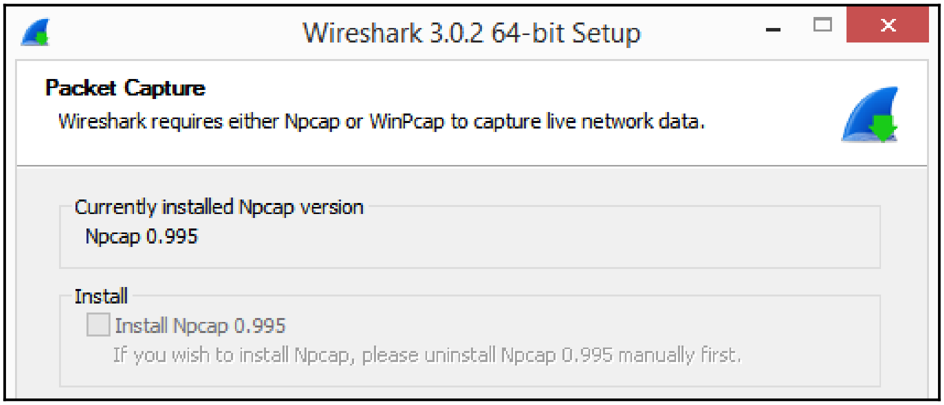
Capturing in promiscuous mode

You can capture on all interfaces, but make sure you check **Promiscuous,** as shown in the preceding screenshot, as one of the column headers. This will allow you to see all the traffic that is coming into the network interface card. You can also check **Enable promiscuous mode on all interfaces**, as shown in the lower left-hand corner of the preceding screenshot. After choosing an interface to listen on, and placing it in promiscuous mode, the interface gathers up network traffic.

Using a capture engine

Part of effectively capturing traffic is the capture engine. A packet capture or pcap engine provides an **Application Programming Interface** (**API**) to capture traffic from the network before the traffic is processed by the operating system.

As a result, when installing Wireshark, you will see a window appear, prompting you to install Npcap. A lot of times, people aren't really sure if we should install Npcap. However, as shown in the following screenshot, Wireshark requires either Npcap or WinPcap to capture data. If you don't install it, Wireshark won't run as expected. The following screenshot represents the prompt to install Npcap:



Wireshark 3.0.2 64-bit Setup

Packet Capture

Wireshark requires either Npcap or WinPcap to capture live network data.

Currently installed Npcap version

Npcap 0.995

Install

□ Install Npcap 0.995

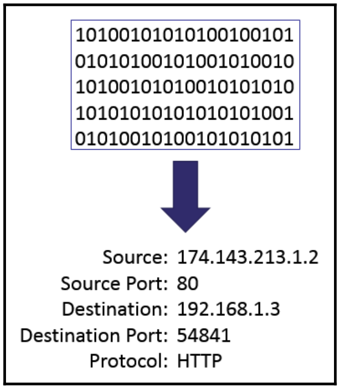
If you wish to install Npcap, please uninstall Npcap 0.995 manually first.

Prompt to install Npcap

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. While capturing data from the network is possible, it will enter the interface as random binary data. The following diagram represents the illustration of converting bits into a human-readable format:

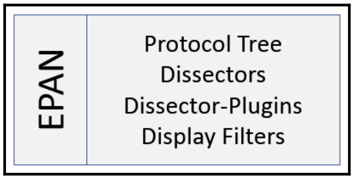


Converting bits into a human-readable form

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

Enhanced Packet Analyzer (EPAN)

Wireshark was called Ethereal before 2006, but the main core is the same. EPAN is the packet-analyzing engine for Wireshark. EPAN uses decoders or dissectors which provide information on how to recreate the protocols in the proper format:

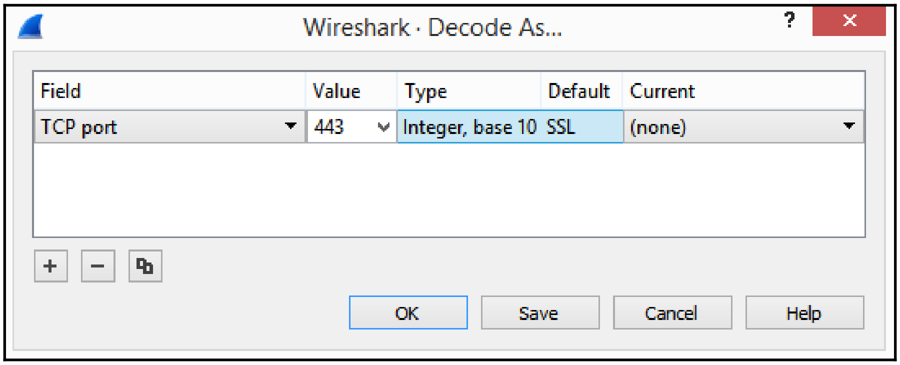


Enhanced packet analyzer

The EPAN contains four main Application Programming Interfaces (APIs), as shown in the preceding diagram:

- **Protocol tree**: Detailed analysis of a single packet
- **Dissectors**: Provide information on how to break down the protocols into the proper format according to the appropriate Request for Comment (RFC) or other specification
- **Dissector plugins**: Uses dissectors as separate functions
- **Display filters**: Allows you to filter captured data

In most cases, Wireshark is able to correctly identify and decode the protocol. However, there are times when you will need to help Wireshark decode the protocol. That is achieved by right-clicking the frame and selecting **Decode As...**, which will bring up the following window. Once in the window, you can modify the values to match the appropriate protocol:



Field	Value	Type	Default	Current
TCP port	443	Integer, base 10	SSL	(none)

Decode As...

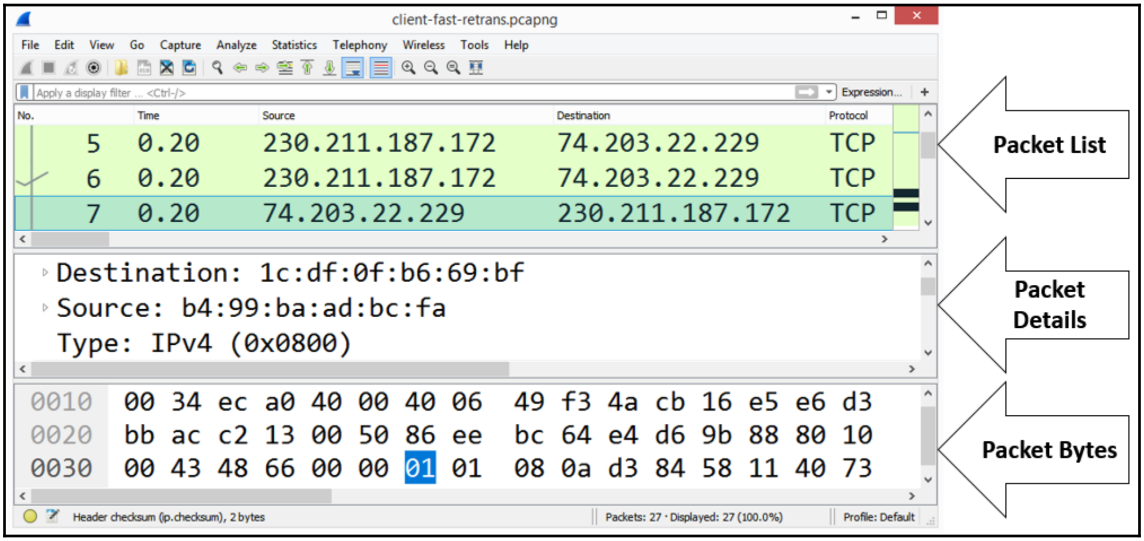
This function is very useful when protocols either don't have a dedicated port or they're running on a different port than usual. For example, you should use **Decode As...** when HTTP is running on port `8080` instead of port `80`.

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 displays three panels, as shown in the following screenshot:

- Packet list
- Packet details
- Packet bytes:


client-fast-retrans.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+
Packet List
Packet Details
Packet Bytes

No.	Time	Source	Destination	Protocol
5	0.20	230.211.187.172	74.203.22.229	TCP
6	0.20	230.211.187.172	74.203.22.229	TCP
7	0.20	74.203.22.229	230.211.187.172	TCP

Destination: 1c:df:0f:b6:69:bf
Source: b4:99:ba:ad:bc:fa
Type: IPv4 (0x0800)

```
0010  00 34 ec a0 40 00 40 06  49 f3 4a cb 16 e5 e6 d3
0020  bb ac c2 13 00 50 86 ee  bc 64 e4 d6 9b 88 80 10
0030  00 43 48 66 00 00 01 01  08 0a d3 84 58 11 40 73
```

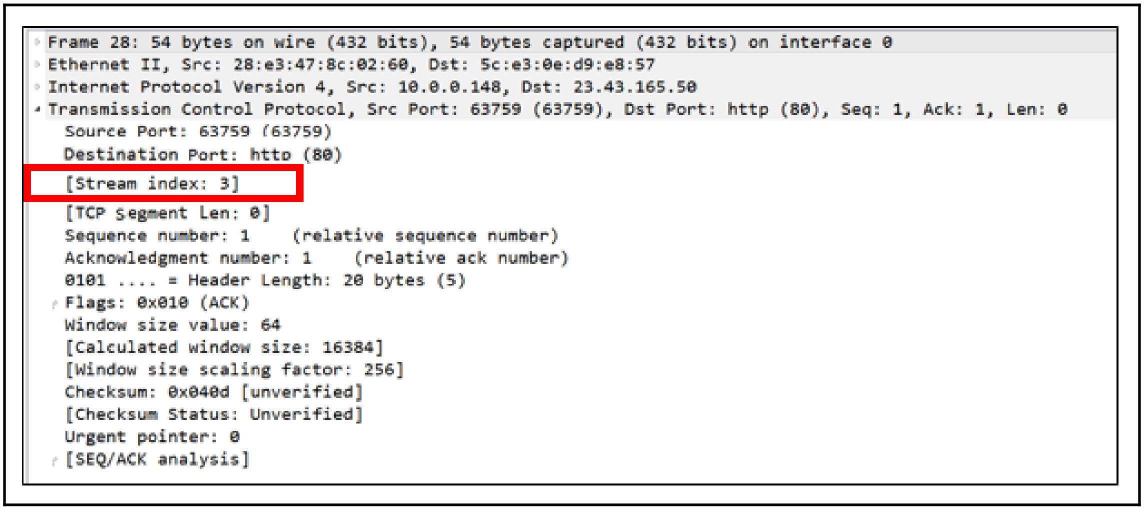
Header checksum (ip.checksum), 2 bytes | Packets: 27 · Displayed: 27 (100.0%) | Profile: Default

The Wireshark interface with three panels

The appearance of the display can be modified in the preferences by going to **Edit**, and then **Preferences**:

- **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 pane, the user can use the scroll bar on the right to navigate through the capture.

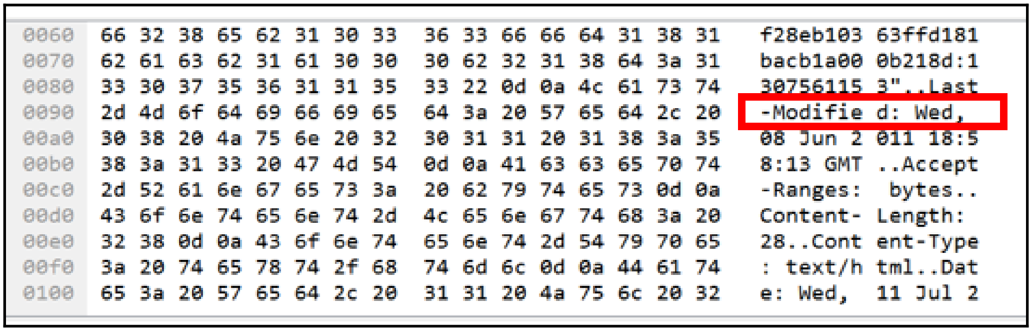
- **Packet details**: This displays the details of a single packet and includes the protocols and field values. It also displays Wireshark-specific hints. For example, there is no field value called stream index, but Wireshark lists `[Stream index: 0]` in a **Transmission Control Protocol** (**TCP**) header underneath the source and destination ports as a way to keep track of all the streams, as shown in the following screenshot:



```
Frame 28: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
Ethernet II, Src: 28:e3:47:8c:02:60, Dst: 5c:e3:0e:d9:e8:57
Internet Protocol Version 4, Src: 10.0.0.148, Dst: 23.43.165.50
Transmission Control Protocol, Src Port: 63759 (63759), Dst Port: http (80), Seq: 1, Ack: 1, Len: 0
  Source Port: 63759 (63759)
  Destination Port: http (80)
  [Stream index: 3]
  [TCP Segment Len: 0]
  Sequence number: 1    (relative sequence number)
  Acknowledgment number: 1    (relative ack number)
  0101 .... = Header Length: 20 bytes (5)
  Flags: 0x010 (ACK)
  Window size value: 64
  [Calculated window size: 16384]
  [Window size scaling factor: 256]
  Checksum: 0x040d [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  [SEQ/ACK analysis]
```

Packet details pane

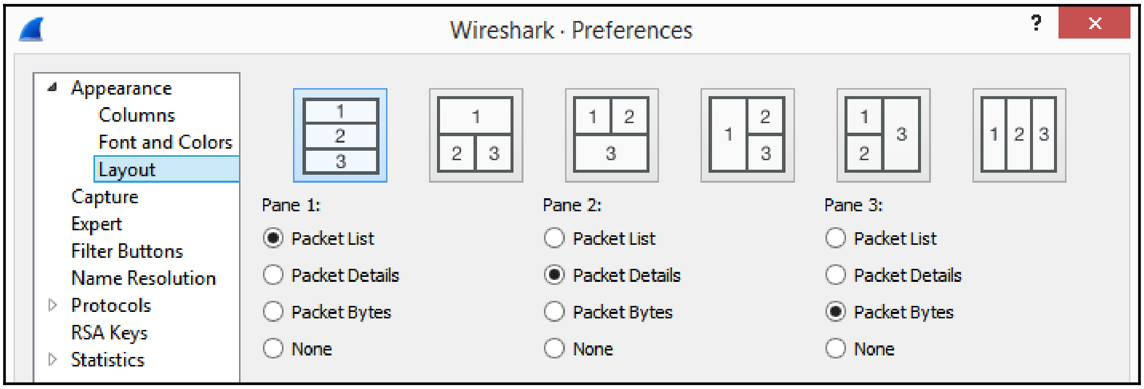
- **Packet bytes**: This is a hexadecimal representation of the single packet, as shown in the packet details pane. Any data will be displayed on the right-hand side, as shown in the following screenshot:



```
0060  66 32 38 65 62 31 30 33  36 33 66 66 64 31 38 31   f28eb103 63ffd181
0070  62 61 63 62 31 61 30 30  30 62 32 31 38 64 3a 31   bacb1a00 0b218d:1
0080  33 30 37 35 36 31 31 35  33 22 0d 0a 4c 61 73 74   30756115 3"..Last
0090  2d 4d 6f 64 69 66 69 65  64 3a 20 57 65 64 2c 20   -Modifie d: Wed,
00a0  30 38 20 4a 75 6e 20 32  30 31 31 20 31 38 3a 35   08 Jun 2 011 18:5
00b0  38 3a 31 33 20 47 4d 54  0d 0a 41 63 63 65 70 74   8:13 GMT ..Accept
00c0  2d 52 61 6e 67 65 73 3a  20 62 79 74 65 73 0d 0a   -Ranges:  bytes..
00d0  43 6f 6e 74 65 6e 74 2d  4c 65 6e 67 74 68 3a 20   Content- Length:
00e0  32 38 0d 0a 43 6f 6e 74  65 6e 74 2d 54 79 70 65   28..Cont ent-Type
00f0  3a 20 74 65 78 74 2f 68  74 6d 6c 0d 0a 44 61 74   : text/h tml..Dat
0100  65 3a 20 57 65 64 2c 20  31 31 20 4a 75 6c 20 32   e: Wed,  11 Jul 2
```

Packet bytes

The appearance of the display can be modified in the preferences by going to **Edit**, and then **Preferences**:



Pane 1:
● Packet List
○ Packet Details
○ Packet Bytes
○ None

Pane 2:
○ Packet List
● Packet Details
○ Packet Bytes
○ None

Pane 3:
○ Packet List
○ Packet Details
● Packet Bytes
○ None

Preferences—Layout

Once in **Preferences**, and then **Layout**, you can change your layout to one of many different configurations, as shown in the previous screenshot.

After displaying the result, we then move to taking a good look at the captured data and doing an analysis of what we have captured. The next section provides a summary on the final stage of packet analysis: analyze.

Analyzing the packet capture

Once you have the capture, you can start analysis in real time or use a pre-captured file. Use the built-in tools to troubleshoot and examine the traffic:

- Filter traffic to display specific types of flows, such as DNS or HTTP traffic.
- Search for specific packets, that is, `tcp.port = = 443`.
- Turn on the coloring rules or use the expert system to easily spot problems.
- Follow the stream to see the details of a single conversation.
- Do a deep packet analysis of individual frames and examine the field values of the headers.

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

In addition to the tools within Wireshark, you can subset the data to share the smaller file with coworkers and add comments to the file or in an individual frame.

Although the Wireshark 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 command-line tools, which is what the next section is all about.

Using command-line tools

Wireshark has several command-line tools that complement Wireshark's basic functionality and will allow you to do several tasks, such as edit, split, and manipulate packet captures. The following table lists some of the tools available. All the CLI tools are baked into Wireshark; however, they are also available to use as lightweight tools to work with packet captures:

Tool	**Function**
`dumpcap`	A program used to capture network traffic
`editcap`	Can edit and subset capture files
`capinfos`	Provides basic statistics on the capture file
`mergecap`	Can merge multiple capture files into one
`text2pcap`	Converts a hexdump of **ASCII** (short for **American Standard Code for Information Interchange**) packets into a capture file
`tshark`	A lightweight command-line equivalent of Wireshark

As you can see, there are many command-line tools to capture network traffic. 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 Ethereal development process included **Terminal Ethereal** (**Tethereal**), which was a CLI tool. Tethereal was later renamed **tshark** or **Terminal Wireshark**.

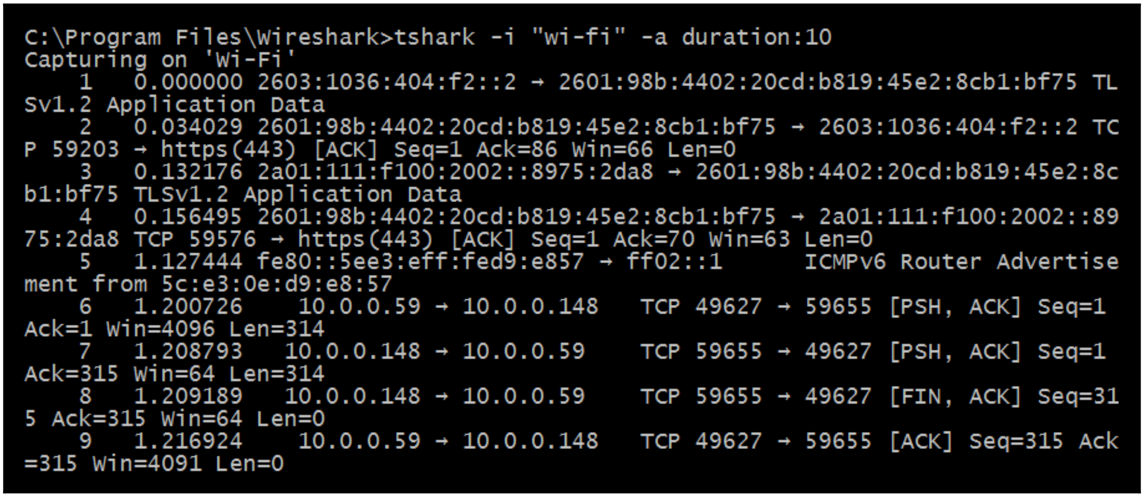
tshark is a lightweight CLI tool. To capture using tshark on a Windows machine, go into the CLI. If you have multiple interfaces, find which interface is active using `ipconfig`, then build a command, as the following code shows. Keep in mind that the commands on a Windows machine are not case-sensitive:

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

To run the tshark example, follow these steps:

1. Begin the command with tshark.
2. Identify the interface by using `-i`, then the interface name.
3. To write to a file, use `-w`, then 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
```

Output from running tshark

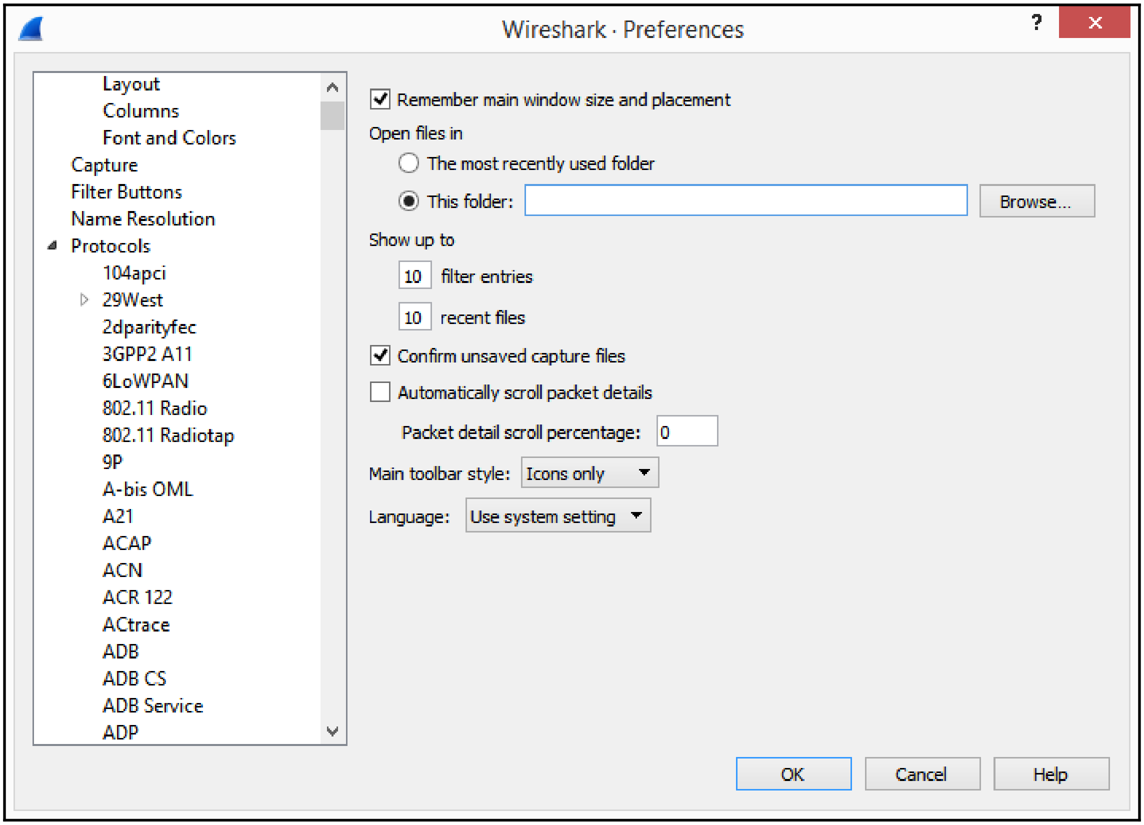
The Wireshark documentation lists a number of switches to use with tshark. The following table of command-line tools are from the documentation, which can be found at `https://www.wireshark.org/docs/wsug_html_chunked/ChCustCommandLine.html`.

Many are the same options that you can use while using Wireshark's graphical user interface, such as adding filters and specific field values. The following table represents the options in tshark:

Output	
`-w <outfile|->`	Set the output filename (or − for stdout)
`-i <interface>`	Name or idx of interface (def: first non-loopback)

Capture stop conditions	
`-c <packet count>`	Stop after *n* packets (def: infinite)
`-a <autostop cond.> ...`	• **duration**:NUM - stop after NUM seconds • **filesize**:NUM - stop this file after NUM KB • **files**:NUM - stop after NUM files

When Gerald Combs and the original development team first released Ethereal, it had limited functionality and could decode less than six protocols. 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 **General Public License** (**GPL**) and can be found in the current Wireshark source code repository. Here is a snap of Wireshark preferences—protocols:



☑ Remember main window size and placement

Open files in

○ The most recently used folder

● This folder:

Show up to

10 filter entries

10 recent files

☑ Confirm unsaved capture files

☐ Automatically scroll packet details

Packet detail scroll percentage: 0

Main toolbar style: Icons only

Language: Use system setting

Wireshark Preferences—protocols

Wireshark NG is loaded with 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 preceding screenshot—and then scroll to see the desired protocol.

Summary

In this chapter, we learned about the evolution of Wireshark and how the current interface allows you to quickly begin capturing by clicking on a sparkline, easily add columns to the interface, and use intelligent scrollbar coloring.

You can now appreciate how each new version of Wireshark improves the application. We learned about how Wireshark developers constantly update the software as many people contribute to the success of Wireshark. We then explored the phases of packet capture, as it progresses from gathering the traffic from the network to processing it into a human-readable format that allows you to conduct an analysis. Finally, we saw how Wireshark can be resource intensive; therefore, it's important to understand why sometimes, it's better to use CLI tools such as tshark, a lightweight application for capturing packets.

In the next chapter, we will explore downloading and installing Wireshark on various OSes, such as Windows, macOS, and Linux. We will take the time to explore the different capture engines. Once you do decide to download Wireshark, we will evaluate the different available download options. During installation on a PC or Mac, you'll see the various 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 *Assessment*:

1. When Gerald Combs began working for CACE Technologies, he had to leave the name Ethereal, and any active development behind due to trademark issues. In May _____, Ethereal officially became Wireshark.
    1. 1998
    2. 2001
    3. 2006
    4. 2015

2. In July _____, Ethereal version 0.2.0. was initially released.
   1. 1998
   2. 2001
   3. 2006
   4. 2015
3. A _____ engine provides an API to capture traffic from the network, before the traffic is processed by the operating system.
   1. CACE
   2. Pcap
   3. Tcap
   4. Capinfos
4. _____ provides information on how to break down the protocols into the proper format, according to the appropriate RFC or other specification.
   1. Protocol Tree
   2. Dissector Filters
   3. Capinfos
   4. Dissectors
5. Wireshark has several CLI tools that complement the basic functionality, ____ can merges multiple capture files into one.
   1. Tshark
   2. Capinfos
   3. Mergecap
   4. Text2pcap

3
Installing Wireshark on a PC or macOS

To start capturing and analyzing packets, you'll first have to download and install Wireshark on your computer or laptop. In this chapter, we will discover how easy it is to install Wireshark on a variety of different OS. We will learn about the importance of a capture engine, and why it is necessary to capture network traffic.

You will see that when installing Wireshark on a PC, several options are presented. We will review the different options so that you can confidently navigate the installation and make the correct selections so that you can begin capturing traffic. Finally, because Wireshark is open source, with constant enhancements and improvements, you will learn about the many online resources to see the latest news and updates, download options, and access help to improve your workflow.

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

- Discovering support for different OS
- Comparing the different capture engines
- Performing a standard Windows installation
- Reviewing the resources available at Wireshark.org

Discovering support for different OS

Wireshark is an open source packet analysis tool developed as a cross-platform application. Wireshark now uses the Qt graphical UI library, which is capable of running on a variety of hardware and software platforms with little or no modification to the underlying code. The three main OS Wireshark supports are Microsoft Windows, Linux, and macOS.

Of all the OS systems today, Windows has the highest market share. The following section outlines support for the Windows OS family.

Using Wireshark on Windows

Wireshark will run on most supported Windows systems as it natively interacts with the Windows API. Currently, Wireshark can be compiled and run on the following: Windows 7, Windows 8, Windows 8.1, and Windows 10. In addition, Wireshark will run on several of the Windows Server OS, including Windows Server 2008 R2, 2012, 2012 R2, 2016, and 2019.

Running Wireshark or Wireshark Legacy may be possible on older OS, such as Windows Vista, XP, or Server 2008. However, Wireshark Legacy is no longer supported on those OS and may not perform as expected. Currently, there are still users who require a solution for Windows XP, as XP still has a small percentage of the market share worldwide. Users can obtain a copy of Wireshark for Windows XP at `https://www.wireshark.org/download/win32/all-versions/Wireshark-win32-1.10.14.exe`.

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

Running Wireshark on Unix

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

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

Installing Wireshark on macOS

On Wireshark.org, you will find an install for macOS (10.12 and later). The install for macOS is a significant improvement since Wireshark moved from GTK to Qt in that now, there is a native interface for macOS.

Prior to that, when you downloaded Wireshark on your macOS, you had to use X11, which is the X Windows system. Version 1.12 was the final release that required the X Windows system. Now, you can download and install Wireshark on a macOS just like any other OS. The installer will guide you through the installation in much the same manner as installing on a Windows machine.

As you can see, this has made Wireshark more user-friendly to the growing population of macOS users. Because of the widespread use of Linux, the following 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. Installing Wireshark on Linux may be possible, but you may run into errors during the build and installation phases.

Common problems arise when you don't have the necessary development package on your system, or when the development package is outdated. Other issues may be that you are missing libpcap.

Running Wireshark as a root user also causes problems as 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.

If you are able to install Wireshark, then you may have an issue with capturing packets and you may see a permission error:

```
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, then additional permission modifications and advanced configuration are required to capture traffic. The Wireshark community is very helpful in trying to assist users with issues, but there are options that are more reasonable, especially for novice users.

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 VM in order to get a feel of how to use Wireshark on a Linux OS for training or testing purposes.

Downloading premade virtual images

Premade virtual images are available at `https://www.osboxes.org/` where it is easy to download and run a Linux OS that has Wireshark pre-installed and ready to run. Once on osboxes.org, you'll find you can choose from one of many OS.

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.

As new OS come into 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 OS.

Working with Wireshark on other systems

Wireshark can be used on network devices and servers to monitor and analyze traffic in order to understand the traffic flow. Several Cisco devices are Wireshark capable. The devices provide the network specialist with comprehensive documentation on best practices while capturing network traffic.

Some guidelines for capturing traffic while in 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.

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 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 or PCap engine provides an API to capture traffic. Wireshark uses one of several capture engines, such as libpcap, WinPCap, AirPCap, and NPCap. Let's begin with libpcap.

Understanding libpcap

Libpcap is a capture engine that was originally developed for Unix-like OS and 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/`. A version of libpcap was adapted for Windows and is called WinPcap, as we will discuss next.

Examining WinPcap

WinPcap is a capture engine that has drivers specific to a Windows OS and can be found at `https://www.winpcap.org/`. WinPcap has been around for many years, and works in a Windows environment, specifically the Windows NT family.

WinPcap enables packet capture right from a network adapter and presents it to Wireshark before any processing is done by the OS. WinPcap (or a similar capture engine) must be installed on a Windows OS in order to capture packets. During the installation process, Wireshark will look for a copy of WinPcap and prompt the user to install WinPcap if it is not present.

WinPcap uses the **Network Driver Interface Specification** (**NDIS**) version 5.x API and has not had any recent updates, as evidenced by the changelog found at `https://www.winpcap.org/misc/changelog.htm`. As a result, WinPcap may not perform well on certain versions of Windows 10. To overcome performance issues, users are directed to use NPCap, as it might perform better.

For many years anyone who needed to analyze 802.11 management or control packets used AirPCap wireless capture devices, as we'll see in this next section.

Reviewing AirPCap

For analyzing wireless traffic, Wireshark users could capture traffic using AirPCap. The adapter was a USB Windows-friendly device that provided 802.11 capture support and worked well with Wireshark.

AirPCap was compatible with several versions of Windows, from Windows 2000 through Windows 7, and many reported having successes when using AirPCap after installing the AirPCap Nx driver, which adds support for Windows 8 and Windows 10. As of writing this, the parent company, Riverbed, no longer lists AirPCap adapters or sells the devices.

Over time, OS have changed, along with wireless in general, and 802.11ac is becoming more commonplace. AirPCap has support for 802.11a/b/g/n, but does not list 802.11ac.

If you do need to capture raw 802.11 traffic, then you could use Linux, macOS, or try NPcap, a driver that will provide support for raw 802.11 packet capture.

Let's take a look at the newest packet capture engine, Npcap.

Grasping Npcap

When installing Wireshark, users will now see an option to install NPpcp. Npcap comes from the Nmap project and is the packet sniffing library for Windows. Npcap is based on WinPcap/LibPcap but has improved features for enhanced ability to capture.

Understanding Npcap features

Npcap provides support for 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 7 and later machines.

A standard Wi-Fi card on a Windows machine can *only* be put into promiscuous mode, not monitor mode. As a result, you won't see raw 802.11 traffic or the radiotap headers, as they are wrapped so they look like an Ethernet packet, and are sometimes called **fake Ethernet packets**. With Npcap, users can capture raw 802.11 packets when using an unsupported wireless adapter.

This is easily achieved by selecting the following option during installation of Npcap:

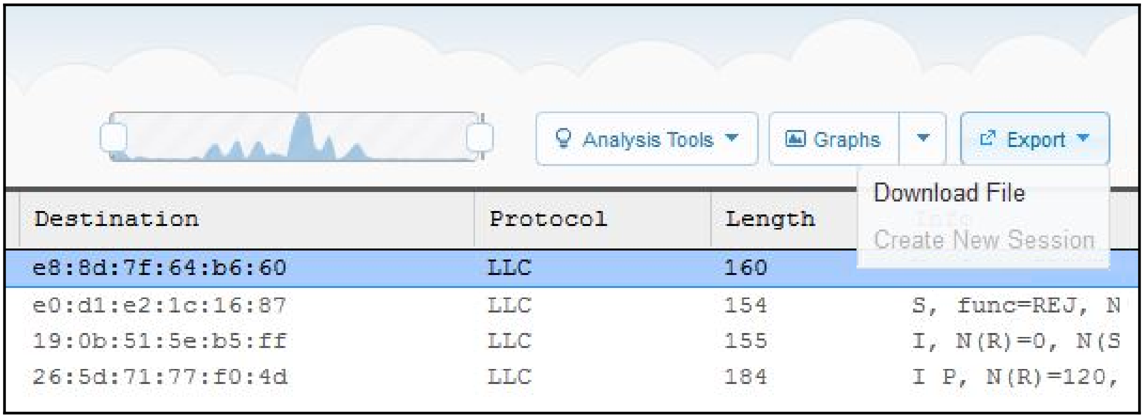
```
Support raw 802.11 traffic (and monitor mode) for wireless adapters
```

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 and control, along with the management packets that have radiotap headers

Radiotap headers can be used when troubleshooting Wi-Fi, as they can provide a lot of information such as antennae noise and channel frequency. To see an example of a radiotap header, go to `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. This is found on the right-hand side of the screen, as shown here:

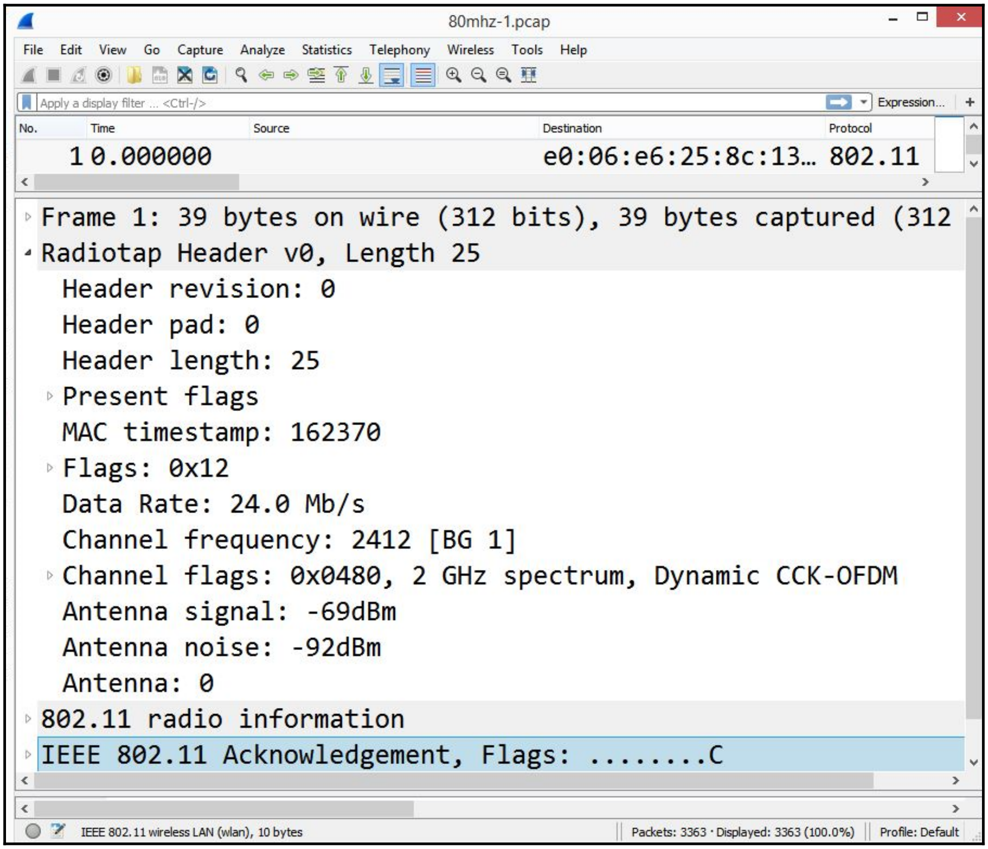


Destination	Protocol	Length	Info
e8:8d:7f:64:b6:60	LLC	160	
e0:d1:e2:1c:16:87	LLC	154	S, func=REJ, N
19:0b:51:5e:b5:ff	LLC	155	I, N(R)=0, N(S
26:5d:71:77:f0:4d	LLC	184	I P, N(R)=120,

Download file from Cloudshark

When the **Download** window opens, select **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:


80mhz-1.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No.   Time        Source   Destination          Protocol
   1 0.000000              e0:06:e6:25:8c:13…  802.11
```

```
▹ Frame 1: 39 bytes on wire (312 bits), 39 bytes captured (312
◂ Radiotap Header v0, Length 25
    Header revision: 0
    Header pad: 0
    Header length: 25
  ▹ Present flags
    MAC timestamp: 162370
  ▹ Flags: 0x12
    Data Rate: 24.0 Mb/s
    Channel frequency: 2412 [BG 1]
  ▹ Channel flags: 0x0480, 2 GHz spectrum, Dynamic CCK-OFDM
    Antenna signal: -69dBm
    Antenna noise: -92dBm
    Antenna: 0
▹ 802.11 radio information
▹ IEEE 802.11 Acknowledgement, Flags: ........C
```

```
IEEE 802.11 wireless LAN (wlan), 10 bytes    Packets: 3363 · Displayed: 3363 (100.0%)    Profile: Default
```

Radiotap header

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.

Npcap is compatible with WinPcap and can run alongside WinPcap, or you can uninstall WinPcap and use the Npcap driver exclusively. However, Wireshark documentation suggests using Npcap if you are using Windows 10. Users can compare the features of WinPcap or Npcap by going to `https://nmap.org/npcap/vs-winpcap.html`.

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, 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

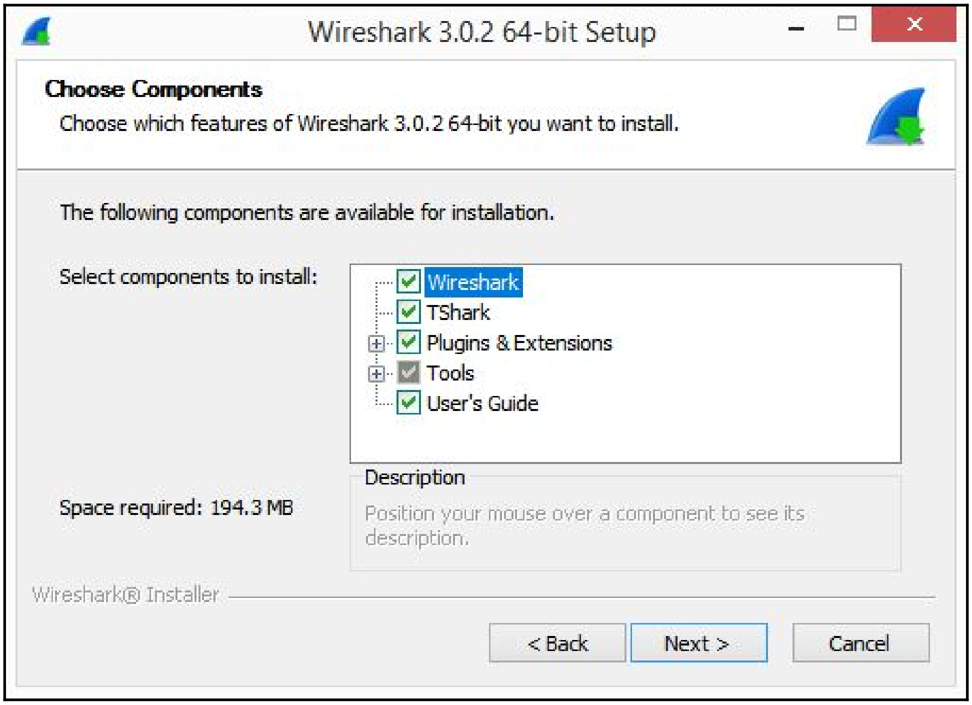
As you begin the installation, Wireshark displays a series of prompts. The following are generally the first two screens you will see:

- **Welcome Screen**: The Wireshark installation begins with a **Welcome Screen** with a warning to make sure Wireshark is not running before launching the wizard. The wizard will then guide you through the installation.
- **License Agreement**: The next screen is the **License Agreement**, which must be read and agreed upon before moving on to the next step. It might be worthwhile to read this as it provides a detailed overview of the license agreement, specifically that Wireshark is distributed under the GNU (not Unix) **General Public License**.

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. Keep in mind that these options periodically change, as shown here:



Wireshark 3.0.2 64-bit Setup

Choose Components

Choose which features of Wireshark 3.0.2 64-bit you want to install.

The following components are available for installation.

Select components to install:

- ☑ Wireshark
- ☑ TShark
- ⊞ ☑ Plugins & Extensions
- ⊞ ☑ Tools
- ☑ User's Guide

Space required: 194.3 MB

Description

Position your mouse over a component to see its description.

Wireshark® Installer

The Choose Components screen

The following are the options that appear when selecting a component to install:

- **Wireshark**: Select this choice if you want to install Wireshark. While this may be obvious, the user may only want to install TShark.
- **TShark**: TShark is a lightweight CLI tool that is not as resource intensive as the full Wireshark GUI.

- **Plugins & Extensions**: These are extra features and protocol dissectors for Wireshark and TShark:
    - **Dissector Plugins**: Plugins with some extended dissections.
    - **Tree Statistics Plugins**: Extended statistics.
    - **Meta Analysis and Tracing Engine (MATE)**: This is experimental; MATE offers configurable extensions for display filters.
    - **Transum**: A newer tool developed by `https://community.tribelab.com/` that computes response time with a number of different protocols.
    - **Codec Plugins**: Provides additional support for codecs.
    - **Simple Network Monitor Protocol (SNMP) MIBs**: Provides a more extensive dissection of the SNMP.
- **Tools**: Provides a list of command tools to select, and includes the following:
    - **editcap**: This allows you to adjust timestamps, delete packets, and convert file formats.
    - **text2pcap**: This provides the ability to take an ASCII hexdump and convert the file to a libpcap-format capture file.
    - **mergecap**: This is used when you need to combine two capture files as it merges two or more capture 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**: When you have to debug a display filter (dfilter), dftest will show the display filter byte code.
    - **capinfos**: This provides information such as the number of packets, duration, and other information about a capture file.
    - **raw shark**: This outputs and analyzes raw PCap data when required for external (third-party) integration or exports.
    - **mmdbresolve**: This program will identify and print a packet's geolocation by using an IPv4 and IPv6 address. You will need to obtain the latest GeoLite2 at `databaseshttps://dev.maxmind.com/geoip/geoip2/geolite2/`.

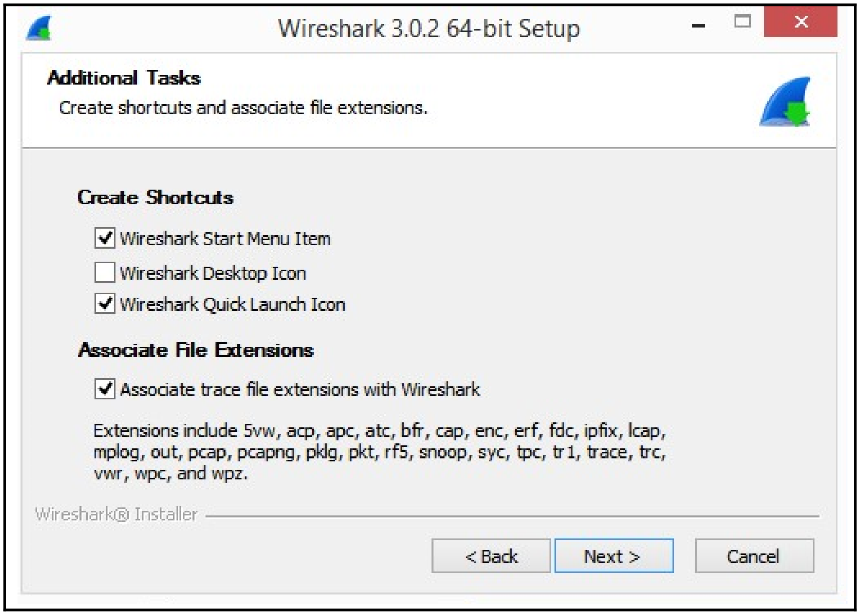
    - **androiddump**: When it's necessary to capture from an Android device, androiddump provides an interface. You'll need to have the Android **Software Development Kit** (**SDK**) along with permission to access the device.
    - **sshdump**: This provides an interface to capture from a remote host when in a **Secure Shell** (**SSH**) connection.
    - **UDPdump**: This offers a capture interface that pulls UDP packets from network devices when debugging applications that run over UDP.
    - **randpktdump**: This is a tool that enables access to the **Random Packet Generator** (**randpkt**) during testing or for educational purposes.
    - **extcap**: This is an external interface that can be used for testing, fetching and displaying data from a non-traditional remote source, or conducting Bluetooth sniffing.
- **User's Guide**: This is a copy of the user guide that you can access offline.

As you can see, there are many components included that you can to select within the Wireshark installation. 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, along with outlining the available file extensions. In addition, you'll need to decide where to store the installation folder, as outlined here:

- **Additional Tasks:** This prompt will provide choices on whether to create shortcuts for in the **Wireshark Start Menu**, **Wireshark Desktop**, or **Wireshark Quick Launch** icon, as shown here:



Additional Tasks

Create shortcuts and associate file extensions.

Create Shortcuts

☑ Wireshark Start Menu Item
□ Wireshark Desktop Icon
☑ Wireshark Quick Launch Icon

Associate File Extensions

☑ Associate trace file extensions with Wireshark

Extensions include 5vw, acp, apc, atc, bfr, cap, enc, erf, fdc, ipfix, lcap, mplog, out, pcap, pcapng, pklg, pkt, rf5, snoop, syc, tpc, tr1, trace, trc, vwr, wpc, and wpz.

Additional Tasks screen

- **Choose install location**: The user selects the default location or browses to a user-defined folder. Wireshark will provide information on how much space is required.

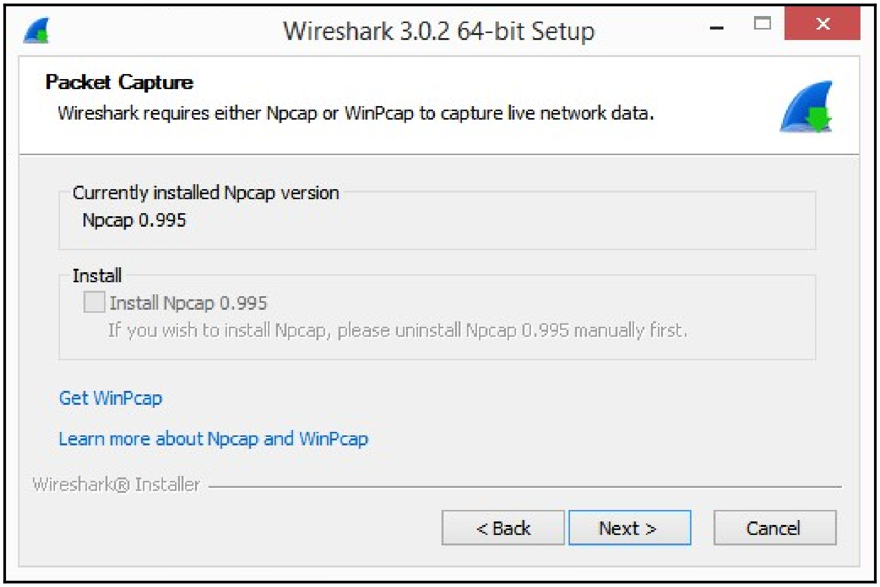
As with most software installations, the user is given some choices. In addition to those listed earlier, the user will have a few more selections to make on capture engines and USB capture before completing the installation.

Capturing packets and completing the installation

Wireshark needs a capture engine to gather network traffic, and will query the system to see if one is present. Wireshark also offers a USB capture, which is optional.

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

- **Packet Capture**: At this point, Wireshark will check whether Npcap or WinPcap is installed. The user is presented with a screen that states **Wireshark requires either Npcap or WinPcap to capture live network data**, as shown here:



Wireshark 3.0.2 64-bit Setup

Packet Capture

Wireshark requires either Npcap or WinPcap to capture live network data.

Currently installed Npcap version

Npcap 0.995

Install

□ Install Npcap 0.995

If you wish to install Npcap, please uninstall Npcap 0.995 manually first.

Get WinPcap

Learn more about Npcap and WinPcap

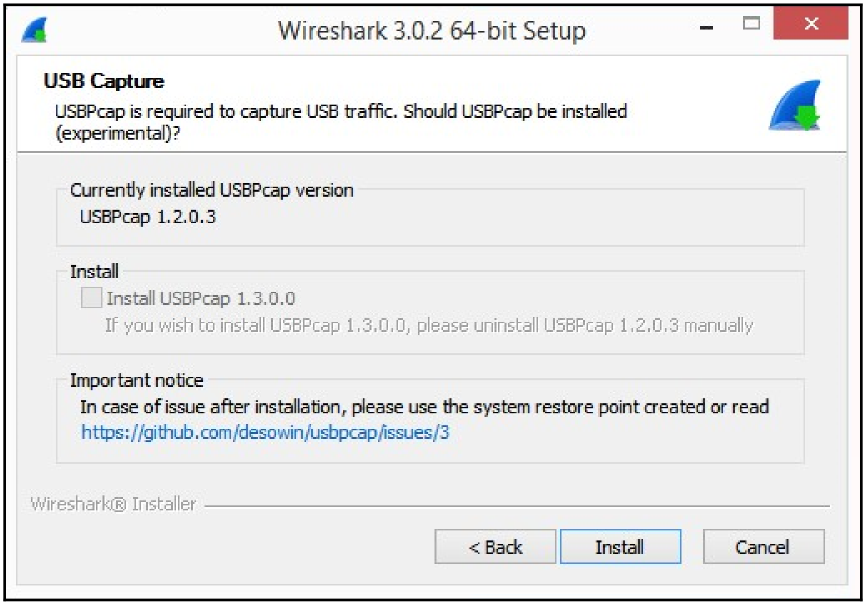
Wireshark® Installer

The Packet Capture Screen

  If you have Windows 7 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

- **USB Capture**: At times, it is necessary to capture USB traffic. This option checks to make sure you have the USBPcap currently installed and gives you an option to install it, which is shown as follows:



USB Capture

USBPcap is required to capture USB traffic. Should USBPcap be installed (experimental)?

Currently installed USBPcap version

USBPcap 1.2.0.3

Install

□ Install USBPcap 1.3.0.0

If you wish to install USBPcap 1.3.0.0, please uninstall USBPcap 1.2.0.3 manually

Important notice

In case of issue after installation, please use the system restore point created or read https://github.com/desowin/usbpcap/issues/3

The USB capture screen

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

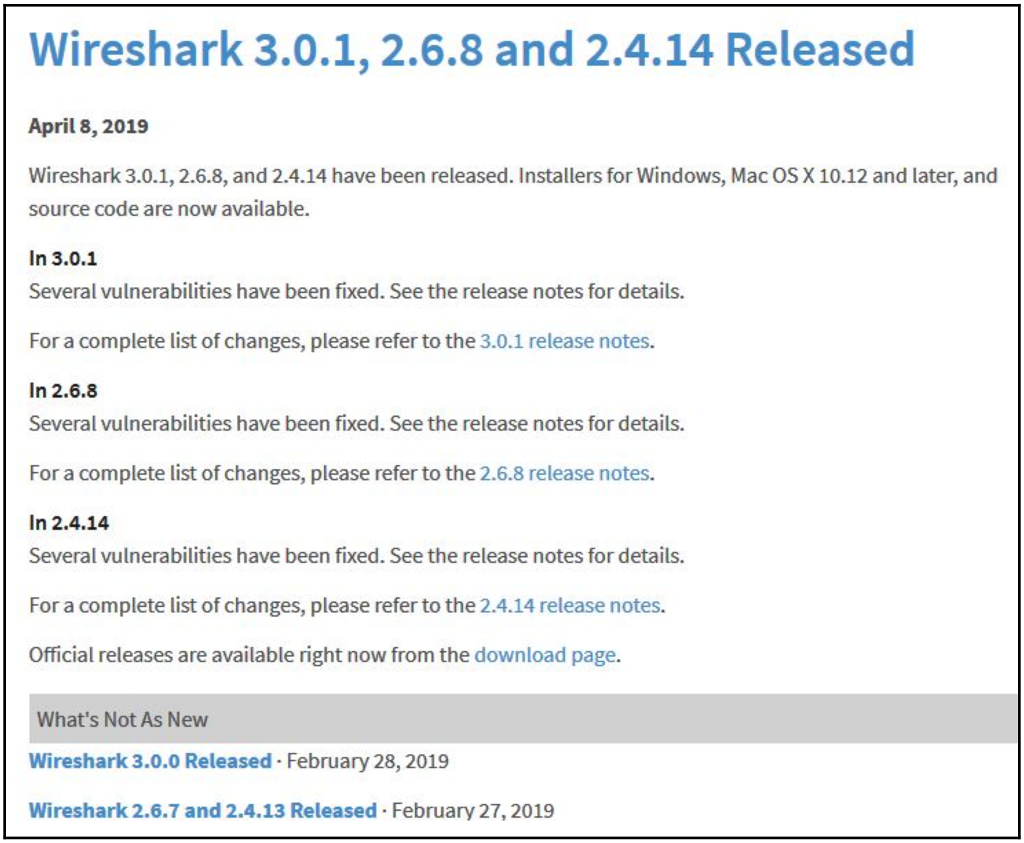
- **Completing Wireshark Setup**: Once you have made all of your selections, Wireshark will present a notification that the process has completed. The screen will show the output of the files extracted during the installation. At this time, you can choose to **Run Wireshark**. In addition, you can also 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 at the Wireshark home page.

Reviewing the resources available at Wireshark.org

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.

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 here:



Wireshark 3.0.1, 2.6.8 and 2.4.14 Released

April 8, 2019

Wireshark 3.0.1, 2.6.8, and 2.4.14 have been released. Installers for Windows, Mac OS X 10.12 and later, and source code are now available.

In 3.0.1
Several vulnerabilities have been fixed. See the release notes for details.

For a complete list of changes, please refer to the 3.0.1 release notes.

In 2.6.8
Several vulnerabilities have been fixed. See the release notes for details.

For a complete list of changes, please refer to the 2.6.8 release notes.

In 2.4.14
Several vulnerabilities have been fixed. See the release notes for details.

For a complete list of changes, please refer to the 2.4.14 release notes.

Official releases are available right now from the download page.

What's Not As New

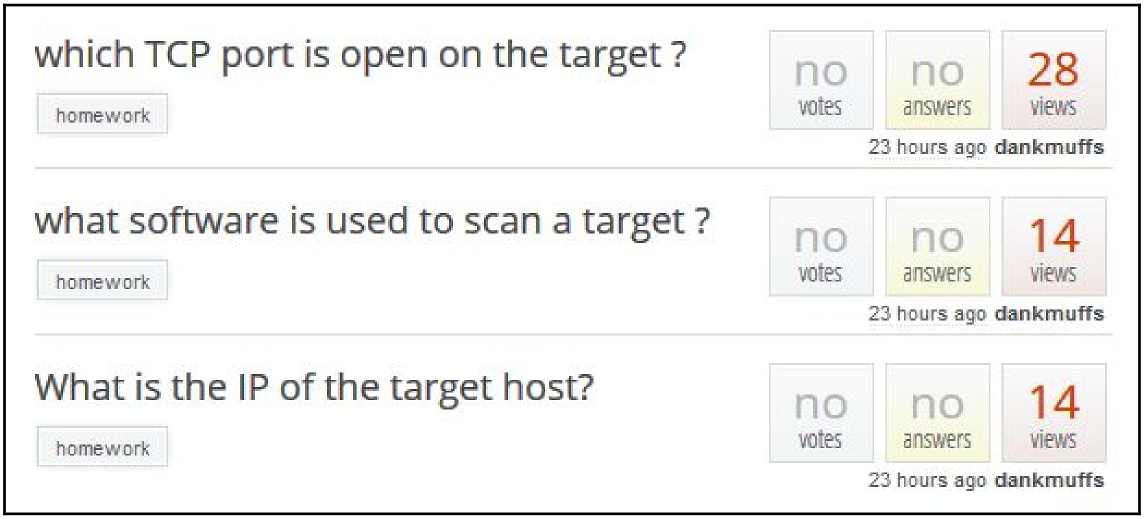
Wireshark 3.0.0 Released · February 28, 2019

Wireshark 2.6.7 and 2.4.13 Released · February 27, 2019

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. You will also 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, various online tools, the Wireshark Wiki, Bug Tracker, and Mailing Lists that help you keep up to date on Wireshark. Users are encouraged to post questions to the forum where you can view the question and registered users can post a response. As shown in the following diagram, there are several topics to investigate:



which TCP port is open on the target ?

homework

no votes | no answers | 28 views

23 hours ago dankmuffs

what software is used to scan a target ?

homework

no votes | no answers | 14 views

23 hours ago dankmuffs

What is the IP of the target host?

homework

no votes | no answers | 14 views

23 hours ago dankmuffs

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 to **Get involved**, **Developers guide**, **Browse the code**, and **the Latest build**.

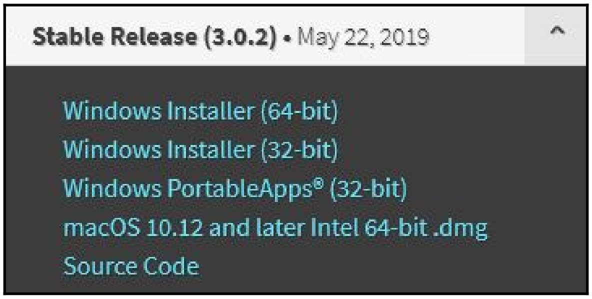
Everyone needs a sponsor. The **Our Sponsor** menu choice takes you to Riverbed's home page, which is an IT company that provides a variety of products and services.

SharkFest is the Wireshark conference, where you can undergo training, gain practical experience, and network among the Wireshark community and the developers that make Wireshark 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 different download options

Once on the **Download** page at `https://www.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 development. The following screenshot shows the choices for downloads under **Stable Release**:



Stable Release (3.0.2) • May 22, 2019

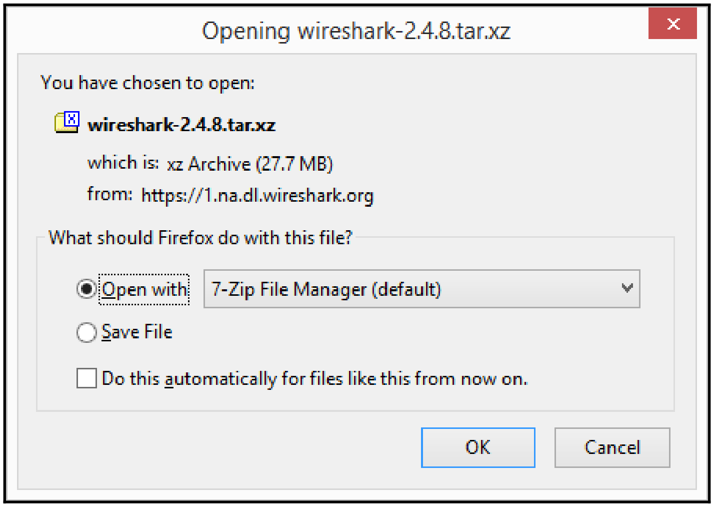
Windows Installer (64-bit)
Windows Installer (32-bit)
Windows PortableApps® (32-bit)
macOS 10.12 and later Intel 64-bit .dmg
Source Code

Choices for downloads under stable release

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

- **Windows Installer (64-bit)**: This provides a standard download for a 64-bit Windows OS.
- **Windows Installer (32-bit)**: This provides a standard download for a 32-bit Windows OS.

- **Wireshark PortableApps (32-bit)**: This is an option that you can run from a flash drive for troubleshooting without having to install on a system. However, the app is limited for use only with Windows 2000, XP, or Vista.
- **macOS 10.12 and later 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. 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 you can keep your source updated. The following is a screenshot of the **Source Code** option on the **Download** page:



Opening wireshark-2.4.8.tar.xz

You have chosen to open:

wireshark-2.4.8.tar.xz

which is: xz Archive (27.7 MB)

from: https://1.na.dl.wireshark.org

What should Firefox do with this file?

● Open with 7-Zip File Manager (default)

○ Save File

□ Do this automatically for files like this from now on.

Source Code option in the download page

Below the stable release, you will find two other options:

- **Old Stable**: This includes the same options as the stable release, but are older versions, rather than the latest build. This might be used if your organization has a strict change management policy that doesn't allow any new software until you run vulnerability testing. In that case, you can select an **Old Stable** option.
- **Development Release**: If you would like to test new features, then you can select one of the choices under the **Development** section.

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

One of the first things that must be done in order to conduct packet analysis is to download and install Wireshark. This chapter went through how Wireshark has support for many different OS so that you can confidently download and install it on your own system. By now, you have a better understanding of the different capture engines and how they in provide a way to gather the traffic from the network before passing the data to the OS.

When you're ready to install Wireshark on a Windows machine, you'll be more confident as you step through all the prompts, from the **Welcome Screen** to completing the installation. So that you are more aware of the many choices for download, we reviewed the various options for the type of file you want, along with what type of release. And finally, you now understand that if you do run into trouble, there is help, as evidenced by the many resources on `https://www.wireshark.org/`.

Now that you have installed Wireshark, you're ready for the next chapter where we explore the Wireshark interface. We will 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. Then, we will 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. Select the best response and then check your answers, which can be found in the *Assessment*:

1. A significant improvement since moving from GTK to Qt is that Wireshark provides a native interface for macOS that doesn't require the use of _____.
    1. MATE
    2. X11
    3. Transum
    4. Capinfos
2. _____is a capture engine originally developed for Unix-like OS, and is baked into Snort, TCPDUMP, and other packet analyzers to grab packets as they come off the network interface.
    1. capinfos
    2. Mate
    3. libpcap
    4. Transum
3. _____ is a lightweight CLI tool that is not as resource intensive.
    1. TShark
    2. mergecap
    3. dftest
    4. androiddump
4. This program will identify and print a packet's geolocation by using an IPv4 and IPv6 addresses.
    1. dftest
    2. TShark
    3. mergecap
    4. mmdbresolve
5. This is the newest capture engine option for Wireshark, with many benefits and features to enhance your packet capture:
    1. AirPcap
    2. NpCap
    3. WinPcap
    4. libpcap

4
Exploring the Wireshark Interface

When you launch Wireshark for the first time, you might find it hard to navigate around the interface until you are familiar with all of the elements required to begin capturing and analyzing traffic. In this chapter, we will step through the Wireshark interface. You will come to understand all of the elements of the welcome page, the sparklines, capture filters, and select interfaces.

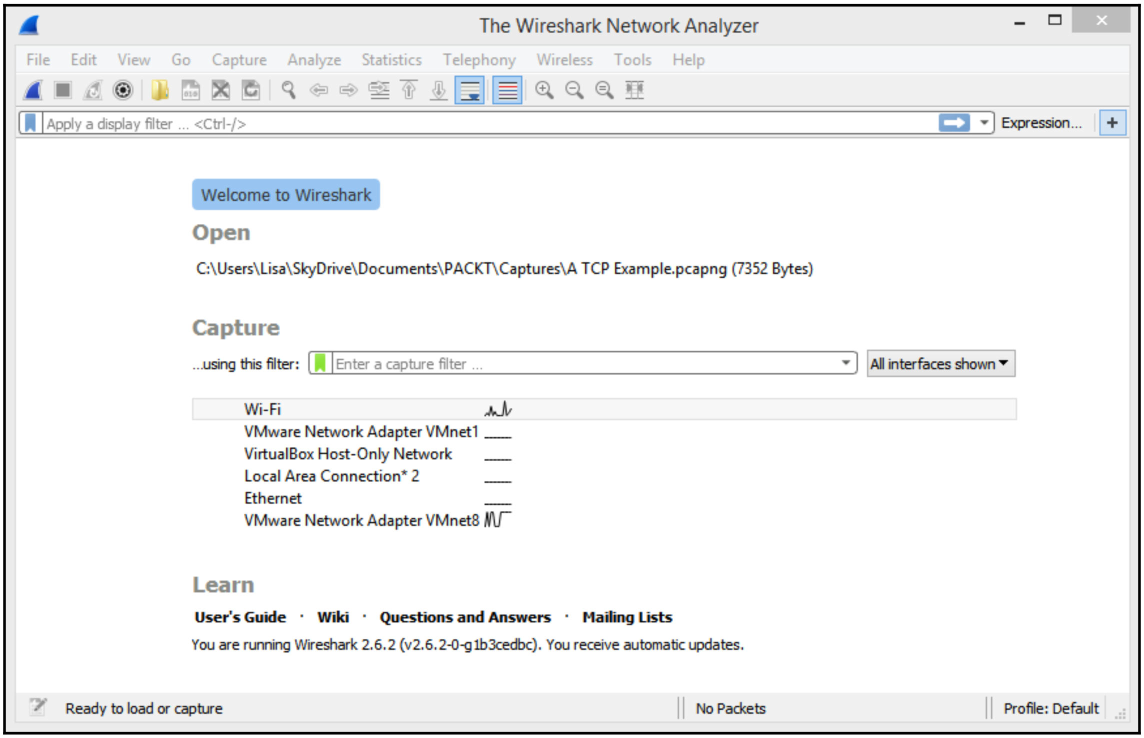
Although Wireshark currently has over ten menu choices, in most cases, you'll find that there are a few that are more commonly accessed. We'll take a look at those so that you are more confident when moving about the interface. We'll examine the **File** menu, where you will not only open a packet capture, but can also save, print, and export the capture. We will 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 you can learn how to customize the look and feel of the Wireshark interface.

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

- Understanding the welcome screen
- Exploring the **File** menu options
- Discover the **Edit** menu options
- Grasping the **View** menu options

Understanding the Wireshark welcome screen

When you first launch Wireshark, you may think that there isn't much on the welcome screen, as shown here:


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
+
Welcome to Wireshark
Open
Capture
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists

Apply a display filter ... <Ctrl-/>

C:\Users\Lisa\SkyDrive\Documents\PACKT\Captures\A TCP Example.pcapng (7352 Bytes)

...using this filter: Enter a capture filter ... All interfaces shown

Wi-Fi
VMware Network Adapter VMnet1
VirtualBox Host-Only Network
Local Area Connection* 2
Ethernet
VMware Network Adapter VMnet8

You are running Wireshark 2.6.2 (v2.6.2-0-g1b3cedbc). You receive automatic updates.

Ready to load or capture | No Packets | Profile: Default

The Wireshark welcome screen

While it is a streamlined interface, you will find that there is everything you need to begin capturing packets and analyzing traffic.

Across the top, you will find the menu choices. If you don't have a capture file loaded, you will see the menu choices that are all available; however, the icons may be dimmed. The icons will become active once you have a packet capture open or are actively capturing packets.

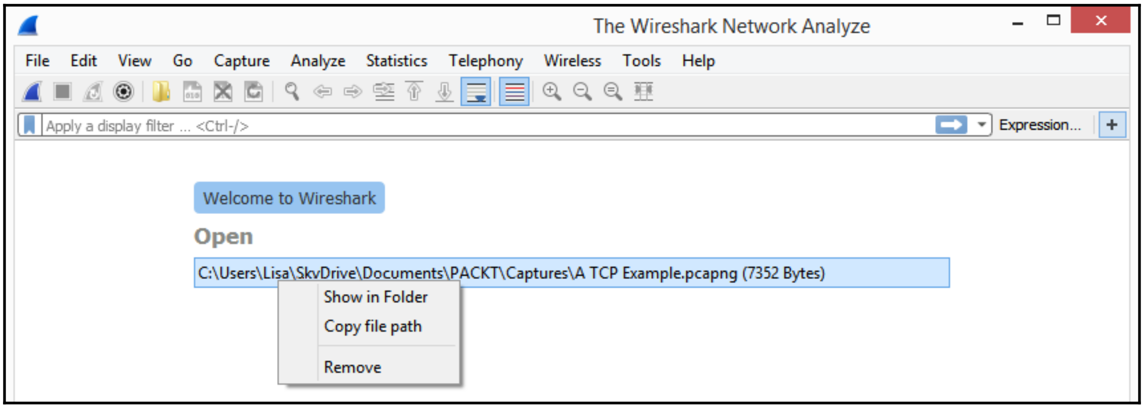
One of the first things you will do while in Wireshark is open a capture. The next section explains the many options available when opening a packet capture.

Opening files

Below the icons, you will see a banner reading **Welcome to Wireshark**. Underneath the banner, you will see the label **Open**, which will identify any previously opened packet captures that are available. If you right-click on a file, you will have the following choices:

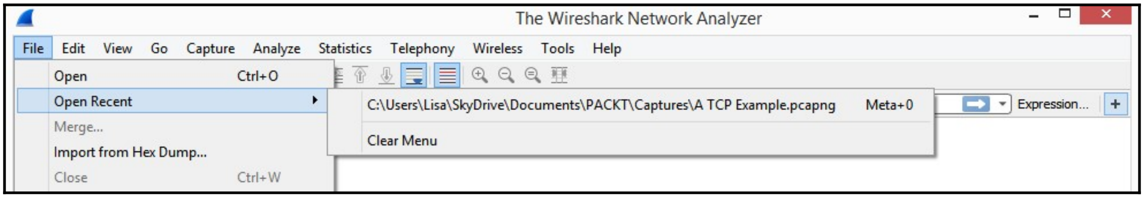
- **Show in Folder**
- **Copy file path**
- **Remove**

While in Windows, if you select **Show in Folder**, as shown here, you can select a file and then drag it onto the Wireshark screen and the file will open:


The Wireshark Network Analyze
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+
Welcome to Wireshark
Open
C:\Users\Lisa\SkyDrive\Documents\PACKT\Captures\A TCP Example.pcapng (7352 Bytes)
Show in Folder
Copy file path
Remove

Right-click and Show in Folder

Once you begin capturing packets, you may have a dozen or so files in the **Open** file area. Although the files are shortcuts for ease of access, they may be distracting:


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Open Ctrl+O
Open Recent
Merge...
Import from Hex Dump...
Close Ctrl+W
C:\Users\Lisa\SkyDrive\Documents\PACKT\Captures\A TCP Example.pcapng Meta+0
Clear Menu
Expression...
+

The Clear Menu choice

If you want to remove the files, go to the menu **File** | **Open Recent** | **Clear Menu**, as shown in the preceding screenshot.

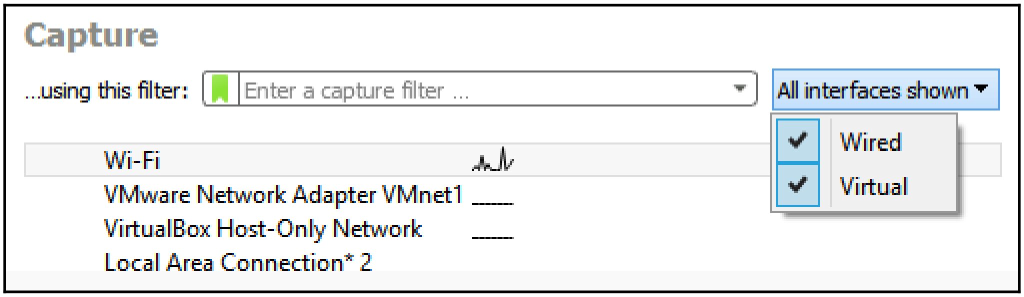
Next, let's take a look at the options for gathering network traffic.

Capturing traffic

If you are getting ready to capture traffic, you'll want to set Wireshark up properly. You can find the **Capture** label in the middle of the screen. Once there, you can apply a capture filter in the space provided. Below the capture filter area, you'll see a list of interfaces, with a moving symbol next to the active interface(s). The moving symbol is called a sparkline, which identifies an interface, and the lines represent actively exchanging data.

The capture filter allows you to add a capture filter. 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 may miss the traffic that can help with your analysis.

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



Capture options

In the list of interfaces, you will see the various connections. One of the interfaces may be the USBPcap, which will be available if you installed the USBPcap driver. This is a fairly new option that may be helpful to use during troubleshooting.

The last thing on the interface you'll find is a few links that can provide more information.

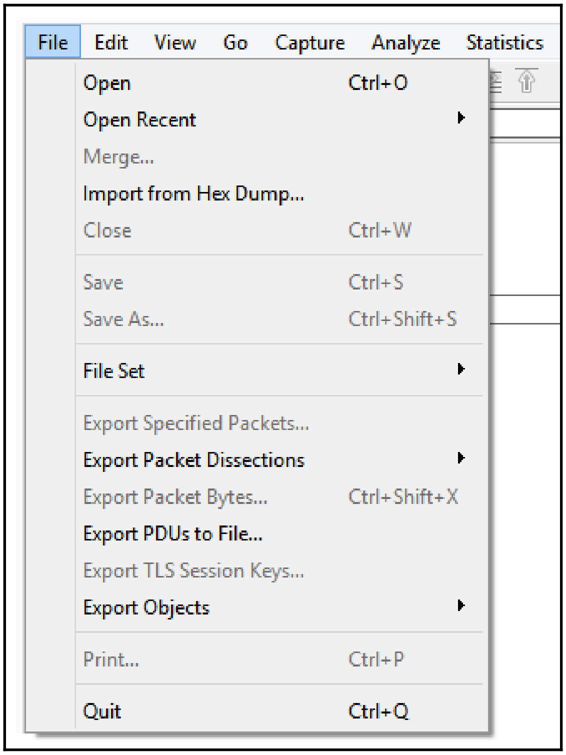
Learning about Wireshark

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

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 covers what is possible in the **File** menu.

Exploring the File menu

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



The File menu

In this section, we'll walk through the many options found in the **File** menu. If you would like to follow along, go to `https://www.cloudshark.org/captures/0012f52602a3`. Download the original file, open the packet capture file, `HTTP.cap`, in Wireshark, and select frame 36.

There are many options found in the **File** menu. Let's begin with ways to locate and open a file, save a capture, and what options are available when you close a file.

Opening a file, close, and save

This first section has many choices for locating and opening files so you can begin your analysis:

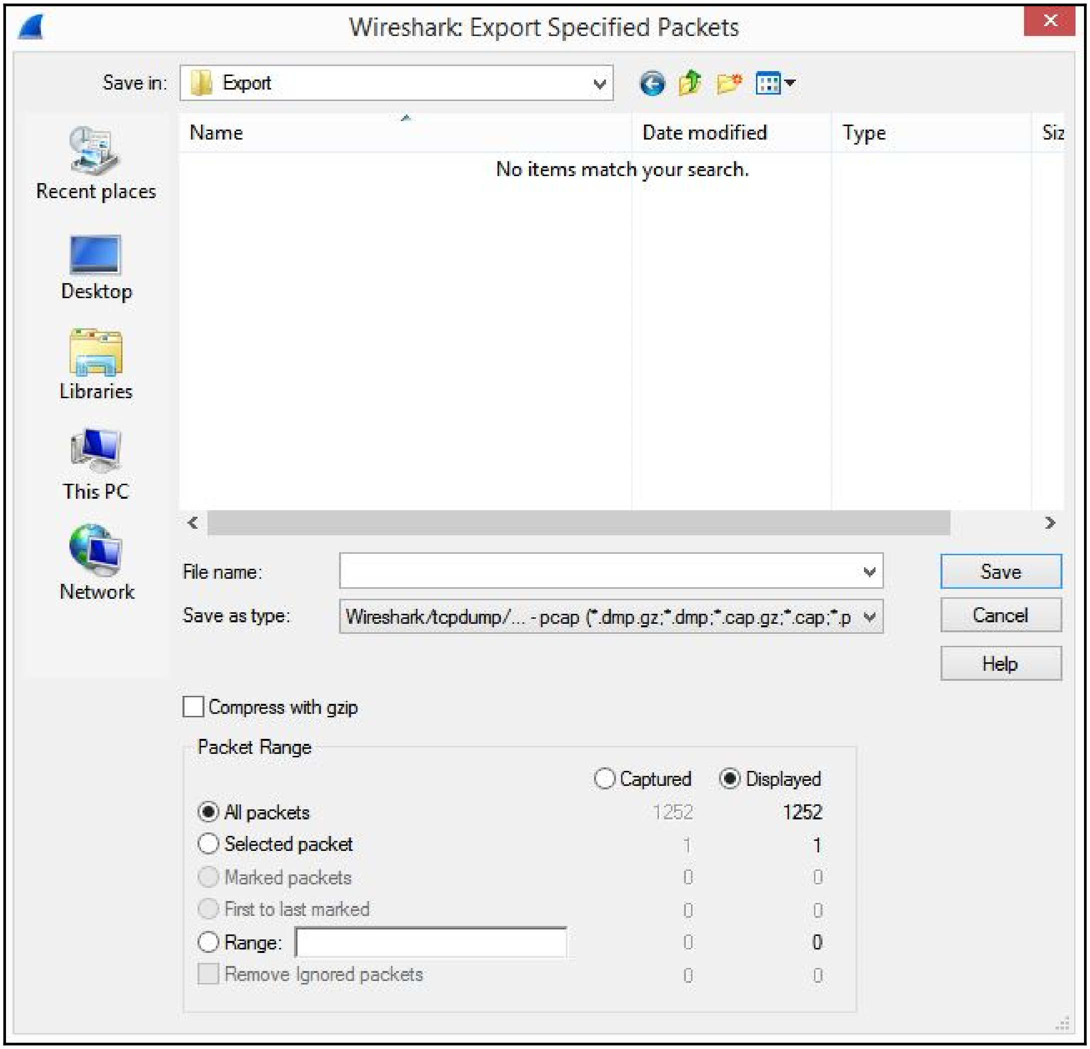
- **Open** will launch a dialog box that will allow you to select any file; it'll go to the location of that file and allow you to select that file.
- **Open Recent** will list the recently accessed files.
- **Merge** will allow you to merge a file with the capture you have open. When merging, it's important 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 through selecting the appropriate choices when importing the file.
- **Close** will close the current capture. Prior to closing, Wireshark will ask you if you'd like to save the file.
- **Save** allows you to save the current file. This would be 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.
- **File Set** 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.

This next **File** menu section takes a look at the many ways to export parts of a capture.

Exporting packets, bytes, and objects

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

- **Export Specified Packets**: This provides a wide range of options that include only displayed packets, a range of packets, and marked packets, as shown here:



Save in: Export

File name:

Save as type: Wireshark/tcpdump/... - pcap (*.dmp.gz;*.dmp;*.cap.gz;*.cap;*.p

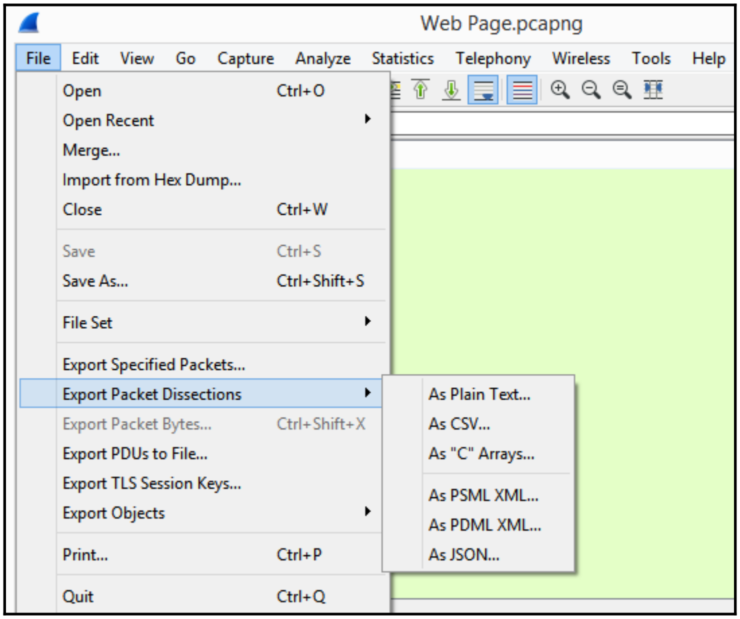
□ Compress with gzip

Packet Range

	○ Captured	● Displayed
● All packets	1252	1252
○ Selected packet	1	1
○ Marked packets	0	0
○ First to last marked	0	0
○ Range:	0	0
□ Remove Ignored packets	0	0

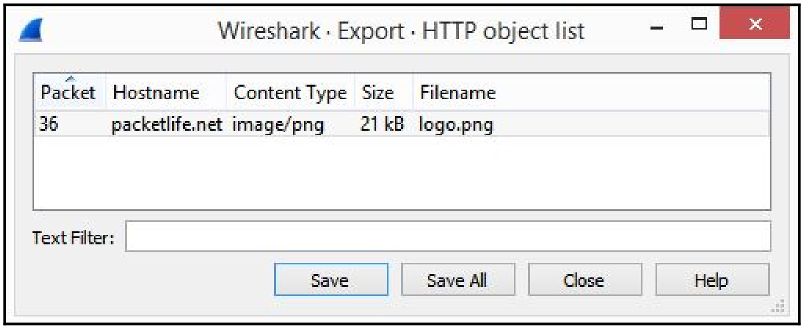
The Export Specified Packets dialog box

- **Export Packet Dissections**: This offers many choices to export, as shown in the screenshot, including CVS, plaintext, and JSON:



The Export Packet Dissections menu

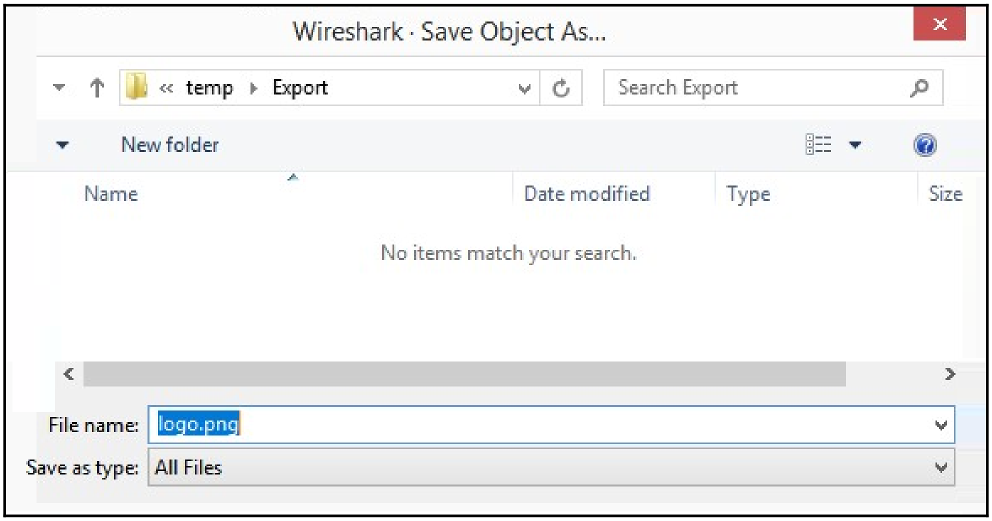
- **Export Packet Bytes**: This feature exports the packet bytes into C arrays so you can import the stream into a C program.
- **Export PDUs to File**: This menu choice offers many selections to export; however, this feature may not show a usable output and may only work with specific applications.
- **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 **TLS** (short for **Transport Layer Security**) keys to save.
- **Export Objects**: This exports objects found within the file, such as images, documents, and executables. For frame 36, select **Export Objects | HTTP**, which will display a list of objects found, as shown here:



Packet	Hostname	Content Type	Size	Filename
36	packetlife.net	image/png	21 kB	logo.png

The Export Objects—HTTP dialog box

Within this window, you can select **Save**, **Save All**, or even use the **Text Filter**:



File name: logo.png
Save as type: All Files

The Export Objects—Save As dialog box

I selected **Save** and then navigated to a temporary folder, `Export`. For the filename, I selected `logo.png`. I included the extension to ensure the object is saved in the correct format. When done, navigate to the folder and open the image and you should see the Packet Life logo.

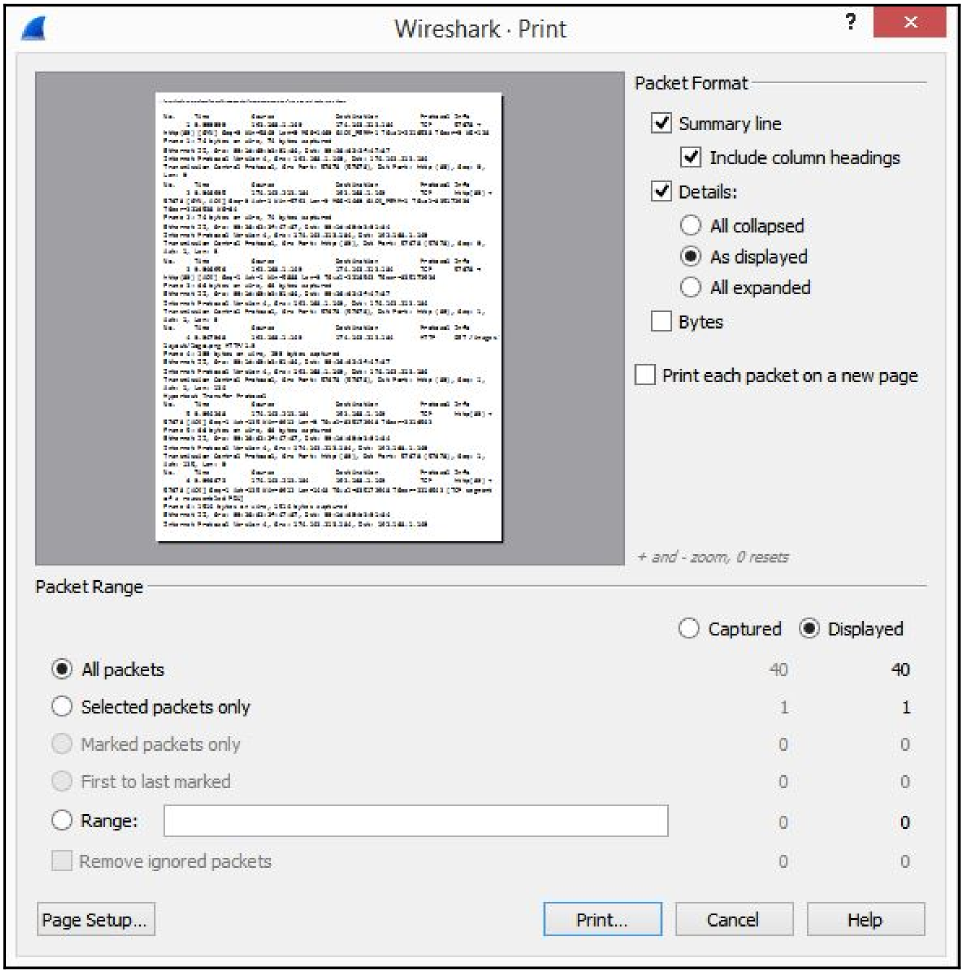
If there are other objects within the capture, you can save them in a similar manner.

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

Printing packets and closing Wireshark

Within this final section, let's take a look at the last two options, **Print** and **Quit**:

- **Print**: While examining packets, Wireshark offers many ways to print different sections of the capture. You can print all of the packets, selected packets only, or a range of packets to PDF, which you can then include in a report. Once you select **Print**, you will see the following:



Wireshark · Print

Packet Format

☑ Summary line

☑ Include column headings

☑ Details:

○ All collapsed

● As displayed

○ All expanded

☐ Bytes

☐ Print each packet on a new page

+ and - zoom, 0 resets

Packet Range

	○ Captured	● Displayed
● All packets	40	40
○ Selected packets only	1	1
○ Marked packets only	0	0
○ First to last marked	0	0
○ Range:	0	0
☐ Remove ignored packets	0	0

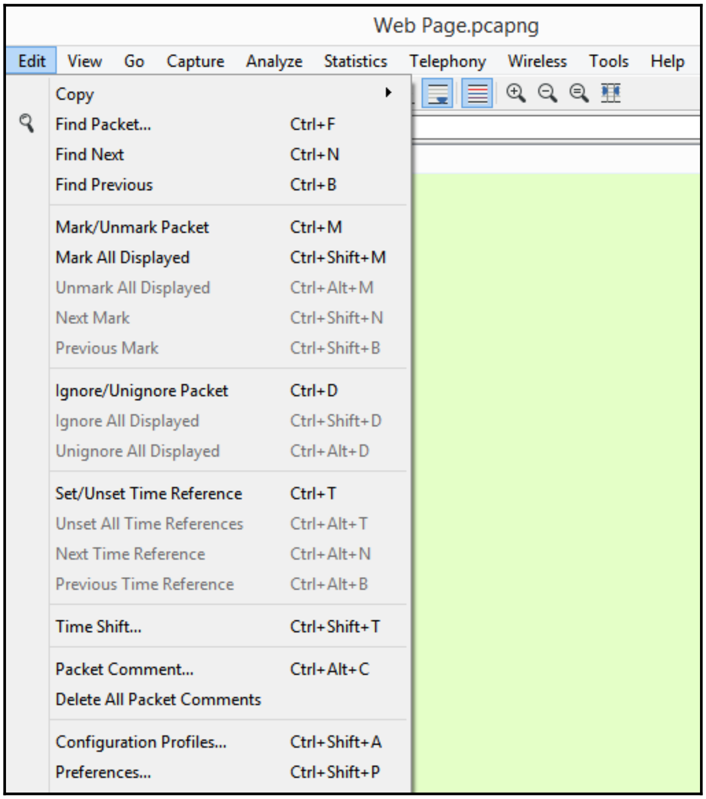
The File | Print option

- **Quit**: Once you are done using Wireshark, 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.

In addition to the **File** menu, you will want to work with your capture file. The following section will cover the **Edit** menu to help you can discover the many possibilities 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 and provide detailed information on creating a configuration profile, or modify preferences. The following is a screenshot of the **Edit** menu:


Web Page.pcapng
Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Copy
Find Packet... Ctrl+F
Find Next Ctrl+N
Find Previous Ctrl+B
Mark/Unmark Packet Ctrl+M
Mark All Displayed Ctrl+Shift+M
Unmark All Displayed Ctrl+Alt+M
Next Mark Ctrl+Shift+N
Previous Mark Ctrl+Shift+B
Ignore/Unignore Packet Ctrl+D
Ignore All Displayed Ctrl+Shift+D
Unignore All Displayed Ctrl+Alt+D
Set/Unset Time Reference Ctrl+T
Unset All Time References Ctrl+Alt+T
Next Time Reference Ctrl+Alt+N
Previous Time Reference Ctrl+Alt+B
Time Shift... Ctrl+Shift+T
Packet Comment... Ctrl+Alt+C
Delete All Packet Comments
Configuration Profiles... Ctrl+Shift+A
Preferences... Ctrl+Shift+P

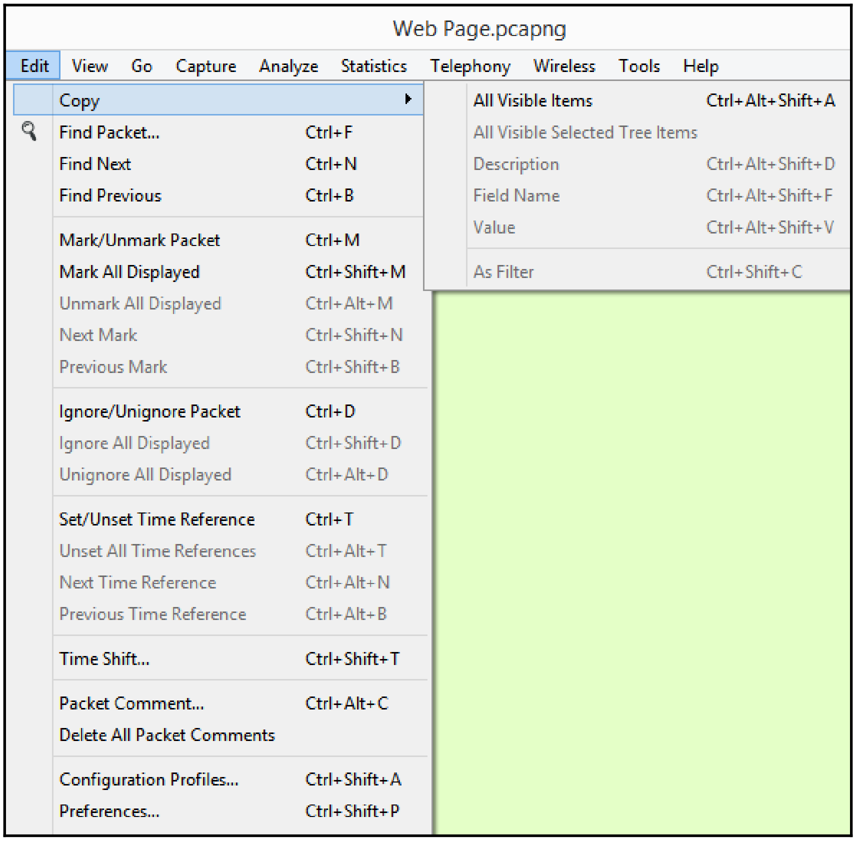
The Edit menu

As you can see in the screenshot, there are many options. The following discussion outlines ways to copy various items and find packets within Wireshark.

Copying items and finding packets

While analyzing packets, you may see an item or value you would like to copy. Wireshark makes this easy to accomplish as the **Copy** menu choice has many submenus to further define copy options. In addition, we'll see how we can locate a specific packet or a string value within the capture.

Copy allows you to copy various items to the clipboard. For example, in frame 5, expand the IP header and select the source IP address. Go to **Edit** | **Copy** and then expand the selections, as shown here:


Web Page.pcapng
Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Copy
Find Packet... Ctrl+F
Find Next Ctrl+N
Find Previous Ctrl+B
Mark/Unmark Packet Ctrl+M
Mark All Displayed Ctrl+Shift+M
Unmark All Displayed Ctrl+Alt+M
Next Mark Ctrl+Shift+N
Previous Mark Ctrl+Shift+B
Ignore/Unignore Packet Ctrl+D
Ignore All Displayed Ctrl+Shift+D
Unignore All Displayed Ctrl+Alt+D
Set/Unset Time Reference Ctrl+T
Unset All Time References Ctrl+Alt+T
Next Time Reference Ctrl+Alt+N
Previous Time Reference Ctrl+Alt+B
Time Shift... Ctrl+Shift+T
Packet Comment... Ctrl+Alt+C
Delete All Packet Comments
Configuration Profiles... Ctrl+Shift+A
Preferences... Ctrl+Shift+P
All Visible Items Ctrl+Alt+Shift+A
All Visible Selected Tree Items
Description Ctrl+Alt+Shift+D
Field Name Ctrl+Alt+Shift+F
Value Ctrl+Alt+Shift+V
As Filter Ctrl+Shift+C

Copy options

The **Copy** submenu has the following options to select from:

- **Value**: This will copy the `168.1.140` IPv4 address.
- **As Filter**: This will create a filter based on the IPv4 address you selected or any other value. You can then paste the filter in the display filter area, press *Enter*, and Wireshark will run the filter.

Within the **Edit** menu choice, there are a few groupings of selections. We'll start with the first grouping, which offers ways to find packets:

- **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.

Marking or ignoring packets

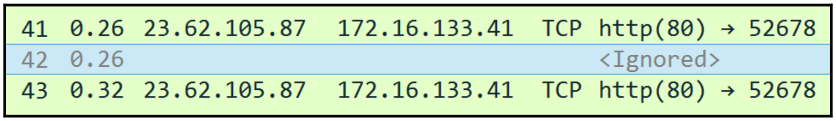
While working with packets, you might find and mark packets that are interesting, so you can return to them at a later date. In addition, you may want to ignore specific packets.

This next grouping of selections offers ways to mark 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 will mark all displayed packets, meaning if you used 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 packets are marked, this option allows you to move to the next marked packet.
- **Previous Mark**: When packets are marked, this option allows you to navigate back to the previous marked packet.

In addition to marking packets to identify items of interest, you may want to ignore specific packets. The following shows 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, and it won't show up in statistics or a flow graph; it's simply ignored. Once you select ignore, the packet line will have a reference reading **<Ignored>**, as shown here:



```
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
```

Using the Ignore Packet option

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

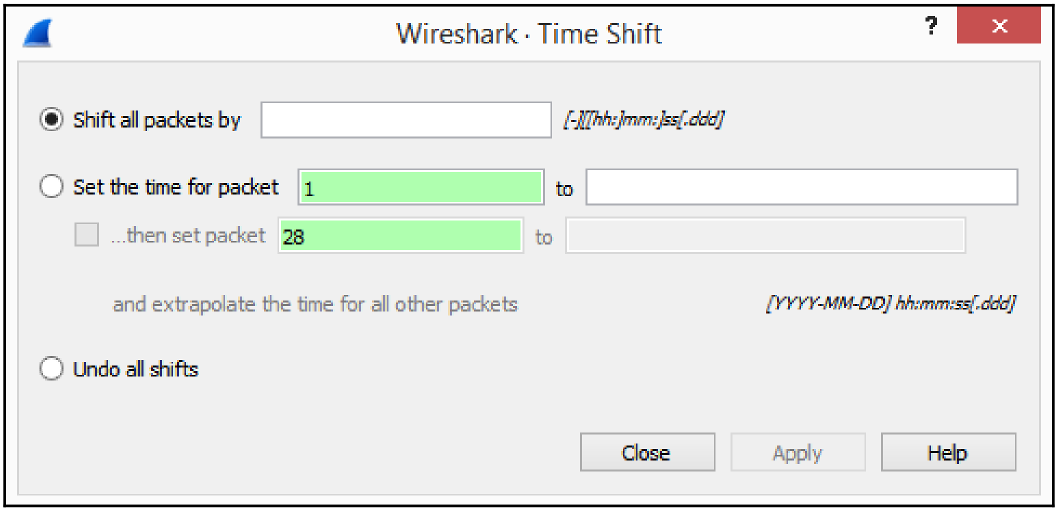
While some packets may be ignored as they hold no value in the analysis, you may want to use some method to determine delays, as we'll see next.

Setting a time reference

In your analysis, you may 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 time references:

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

- **Time Shift**: This is an option you can use when you need to adjust the time reference. For example, if you are examining two captures that each used a different file format—that is, one file used **NTP** (short for **Network Time Protocol**) and the other file used **PTP** (short for **Precision Timing Protocol**)—you may want to do a time shift. If you select this option, it will launch a dialog box where you can set your values, as shown here:



● Shift all packets by [-][[hh:]mm:]ss[.ddd]

○ Set the time for packet 1 to

□ ...then set packet 28 to

and extrapolate the time for all other packets [YYYY-MM-DD] hh:mm:ss[.ddd]

○ Undo all shifts

The Time Shift option

The last option shows where you can undo all shifts if you get unexpected results.

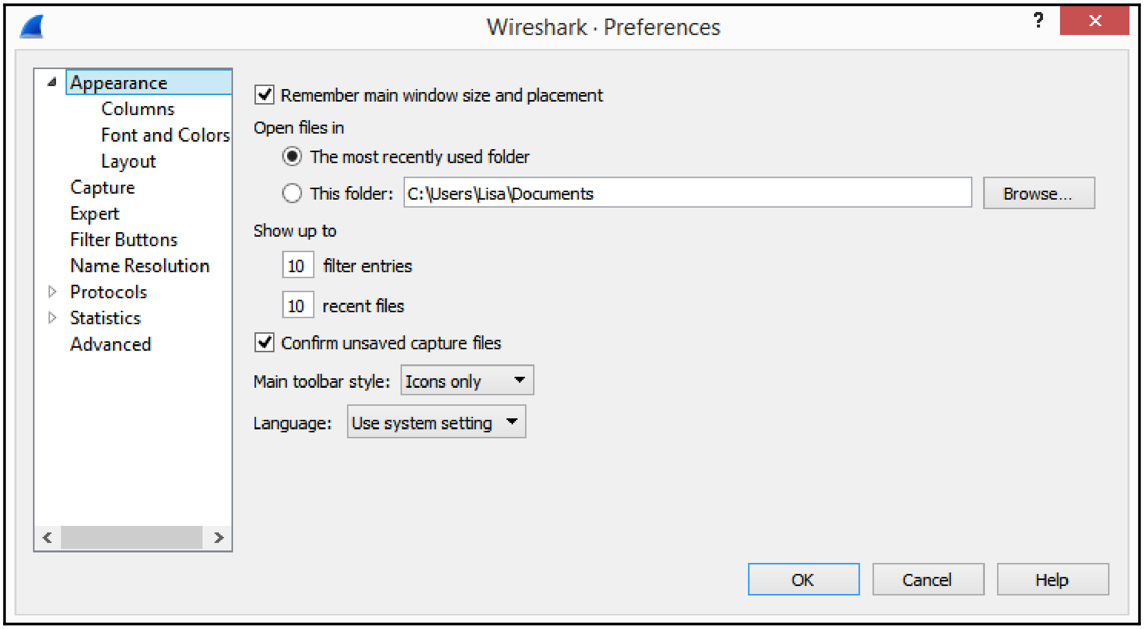
Now that we understand how we can reference or shift time in Wireshark, let's take a look at ways to personalize your work area.

Personalizing your work area

While working with a capture, you can record your changes by using comments. In addition, you can fine-tune the interface by creating a tailored configuration profile and/or modify individual settings using the **Preferences** menu:

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

- **Configuration Profile**: This allows you to create a customized profile, specific to your workflow. This is a powerful feature, as you can create several profiles, so they can be used for specific applications or clients.
- **Preferences**: This brings up the **Preferences** dialog box where you can alter the appearance and elements that influence the functionality of Wireshark. Here, you can adjust the font and color or even the layout, as shown in the following screenshot:



☑ Remember main window size and placement

Open files in

● The most recently used folder

○ This folder: C:\Users\Lisa\Documents

Show up to

10 filter entries

10 recent files

☑ Confirm unsaved capture files

Main toolbar style: Icons only

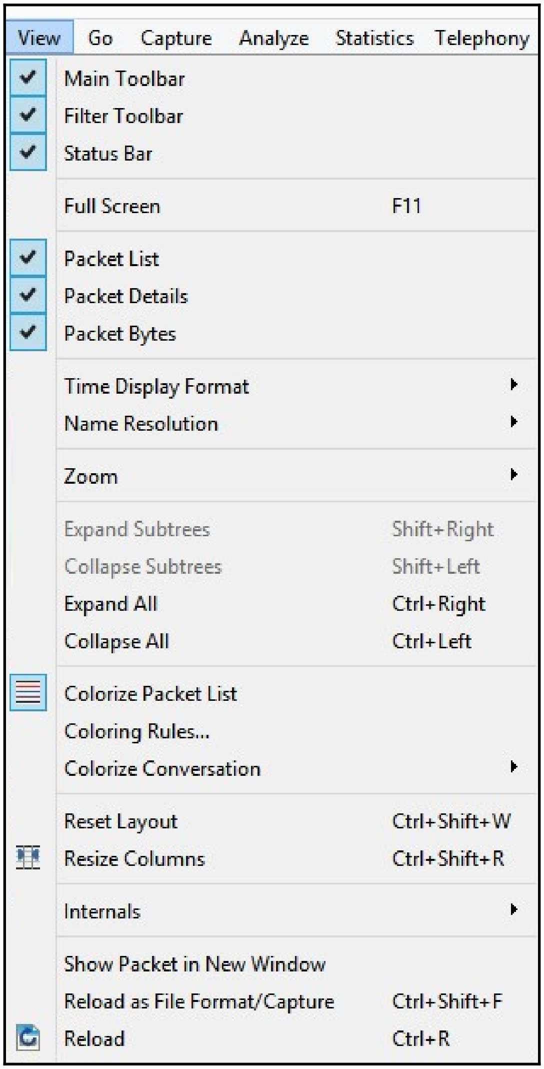
Language: Use system setting

Wireshark Preferences

Although the **Edit** menu is widely used, let's take a look at the **View** menu, so you can see the many ways 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 the subtrees, or show a packet in a separate window:



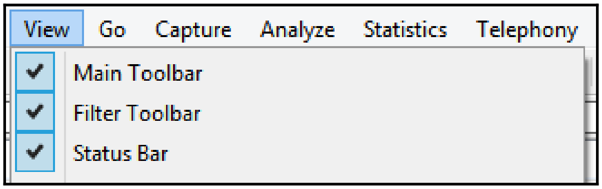
The View menu

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.cap` file.

Enhancing the interface

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

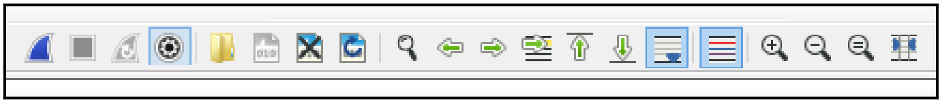
The toolbar section represents a grouping where similar items are combined in many menus. Once in this section, you will see a list of the three available toolbars that are currently available, as shown here:



The View menu—toolbars

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:



Main Toolbar

- **Filter Toolbar**: This is where you will find the display filter.
- **Status Bar**: This is found at the bottom of the Wireshark screen. The **Status Bar** tells how many packets are captured and how many are displayed, what profile is applied, and the name of the file.
- **Full Screen**: This is used when we want Wireshark to go full screen, which will fill the current window.

Once you get used to the toolbars, you will see they provide a handy way to help you to 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.

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

Adjusting time formats and name resolution

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

Once you expand the **Time Display Format** menu choice, you will see several options with how you want your time displayed, which include the following:

- **Date and Time of Day**
- **Year, Day of Year, and Time of Day**
- **Time of Day**
- **Seconds Since 1970-01-01**

When doing an analysis, you will most likely use a format that helps you to visualize gaps in transmission. In that case, the following 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 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 how many seconds have passed since the *previously displayed packet*, which will more accurately show gaps in time.

Time precision is also a consideration. When selecting a format, you have a choice in how many decimal places are displayed, as shown here:

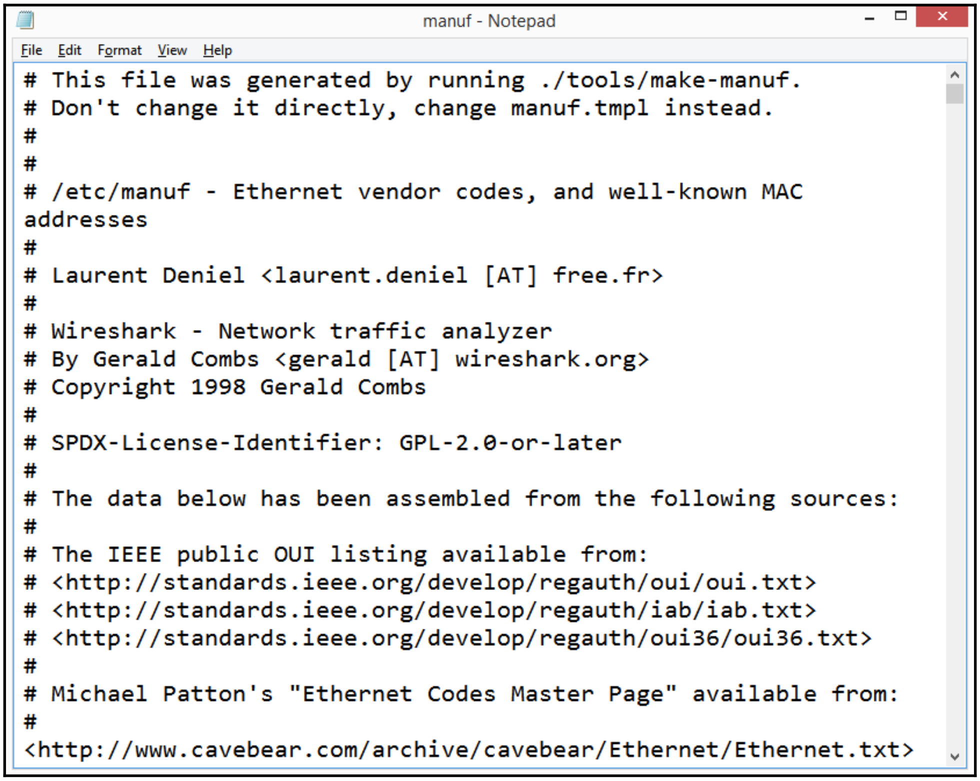
- **Automatic (from Capture File)**
- **Seconds**
- **Tenths of a second**
- **Hundredths of a second**
- **Milliseconds**
- **Microseconds**
- **Nanoseconds**

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

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 will outline the options available to resolve names and the rationale behind why you would select each one.

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 problems 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**), and this comes from the `manuf.txt` file, as shown here:



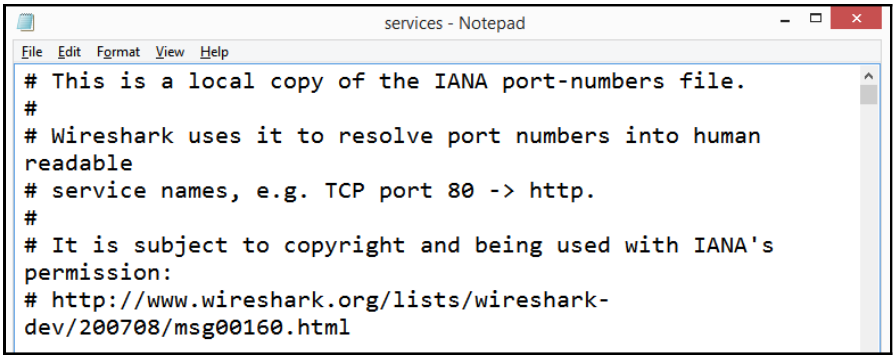
```
# This file was generated by running ./tools/make-manuf.
# Don't change it directly, change manuf.tmpl instead.
#
#
# /etc/manuf - Ethernet vendor codes, and well-known MAC
addresses
#
# Laurent Deniel <laurent.deniel [AT] free.fr>
#
# Wireshark - Network traffic analyzer
# By Gerald Combs <gerald [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>
```

The manuf file listing NIC card vendors

To **Resolve** the **Transport Address** (or port number), Wireshark consults the services file, which is a text file that holds a list of services and the associated port number. The list uses the IANA port-numbers file for consistency.

For example, the service **smtp** 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 following is a screenshot of the `services.txt` file, which is found in the Wireshark folder:



```
# This is a local copy of the IANA port-numbers file.
#
# Wireshark uses it to resolve port numbers into human
readable
# service names, e.g. TCP port 80 -> http.
#
# It is subject to copyright and being used with IANA's
permission:
# http://www.wireshark.org/lists/wireshark-
dev/200708/msg00160.html
```

The services file listing ports and associated services

The **Resolve Network Addresses** will resolve a hostname to an IP address. Normally, this option is not checked because, if it is, Wireshark will contact the DNS server(s) to do the resolution and cause a lot of additional 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.

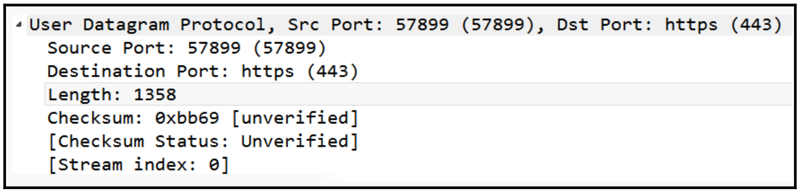
When working with a capture, there are ways to enhance your view, as we shall see in the next section.

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. With the subtrees, you can do the following:
    - Expand subtrees
    - Collapse subtrees
    - Expand all
    - Collapse all

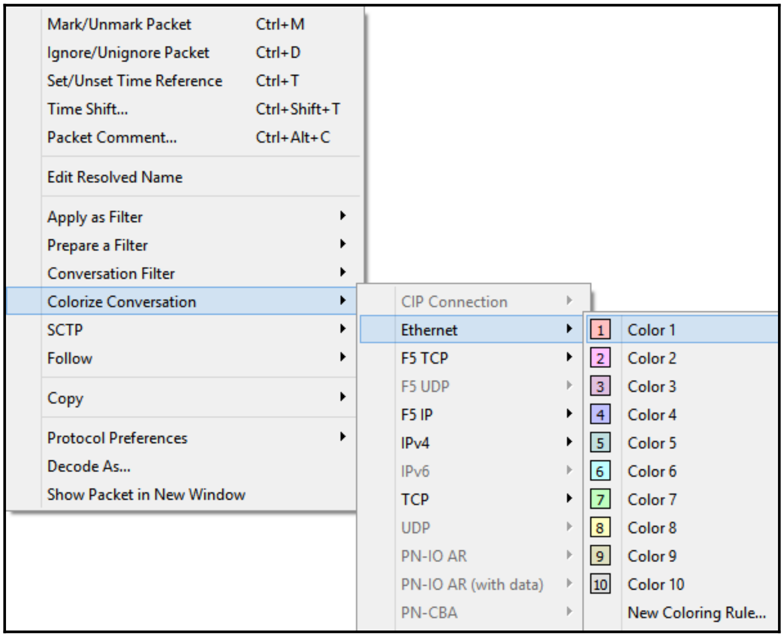
As shown in the following screenshot, the expanded UDP subtree provides a detailed view of all of the field values in the UDP header:



```
◢ 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]
```

A UDP header with expanded subtree

- **Colorize Packet List**: This is a shortcut to turn on or off the coloring rules. This shortcut is also available on the main toolbar (under the **Telephony** menu).
- **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—that is, **Ethernet**, **IPv4**, or **UDP**—along with providing a choice of colors from which you can select, as shown in the following screenshot:



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.

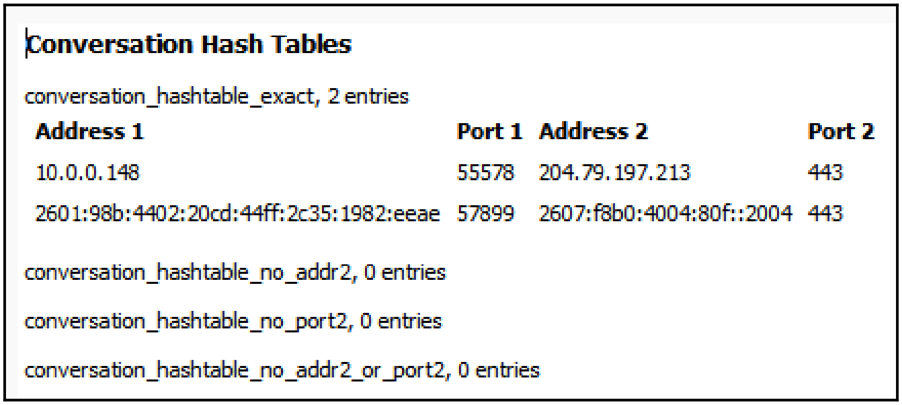
Refreshing the view

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 to view the captured packets:

- **Resize Layout**: This option, when selected will resize the visible panels so they have a uniform appearance.
- **Resize Columns**: Much like you can resize the columns in Excel to autofit to their contents, **Resize Columns** will adjust the columns, so the contents fit. When using IPv4, the columns may adjust nicely, but using IPv6 takes up much more space and may 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:

- **Internals**: The **Internals** menu choice provides advanced options that include the following:
  - **Conversation Hash Tables**: This shows the address and port combinations that identify each conversation, as shown here:



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

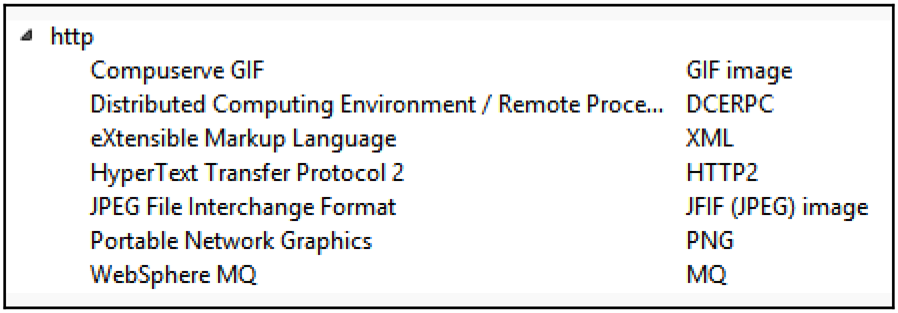
conversation_hashtable_no_port2, 0 entries

conversation_hashtable_no_addr2_or_port2, 0 entries

Conversation Hash Tables information

Select a single packet and then click **Conversation Hash Tables** to bring up the information.

  - **Dissector Tables**: This provides tables of relationships among subdissectors, 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

Dissector Table subset showing HTTP

  - **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.

- **Show Packet in New window**: At times, you may want a single packet in its own window as a popup. This option may be ideal for training or to show a single packet, but doesn't offer any functionality—meaning, you can't right-click or use any shortcuts while in that window.
- **Reload**: This option will reload the capture, which will freshen the capture file. This can be helpful if you have marked packets and manipulated the file already, and you want a fresh start with the file.

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 then took a closer look at commonly accessed menu choices to make navigating around Wireshark easier. First, we evaluated the **File** menu that 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 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. To help you understand how what you see will be dependent 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, which include display options, using multiple files and name resolution, and understanding conversations and endpoints.

Questions

Now, it's time to check your knowledge. Select the best response, then check the answers found in the *Assessment*:

1. Once you begin capturing packets, you may have a dozen or so files in the Open file area. If you want to remove the files, go to the File menu and then choose Open Recent and then the _____ menu.
    1. Clear
    2. Purge
    3. Delete
    4. 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.
    1. Recently Created Epoch
    2. Previously Captured Packet
    3. Beginning of Capture
    4. Previously Displayed Packet
3. ___ is a shortcut to turn on or off the coloring rules. The shortcut is also available on the main toolbar (under the Telephony menu).
    1. Colorize Conversation
    2. Coloring Rules
    3. Stop Color Filters
    4. 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.
    1. File
    2. Edit
    3. View
    4. Go

5. When working with a packet capture, the _____ menu choice edit allows you to find and mark packets, set a time reference, copy and provide detailed information for creating a configuration profile, or modify your preferences.
    1. File
    2. Edit
    3. View
    4. Go

2
Section 2: Getting Started with Wireshark

This section will enable you to tap into the data stream, personalize the Wireshark interface, compare display and capture filters, and review the OSI model and data encapsulation.

This section is comprised of the following chapters:

- `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

When you need to draw liquid from a container, you use a tap. When you tap into the data stream, you capture the data from the network. Once you capture network traffic, you can analyze the packets to understand the traffic flow. In this chapter, we'll review the network architectures and various types of media that may be found on today's networks to help you to 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 you can identify what types of traffic you'll see. We'll also compare and contrast conversations and endpoints, so you understand the difference between the two. Finally, so you can better identify abnormal network behavior, this chapter ends with a discussion on why it's important to baseline network traffic.

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

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

Reviewing the network architecture

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.

In today's networked environments, administrators face numerous challenges to keep the network up and operational. Effective packet analysis begins by 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. And to that end, we will begin our discussion by learning the various types of networks.

Comparing different types of networks

Today's networked environments are complex and can include 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 will influence how data is transmitted and will possibly be the cause of system failures and bottlenecks.

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

Discovering the PAN

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, we also use the term **Wireless Personal Area Network** (**WPAN**).

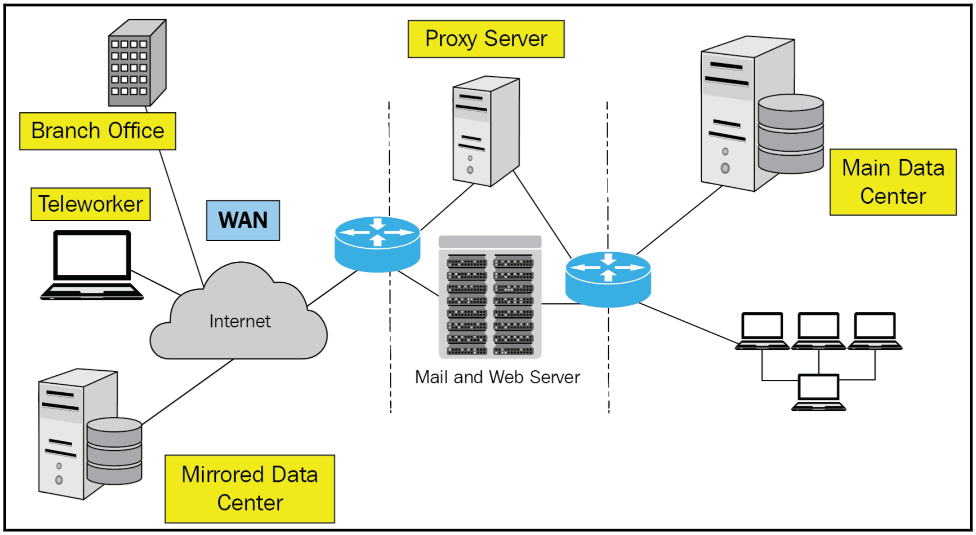
A WPAN is a short-range network that connects personal devices to exchange information using the IEEE 802.15 standard and includes technologies such as Bluetooth, ZigBee, and Ultra-Wide Band.

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 provides an overview of the characteristics of a LAN.

Checking out LANs

A local area network 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 shares resources. The 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:



An example of a networked environment

Within the LAN, the network might have 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 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 local area network in a common entity such as a college, hospital, corporate campus, or military base that has two or more interconnected LANs.

The CAN has a main campus where the central elements of a network reside, such as the data center and telephony, and provides connectivity, data, applications, and services to clients. Away from the main campus, there are remote locations.

Because the CAN, at times, is spread across a larger geographic area such as a city, remote locations 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 telecommunications networks. WANs carry the signal using the **Plain Old Telephone System** (**POTS**), fiber optic cables, wireless transmissions, and satellites.

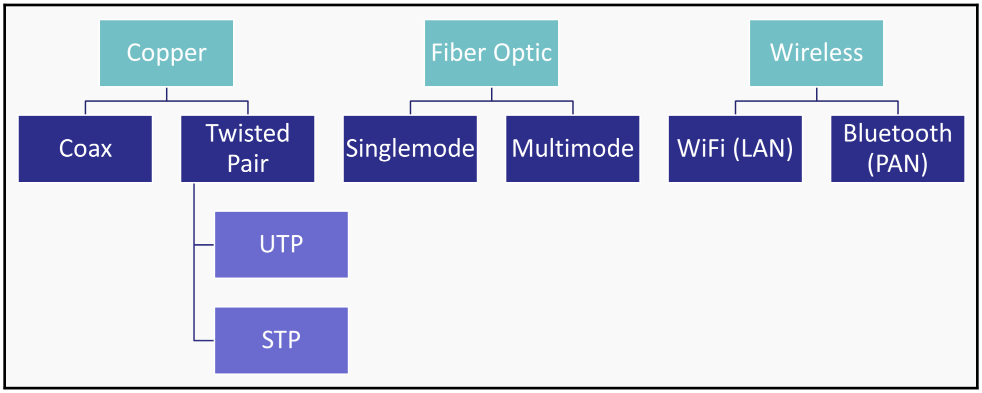
As you can see, there are many different types of networks, from very small PANs to large WANs. In the next section, let's explore each of the different types of media used to carry the signals.

Exploring various types of media

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

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

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



Various types of network media

For enterprise networks, multiple types of media make up the networking environment and include copper ad fiber optic cable, and wireless transmissions. Each media type will influence the data flow.

Let's begin by reviewing copper, which is subdivided into two categories, coaxial and twisted pair.

Exploring copper

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

Coaxial, also called **coax**, was originally used to transmit data on a LAN. 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 but 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 differences in traffic transmitted between the cable modem and router, as Wireshark has a **DOCSIS** (short for **Data Over Cable Service Interface Specification**) dissector for that purpose.

Today, LANs use twisted pair cable, which 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.

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

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 **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 to 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 analyzing a wireless connection.

Discovering wireless

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

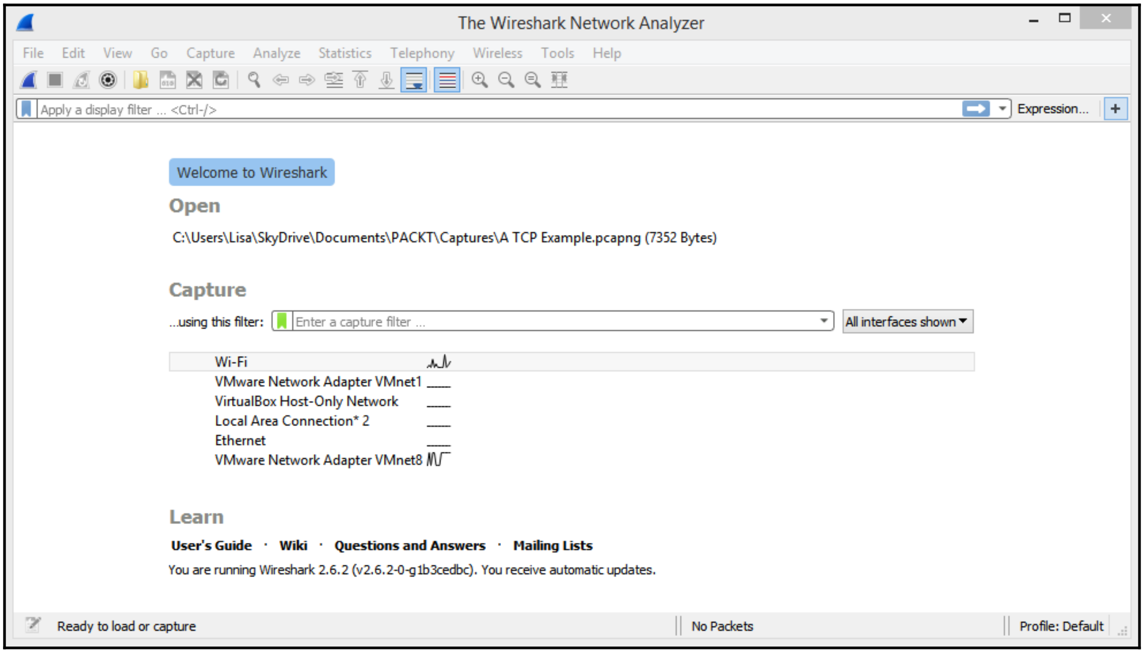
- Wi-Fi provides networking on a LAN using the IEEE 802.11 standards. The 802.11 family has several standards; however, 802.11a, 802.11b/g/n and 802.11ac are currently the most widely used standards.
- Bluetooth provides networking on a PAN over short distances from fixed and mobile devices, which allows 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, as shown in the following screenshot, where we would go to the lower part of the screen to see what interfaces are active by viewing the sparklines. The following screenshot shows the main Wireshark interface:



Welcome to Wireshark

Open

C:\Users\Lisa\SkyDrive\Documents\PACKT\Captures\A TCP Example.pcapng (7352 Bytes)

Capture

...using this filter:

Wi-Fi
VMware Network Adapter VMnet1
VirtualBox Host-Only Network
Local Area Connection* 2
Ethernet
VMware Network Adapter VMnet8

Learn

User's Guide · Wiki · Questions and Answers · Mailing Lists

You are running Wireshark 2.6.2 (v2.6.2-0-g1b3cedbc). You receive automatic updates.

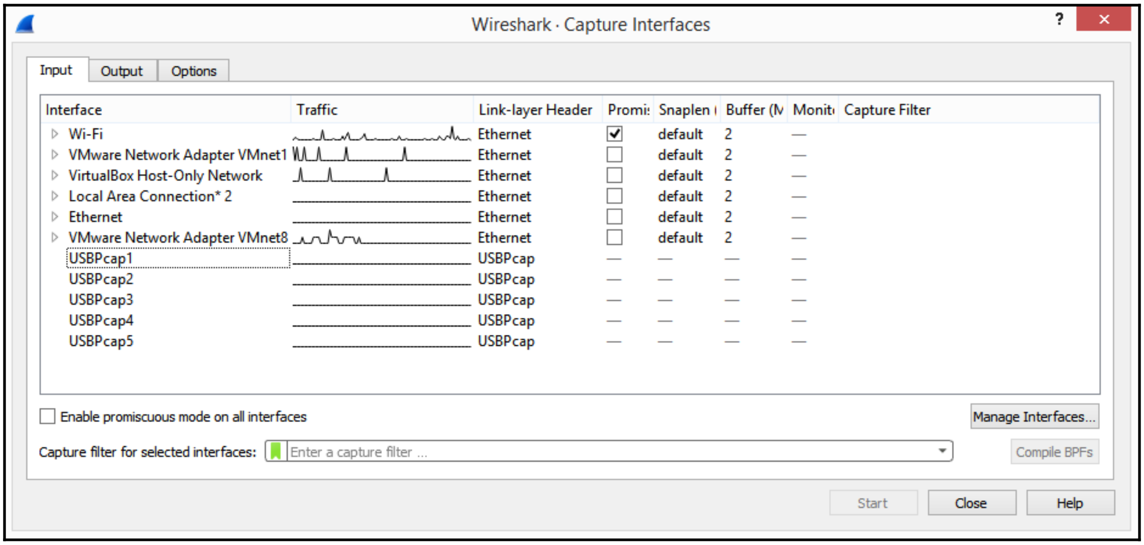
Ready to load or capture | No Packets | Profile: Default

Wireshark interface

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. However, there are a few other capture options that allow you to do advanced configuration before capturing:

1. Go to the **Capture** drop-down menu and then into **Options**.
2. Select the **Capture Interfaces** icon.

Whatever you choose will open the advanced options dialog box. Across the top, you will see three tabs, **Input**, **Output**, and **Options**, as shown in the following screenshot:



Interface	Traffic	Link-layer Header	Promi:	Snaplen (	Buffer (M	Monit	Capture Filter
Wi-Fi		Ethernet	☑	default	2	—	
VMware Network Adapter VMnet1		Ethernet	☐	default	2	—	
VirtualBox Host-Only Network		Ethernet	☐	default	2	—	
Local Area Connection* 2		Ethernet	☐	default	2	—	
Ethernet		Ethernet	☐	default	2	—	
VMware Network Adapter VMnet8		Ethernet	☐	default	2	—	
USBPcap1		USBPcap	—	—	—	—	
USBPcap2		USBPcap	—	—	—	—	
USBPcap3		USBPcap	—	—	—	—	
USBPcap4		USBPcap	—	—	—	—	
USBPcap5		USBPcap	—	—	—	—	

☐ Enable promiscuous mode on all interfaces

Capture filter for selected interfaces: Enter a capture filter ...

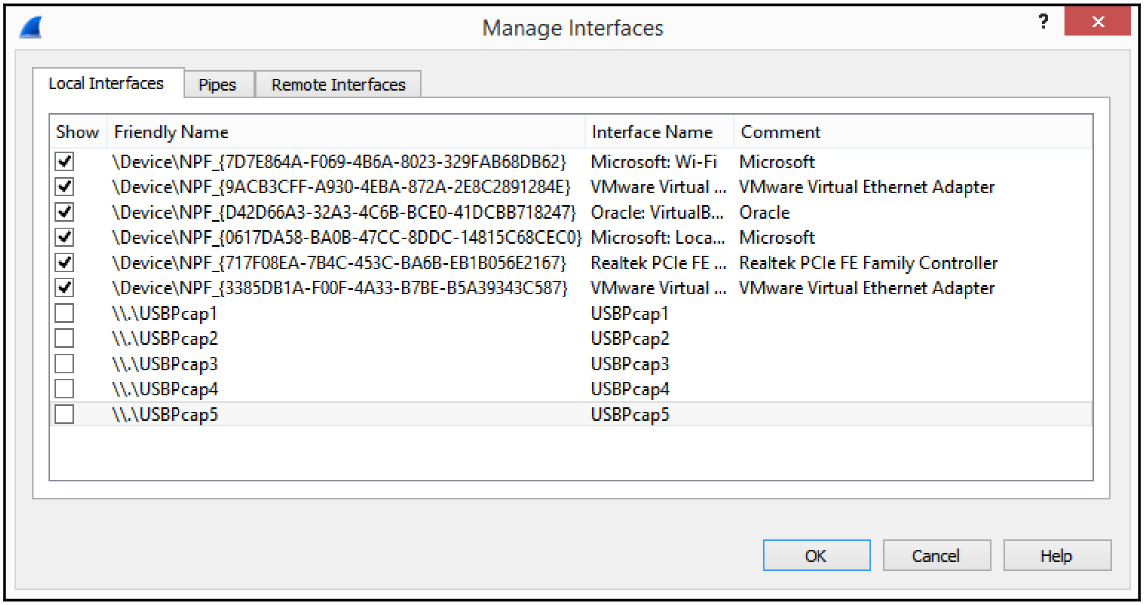
Capture options

To that end, let's start with how to set up a capture by selecting an input interface.

Providing input

In the **Capture Interfaces** dialog box, the **Input** tab will show a list of available interfaces on your device. Across the top, you will see various column headers that 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. In that case, you can then select the interface you want to be in promiscuous mode by checking the box to the right of the interface. At the bottom, you can create a capture filter for the selected interface.

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



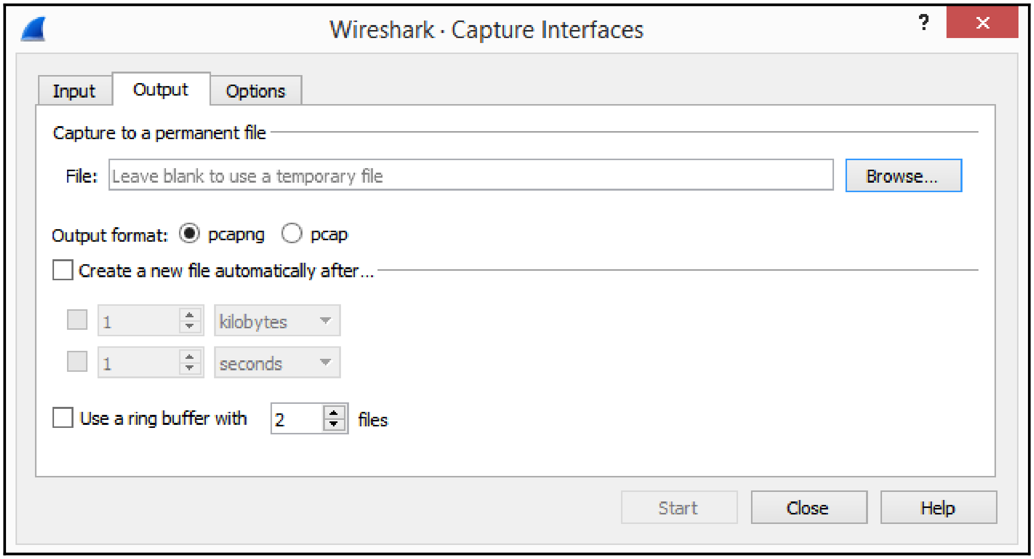
Show	Friendly Name	Interface Name	Comment
☑	\Device\NPF_{7D7E864A-F069-4B6A-8023-329FAB68DB62}	Microsoft: Wi-Fi	Microsoft
☑	\Device\NPF_{9ACB3CFF-A930-4EBA-872A-2E8C2891284E}	VMware Virtual ...	VMware Virtual Ethernet Adapter
☑	\Device\NPF_{D42D66A3-32A3-4C6B-BCE0-41DCBB718247}	Oracle: VirtualB...	Oracle
☑	\Device\NPF_{0617DA58-BA0B-47CC-8DDC-14815C68CEC0}	Microsoft: Loca...	Microsoft
☑	\Device\NPF_{717F08EA-7B4C-453C-BA6B-EB1B056E2167}	Realtek PCIe FE ...	Realtek PCIe FE Family Controller
☑	\Device\NPF_{3385DB1A-F00F-4A33-B7BE-B5A39343C587}	VMware Virtual ...	VMware Virtual Ethernet Adapter
□	\\.\USBPcap1	USBPcap1	
□	\\.\USBPcap2	USBPcap2	
□	\\.\USBPcap3	USBPcap3	
□	\\.\USBPcap4	USBPcap4	
□	\\.\USBPcap5	USBPcap5	

Capture options—Input tab

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 choices. The first choice is to **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. The **Output format** defaults at saving the file as **pcapng** (PCAP next generation); however, you can force Wireshark to save the file as a **pcap**. Most of the time, **pcapng** is the best choice as it allows you to add comments. The following screenshot shows the **Output** tab of the **Capture Interfaces** dialog:



Capture to a permanent file

File: Leave blank to use a temporary file

Output format: ● pcapng ○ pcap

□ Create a new file automatically after...

□ 1 kilobytes

□ 1 seconds

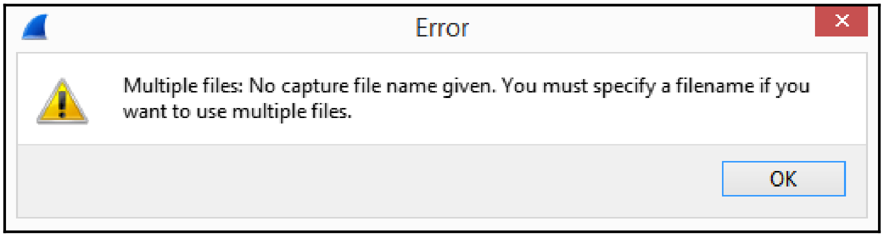
□ Use a ring buffer with 2 files

Capture options—manage interfaces

The next selection allows you to use a ring buffer to monitor traffic. 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 Wireshark holds the capture in a temporary file until you stop the capture and save to a permanent file.

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 file if you want to use multiple files; otherwise, you will throw an error, as shown here:



Multiple files: No capture file name given. You must specify a filename if you want to use multiple files.

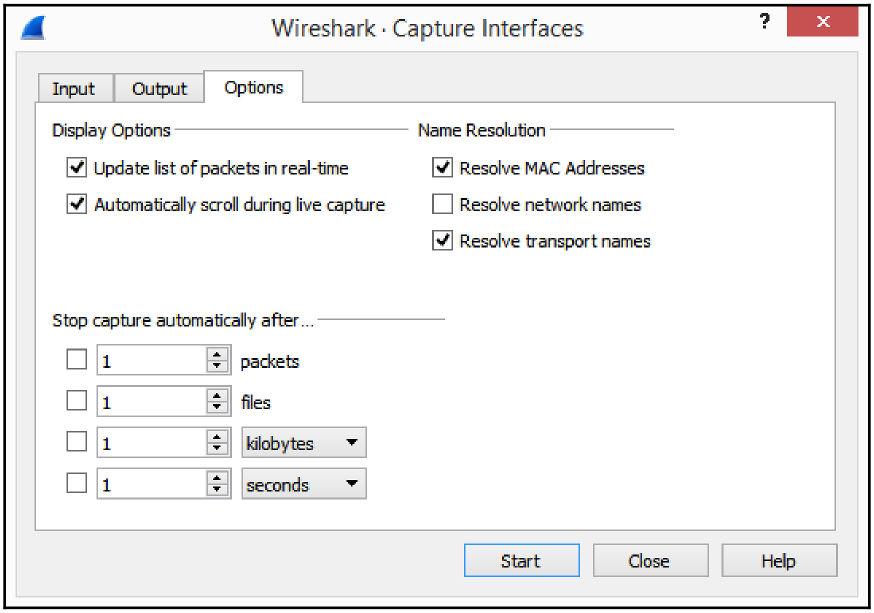
Error message in the capture options

At the bottom, select **Use a ring buffer** 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

The **Display Options** are generally set to **Update list of packets in real-time** and **Automatically scroll during live capture,** as shown in the following screenshot:



Display Options

☑ Update list of packets in real-time

☑ Automatically scroll during live capture

Name Resolution

☑ Resolve MAC Addresses

□ Resolve network names

☑ Resolve transport names

Stop capture automatically after ...

□ 1 packets

□ 1 files

□ 1 kilobytes

□ 1 seconds

Options tab

The **Name Resolution** choices include the following:

- **Resolve MAC Addresses**
- **Resolve network names**
- **Resolve transport names**

It's okay to resolve MAC addresses and transport names, as these are changed into 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 contact the DNS server multiple times while resolving the IP addresses and 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**
- **Size of file**
- **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 you understand the network architecture and the 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 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:

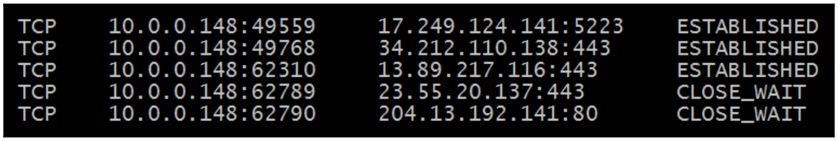
- **Broadcast**: Packets are sent from one to everyone on a network—that is, ARP broadcast.
- **Multicast**: Packets are sent from one to many—that is, routing protocol **EIGRP** (short for **Enhanced Interior Gateway Routing Protocol**) multicasts.
- **Unicast**: This sends packets from one to one—that is, from your computer to a web server.

In a normal conversation with another host, once you have a connection, the operating system creates 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

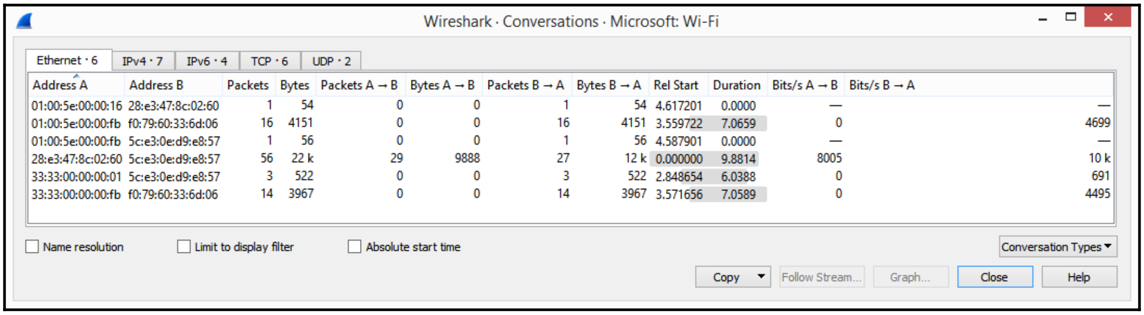
Whenever you are actively connecting with other hosts on the network, the OS keeps track of all the connections. To see all of your active connections on a Windows machine, open a command line and run `netstat` with the parameters `-an` , as shown in the following screenshot:



```
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
```

The netstat command showing 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. In the following screenshot, you can see the five tabs, **Ethernet**, **IPv4**, **IPv6**, **TCP**, and **UDP**:



Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
01:00:5e:00:00:16	28:e3:47:8c:02:60	1	54	0	0	1	54	4.617201	0.0000	—	—
01:00:5e:00:00:fb	f0:79:60:33:6d:06	16	4151	0	0	16	4151	3.559722	7.0659	0	4699
01:00:5e:00:00:fb	5c:e3:0e:d9:e8:57	1	56	0	0	1	56	4.587901	0.0000	—	—
28:e3:47:8c:02:60	5c:e3:0e:d9:e8:57	56	22 k	29	9888	27	12 k	0.000000	9.8814	8005	10 k
33:33:00:00:00:01	5c:e3:0e:d9:e8:57	3	522	0	0	3	522	2.848654	6.0388	0	691
33:33:00:00:00:fb	f0:79:60:33:6d:06	14	3967	0	0	14	3967	3.571656	7.0589	0	4495

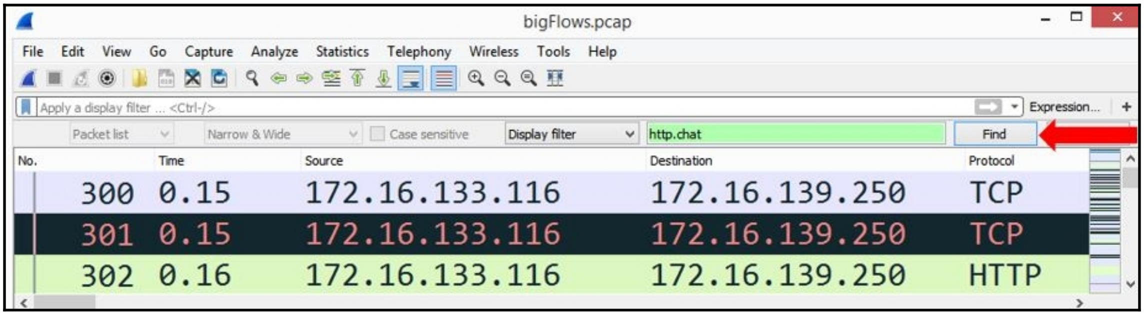
☐ Name resolution ☐ Limit to display filter ☐ Absolute start time

Conversations

Each tab provides details of the type of conversation you selected. For example, the **Ethernet** tab shows Ethernet conversations listing the MAC addresses of the endpoints. Each row represents one conversation. Wireshark has advanced options within this window. We can select any of the conversations by right-clicking and selecting any of 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; to run the filter you must press *Enter*.
- **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**:


bigFlows.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
Find

Packet list | Narrow & Wide | □ Case sensitive | Display filter | http.chat

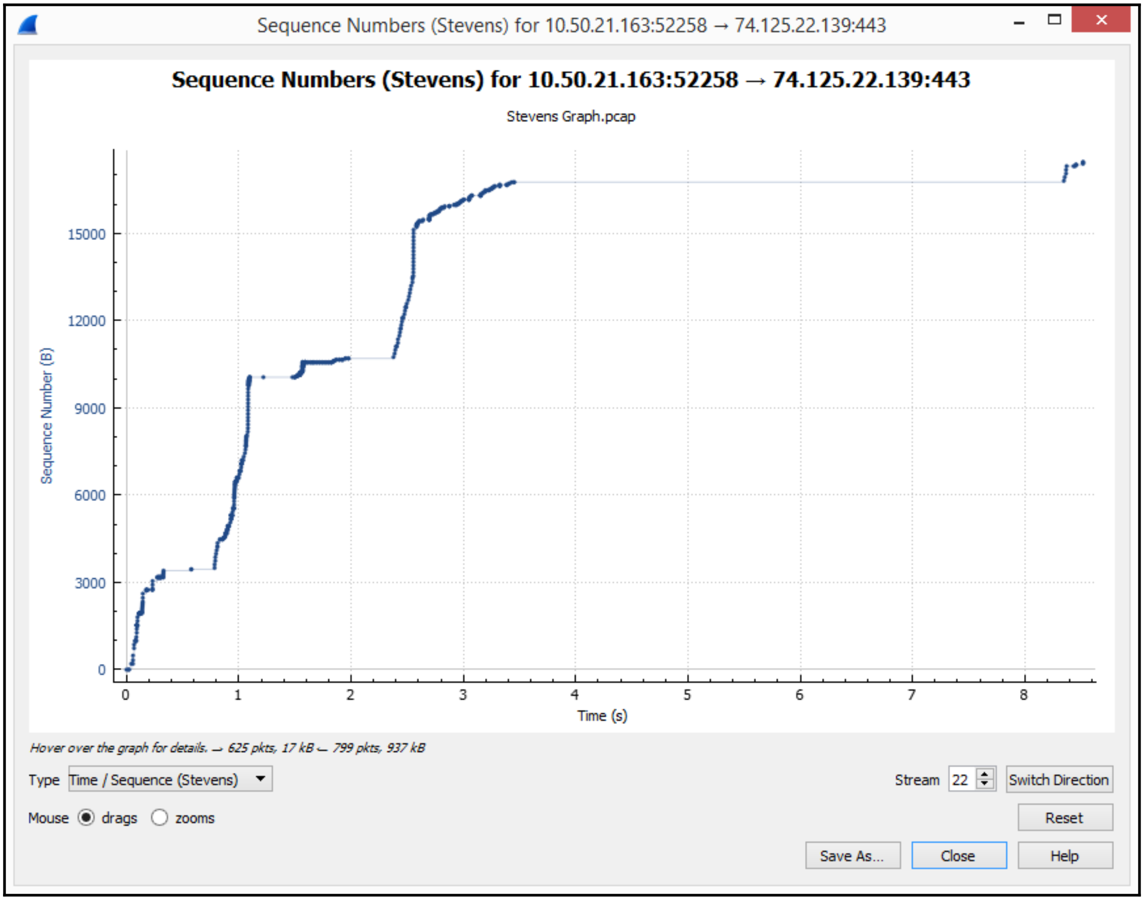
No.	Time	Source	Destination	Protocol
300	0.15	172.16.133.116	172.16.139.250	TCP
301	0.15	172.16.133.116	172.16.139.250	TCP
302	0.16	172.16.133.116	172.16.139.250	HTTP

The Find packets 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**, **A to Any**, among others.

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 CSV or **YAML** (short for **Yet Another Markup Language**) 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 conversation. You must first select either a TCP or UDP conversation.
- **Graph**: This will launch and display a TCP Stream graph on the selected TCP conversation, as shown here:


Sequence Numbers (Stevens) for 10.50.21.163:52258 → 74.125.22.139:443
Sequence Numbers (Stevens) for 10.50.21.163:52258 → 74.125.22.139:443
Stevens Graph.pcap
Sequence Number (B)
0
3000
6000
9000
12000
15000
Time (s)
0
1
2
3
4
5
6
7
8
Save As...
Close
Help
Reset
Switch Direction

Hover over the graph for details. → 625 pkts, 17 kB ← 799 pkts, 937 kB

Type Time / Sequence (Stevens)

Stream 22

Mouse ● drags ○ zooms

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 more effectively troubleshoot a network, it is important to have a packet capture to 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 the 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

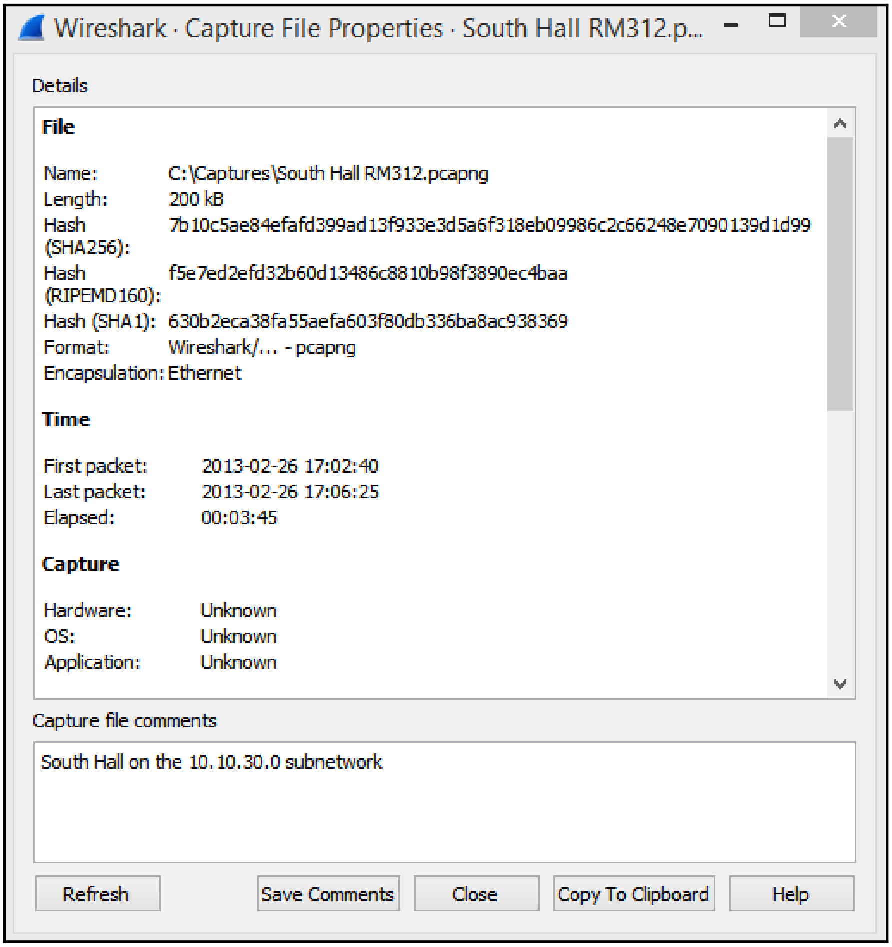
To plan the baseline, create a network map and list all of the subnetworks including all VoIP VLANs. You should have a strategy on how you are going to go about the process, including the time of day, whether you are going to capture wireless or wired traffic, and details about the location and what 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 1,000-2,000 packets so that you have a consistent capture size for the baselines. The process of capturing should be documented; for example, where and what time of day it was and what equipment you used during capture.

The key is to be as consistent as possible with your captures, so 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:



File

Name: C:\Captures\South Hall RM312.pcapng
Length: 200 kB
Hash (SHA256): 7b10c5ae84efafd399ad13f933e3d5a6f318eb09986c2c66248e7090139d1d99
Hash (RIPEMD160): f5e7ed2efd32b60d13486c8810b98f3890ec4baa
Hash (SHA1): 630b2eca38fa55aefa603f80db336ba8ac938369
Format: Wireshark/... - pcapng
Encapsulation: Ethernet

Time

First packet: 2013-02-26 17:02:40
Last packet: 2013-02-26 17:06:25
Elapsed: 00:03:45

Capture

Hardware: Unknown
OS: Unknown
Application: Unknown

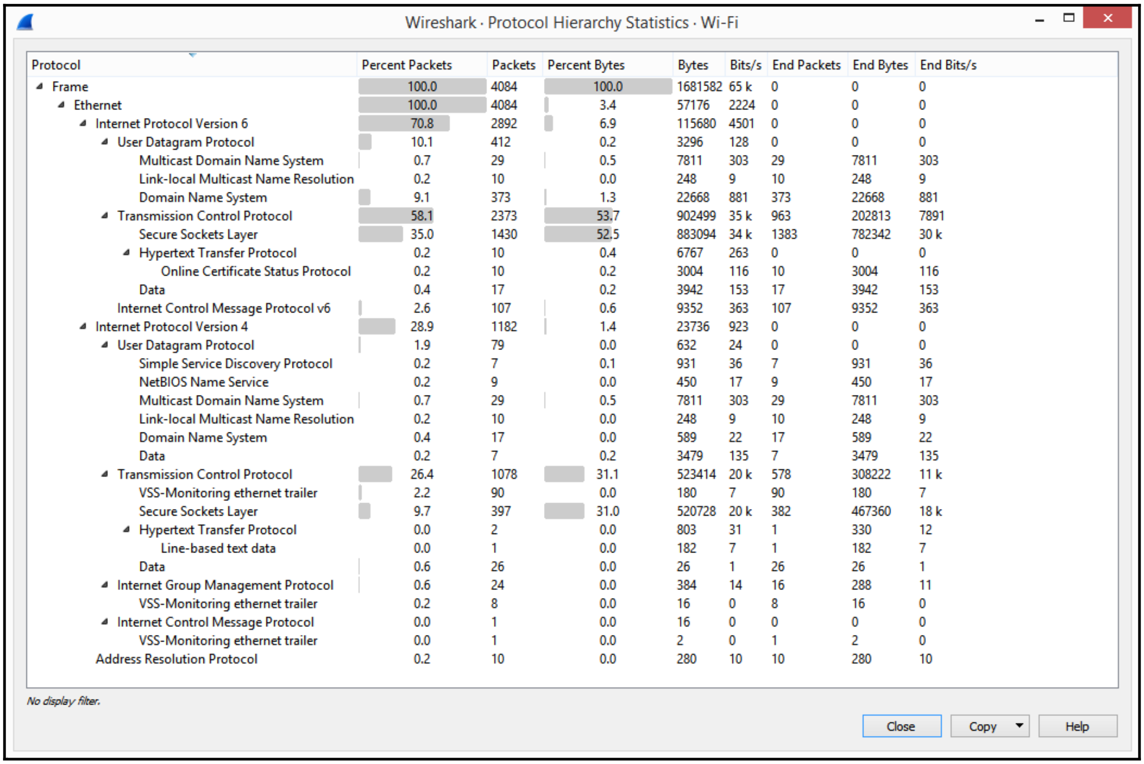
South Hall on the 10.10.30.0 subnetwork

The Capture File Properties window

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

Analyzing the captured traffic

You'll want to review the packet capture to see whether anything stands out as unusual or suspicious, including what protocols are being used and what ports are in use. Within Wireshark, there are various statistics you can run, such as going to **Statistics** and then **Protocol Hierarchy** to spot-check what protocols appear on the subnetwork, as shown here:



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
◢ Frame	100.0	4084	100.0	1681582	65 k	0	0	0
◢ Ethernet	100.0	4084	3.4	57176	2224	0	0	0
◢ Internet Protocol Version 6	70.8	2892	6.9	115680	4501	0	0	0
◢ User Datagram Protocol	10.1	412	0.2	3296	128	0	0	0
Multicast Domain Name System	0.7	29	0.5	7811	303	29	7811	303
Link-local Multicast Name Resolution	0.2	10	0.0	248	9	10	248	9
Domain Name System	9.1	373	1.3	22668	881	373	22668	881
◢ Transmission Control Protocol	58.1	2373	53.7	902499	35 k	963	202813	7891
Secure Sockets Layer	35.0	1430	52.5	883094	34 k	1383	782342	30 k
◢ Hypertext Transfer Protocol	0.2	10	0.4	6767	263	0	0	0
Online Certificate Status Protocol	0.2	10	0.2	3004	116	10	3004	116
Data	0.4	17	0.2	3942	153	17	3942	153
Internet Control Message Protocol v6	2.6	107	0.6	9352	363	107	9352	363
◢ Internet Protocol Version 4	28.9	1182	1.4	23736	923	0	0	0
◢ User Datagram Protocol	1.9	79	0.0	632	24	0	0	0
Simple Service Discovery Protocol	0.2	7	0.1	931	36	7	931	36
NetBIOS Name Service	0.2	9	0.0	450	17	9	450	17
Multicast Domain Name System	0.7	29	0.5	7811	303	29	7811	303
Link-local Multicast Name Resolution	0.2	10	0.0	248	9	10	248	9
Domain Name System	0.4	17	0.0	589	22	17	589	22
Data	0.2	7	0.2	3479	135	7	3479	135
◢ Transmission Control Protocol	26.4	1078	31.1	523414	20 k	578	308222	11 k
VSS-Monitoring ethernet trailer	2.2	90	0.0	180	7	90	180	7
Secure Sockets Layer	9.7	397	31.0	520728	20 k	382	467360	18 k
◢ Hypertext Transfer Protocol	0.0	2	0.0	803	31	1	330	12
Line-based text data	0.0	1	0.0	182	7	1	182	7
Data	0.6	26	0.0	26	1	26	26	1
◢ Internet Group Management Protocol	0.6	24	0.0	384	14	16	288	11
VSS-Monitoring ethernet trailer	0.2	8	0.0	16	0	8	16	0
◢ Internet Control Message Protocol	0.0	1	0.0	16	0	0	0	0
VSS-Monitoring ethernet trailer	0.0	1	0.0	2	0	1	2	0
Address Resolution Protocol	0.2	10	0.0	280	10	10	280	10

No display filter.

The Protocol Hierarchy Statistics window

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 move to the final phase where we save the captures for later comparison.

Saving the baselines

Once you have completed the capture, 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 format for saving the information can be a sharable spreadsheet, so the whole team can update and record their findings. Suggested guidelines for documentation include the following:

- List where the capture was taken, include the name of the building and/or the 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 location of the baseline.

Although much of the information in the preceding list can be recorded within the capture as well in the form of comments, it's best to document the information 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 the *aviation building - room 78 - subnetwork 192.168.10.112*, you can save the file as `AV-78-10.112.pcap`.

Save the information on a sharable spreadsheet, so the whole team can update and record the 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 options that include display options, using multiple files, and name resolution.

We then moved into a discussion on the different types of traffic you will see when tapping into a switched network. We then took a look at conversations and endpoints within the **Statistics** menu so you could understand the difference between the two. We also looked at the many options within the **Conversations** window. We summarized the importance of baselining the network and suggested some steps on how to conduct the baseline.

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, then check your answers with those found in the *Assessment*:

1. A _____ is a private network in a localized area that an organization or individual owns, controls, and manages.
    1. LAN
    2. WAN
    3. CAN
    4. 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.
    1. vendor.txt
    2. services.txt
    3. manuf.txt
    4. network.txt

3. In the capture options, the ___ tab allows you to specify where and how you want to save your file.
    1. Input
    2. Output
    3. Options
    4. Baseline
4. In fiber optic cable, ___ carries a single light beam that can carry a signal for many miles and must use a laser.
    1. Mutimode
    2. Coaxial
    3. STP
    4. Single mode
5. In the output tab of the capture options, the Output format defaults at saving the file as ___; however, you can force Wireshark to save the file as .pcap.
    1. .pcapng
    2. dmp.gz
    3. cap.gz
    4. .txt

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 the color choices.

So that you get a better understanding of how to document capture information in Wireshark, we'll go through tips on how to add comments to a single packet or to the entire capture. Finally, we will look at how to build and modify a complex filter expression so that you can feel more comfortable with display filters during packet analysis. We'll finish by learning about how to create a filter button on the toolbar as a shortcut for commonly used filters in Wireshark.

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

- Personalizing the layout and general appearance
- Creating a tailored configuration profile
- Adjusting columns, font, and color
- Adding comments
- Modifying complex expressions

Personalizing the layout and general appearance

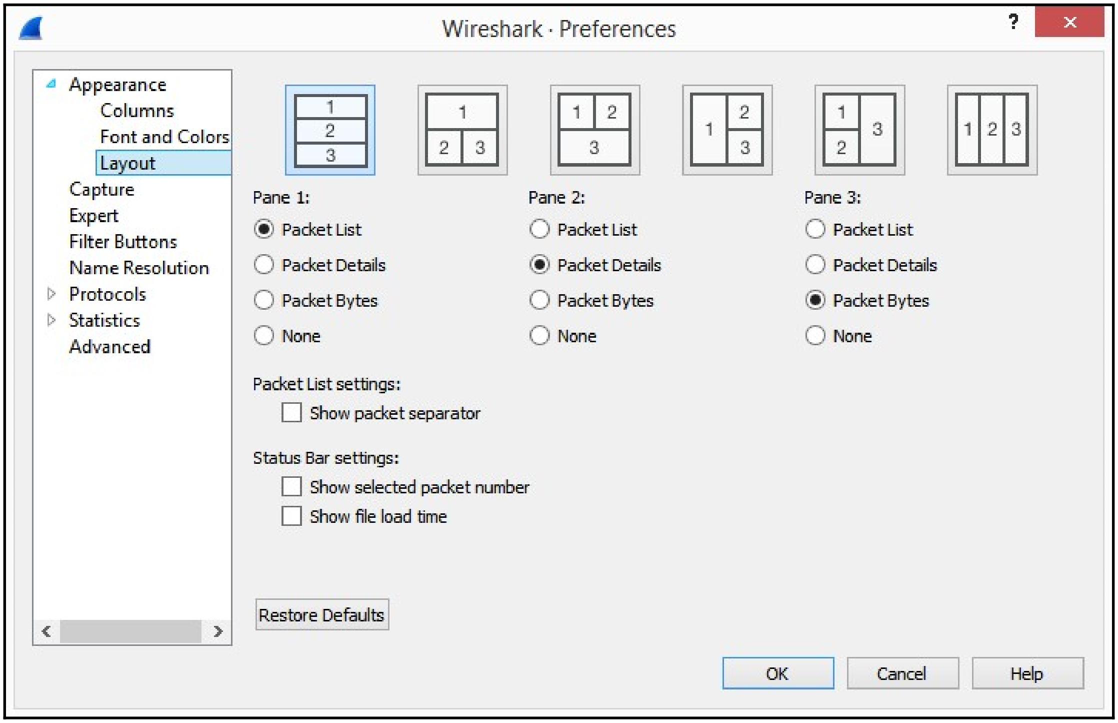
Although Wireshark is functional in the default mode, it's easy to modify the appearance and layout to optimize your workflow. In addition to personalizing the layout and general appearance, you can change the language, as well as customize the number of icons, recent filters, and folders, and what you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

Changing the layout

When working with Wireshark, it's easy to modify the standard layout of the three stacked panels, which are as follows:

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

In addition to the default setting, where the three panes are stacked one above the other, you can also rearrange them in one of five other layouts, as shown in the following screenshot:



Pane 1:
● Packet List
○ Packet Details
○ Packet Bytes
○ None

Pane 2:
○ Packet List
● Packet Details
○ Packet Bytes
○ None

Pane 3:
○ Packet List
○ Packet Details
● Packet Bytes
○ None

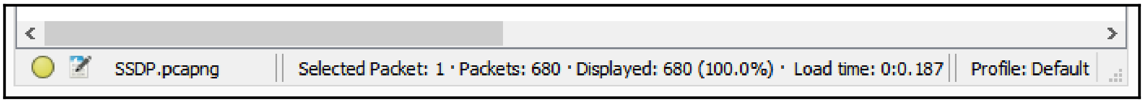
Packet List settings:
□ Show packet separator

Status Bar settings:
□ Show selected packet number
□ Show file load time

Wireshark Preferences dialog box—Layout

Once the dialog box is open, you have choices where you can make and modify your selections, as follows:

- **Packet List settings**: The **Show packet separator** option will insert a fine white line in between each frame in the packet list.
- **Status Bar settings**: The status bar is found at the bottom of the Wireshark interface, and can include **Show selected packet number** and **Show file load time**. If the latter is checked, then you will see the load time in the lower right-hand corner of the interface, as shown in the following screenshot:


SSDP.pcapng | Selected Packet: 1 · Packets: 680 · Displayed: 680 (100.0%) · Load time: 0:0.187 | Profile: Default

Status bar

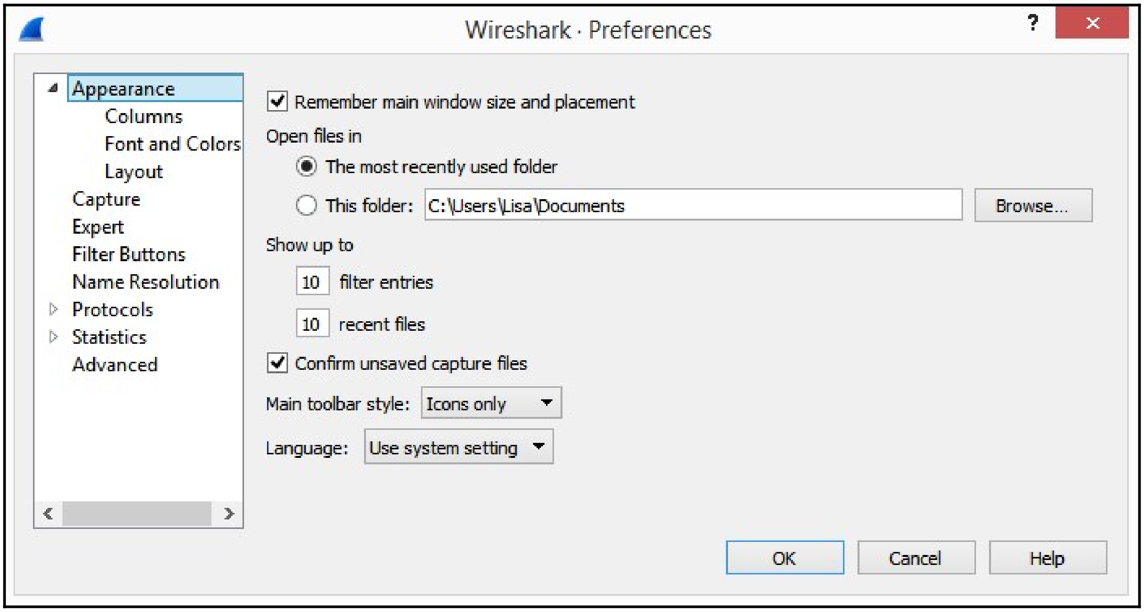
Next, let's evaluate the best way to customize your appearance.

Altering the appearance

In Wireshark, you can customize the general appearance to perform the following:

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

To make changes to the appearance, go to **Edit** and then **Preferences**, which will bring up the dialog box shown in the following screenshot:



☑ Remember main window size and placement

Open files in

● The most recently used folder

○ This folder: C:\Users\Lisa\Documents

Show up to

10 filter entries

10 recent files

☑ Confirm unsaved capture files

Main toolbar style: Icons only

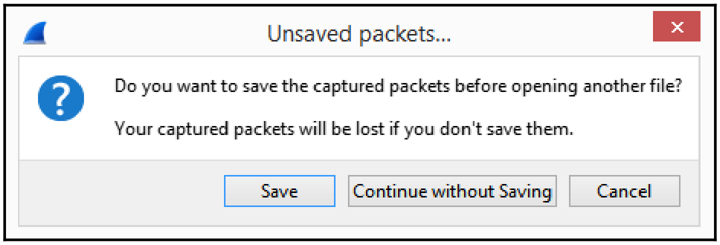
Language: Use system setting

Wireshark Preferences dialog box—Appearance

In order to see the choices, select the **Appearance** menu choice. Once the dialog box is open, you have choices where you can make and modify your selections, as follows:

- **Remember main window size and placement**: If selected, this will retain the window size and placement after you shut down Wireshark. For example, if the main window was the one-quarter size and positioned in the top left-hand quarter of your screen before shutting down, then when you re-open the window, it will appear in the same location and size.

- **Open files in**: Wireshark allows you to point to a location to open files. The default is **The most recently used folder**; however, if you have a standard preference for packet captures, then you might want to choose **This folder**, and then select **Browse...** to select the appropriate folder.
- **Show up to**: When working with files in Wireshark, this is where you would indicate how many recent files to keep visible when you go to **File**, and then **Open Recent**. In addition, you can also select how many filter entries to display. In the screenshot captioned as *Wireshark Preferences dialog box—Appearance*, I have selected `10` **filter entries** and `10` **recent files** to display.
- **Confirm unsaved capture files**: If this is checked, then before closing the application or opening another file, Wireshark will ask you if you want to save the captured packets, as shown here:



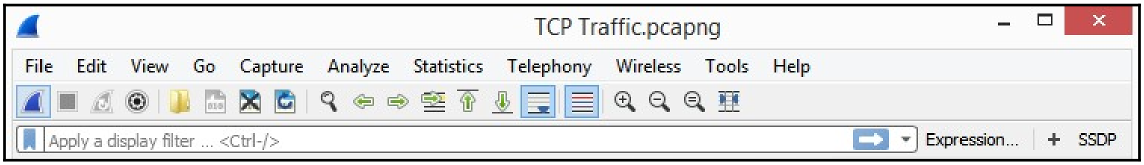
Unsaved packets...

Do you want to save the captured packets before opening another file?

Your captured packets will be lost if you don't save them.

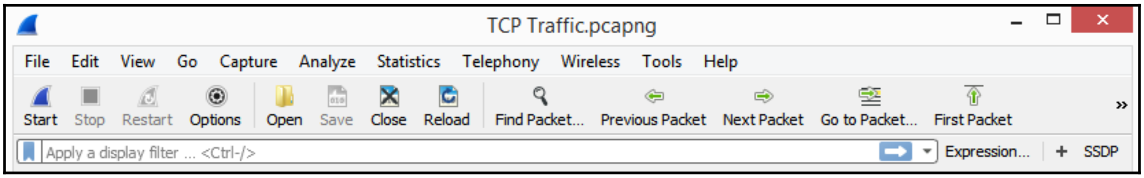
Prompt before closing

- **Main toolbar style**: The main toolbar is across the top, underneath the menu choices. You can alter the appearance to display as **Icons only**, as shown in the following screenshot:



Main toolbar—Icons only

You can also alter the appearance to display **Icons only**, **Text only**, or even **Icons and text**, as shown here:


TCP Traffic.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Start Stop Restart Options Open Save Close Reload Find Packet... Previous Packet Next Packet Go to Packet... First Packet
Apply a display filter ... <Ctrl-/>
Expression... + SSDP

Main toolbar—Icons and text

- **Language**: When installing Wireshark, the wizard prompts you to select a language. In the **Preferences** panel, **Use system setting** is the default. 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**.

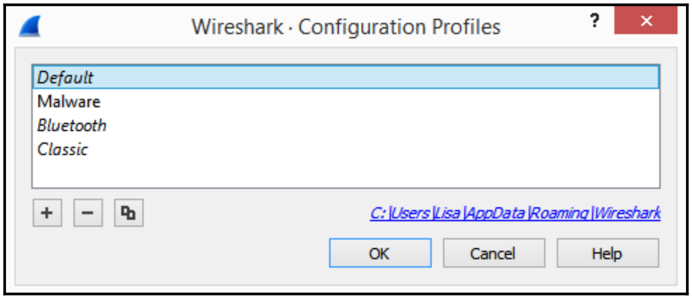
Now that we have set up our workspace, we can examine ways to generate a custom configuration profile, which is a unique set of preferences and configurations.

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 the graphic status bar, which is found at the end of the *Changing the layout* section.

In Wireshark, users can create their own custom configuration profiles, which can include personalized preferences, coloring rules, font styles, or even disabled protocols.

To create a custom profile, go to **Edit** and then **Configuration Profiles.** One the dialog box is open, you will see that Wireshark has three standard configurations: **Default**, **Bluetooth**, and **Classic**, as shown in italics in the following screenshot:


Wireshark · Configuration Profiles
?
×
Default
Malware
Bluetooth
Classic
+
−
C:\Users\Lisa\AppData\Roaming\Wireshark
OK
Cancel
Help

Configuration Profiles dialog box

It's easy to create a new profile: simply select the **+** sign and assign the profile a name. For example, I created a profile named `Malware`, as shown in the preceding screenshot. Once you add the profile, close the dialog box, and then you can modify the profile.

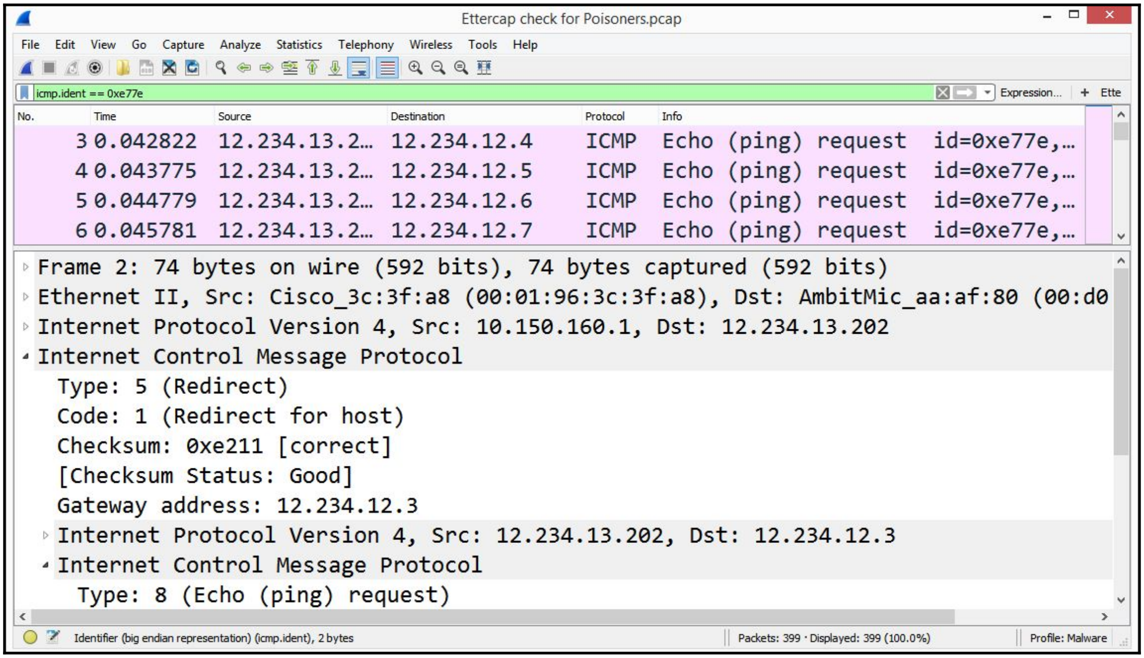
You can make several changes to suit your needs, such as the following:

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

Wireshark will save any changes in the custom profile.

In my `Malware` profile, I wanted to modify the settings so I could 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 Ettercap signature e77e, which translates to **ette** (short for **Ettercap**) in Leetspeak. You can check by visiting `https://www.dcode.fr/leet-speak-1337`. This signature identifies Ettercap as it searches for other poisoners on a LAN.

To customize my profile, I adjusted the columns, removed the lower panel **Packet Bytes**, and added an **Ette** button (discussed in the *Modifying complex expressions* section), as shown at the top right-hand side of the following screenshot:

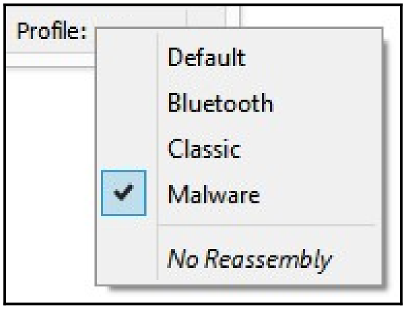

Ettercap check for Poisoners.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
+ Ette

```
icmp.ident == 0xe77e

No.  Time      Source          Destination    Protocol  Info
   3 0.042822  12.234.13.2…    12.234.12.4    ICMP      Echo (ping) request  id=0xe77e,…
   4 0.043775  12.234.13.2…    12.234.12.5    ICMP      Echo (ping) request  id=0xe77e,…
   5 0.044779  12.234.13.2…    12.234.12.6    ICMP      Echo (ping) request  id=0xe77e,…
   6 0.045781  12.234.13.2…    12.234.12.7    ICMP      Echo (ping) request  id=0xe77e,…

▹ Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▹ Ethernet II, Src: Cisco_3c:3f:a8 (00:01:96:3c:3f:a8), Dst: AmbitMic_aa:af:80 (00:d0
▹ Internet Protocol Version 4, Src: 10.150.160.1, Dst: 12.234.13.202
◂ Internet Control Message Protocol
    Type: 5 (Redirect)
    Code: 1 (Redirect for host)
    Checksum: 0xe211 [correct]
    [Checksum Status: Good]
    Gateway address: 12.234.12.3
  ▹ Internet Protocol Version 4, Src: 12.234.13.202, Dst: 12.234.12.3
  ◂ Internet Control Message Protocol
      Type: 8 (Echo (ping) request)

Identifier (big endian representation) (icmp.ident), 2 bytes    Packets: 399 · Displayed: 399 (100.0%)    Profile: Malware
```

Profile: Malware

Using my `Malware` profile I can easily check for Ettercap poisoners by hitting my button, which will apply and run an `icmp.ident == 0xe77e` display filter and show all the packets with that signature, if any are present in the capture.

If you want to return back to the default profile, then right-click on the lower right-hand corner and select the profile you want to use, as shown in the following screenshot:



Modify Profile

This next section illustrates how you can add or remove columns, and also 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 values used in the interface. However, you can adjust font styles and size to personalize the look and feel of the interface. In addition, you can also modify the colors Wireshark uses for the various packet identifiers and display filters.

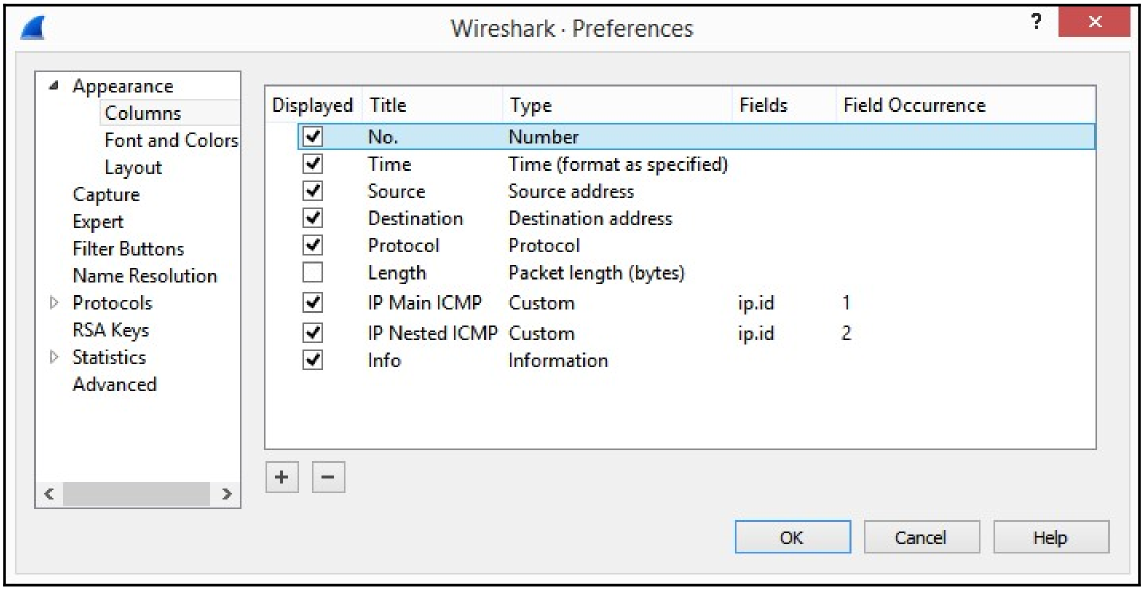
Once you are on the interface, you will see the columns and column headers that are along the top of the interface. While you are working in the interface, you might not ever manipulate the columns. However, you can add, delete, align, and customize the column at any time.

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. This section explains some ways to improve the way you visualize columns.

To customize your columns, go to **Edit**, then **Preferences**, and then **Columns**, as shown in the following screenshot:



Displayed	Title	Type	Fields	Field Occurrence
☑	No.	Number		
☑	Time	Time (format as specified)		
☑	Source	Source address		
☑	Destination	Destination address		
☑	Protocol	Protocol		
□	Length	Packet length (bytes)		
☑	IP Main ICMP	Custom	ip.id	1
☑	IP Nested ICMP	Custom	ip.id	2
☑	Info	Information		

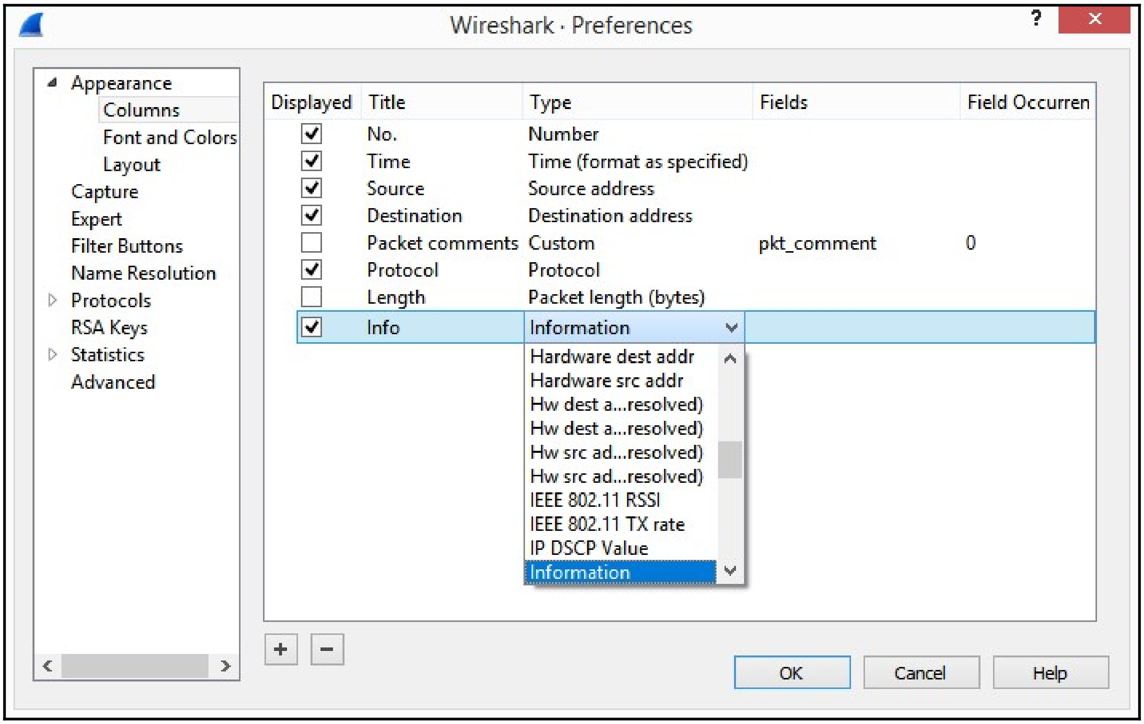
Wireshark Preferences dialog box—Columns

Once selected, you will see a list of columns. Some are present by default, and some are columns you may have added. You can select the checkbox to make the column visible or deselect the checkbox to hide the column. In addition, you can add or remove columns.

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

- **Displayed**: When checked, this 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 column header.

- **Type**: This lists the type of value that is in the column. Within the drop-down menu, there are many pre-loaded column types, as shown in the following screenshot, where I have dropped down the type selection for the **Info** column header:



Displayed	Title	Type	Fields	Field Occurren
☑	No.	Number		
☑	Time	Time (format as specified)		
☑	Source	Source address		
☑	Destination	Destination address		
☐	Packet comments	Custom	pkt_comment	0
☑	Protocol	Protocol		
☐	Length	Packet length (bytes)		
☑	Info	Information		

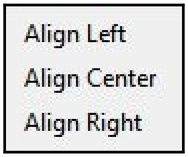
Columns—type selections

- **Fields**: This identifies the field where the column value originated from. In the preceding screenshot, there is a column header called **Packet Comments** (`pkt_comment`). That is because that column header was generated by right-clicking on a packet comment and selecting **Apply a Column**. Wireshark identifies that column header as the `pkt_comment` field value.

- **Field Occurrence**: This only used on a custom column definition. In the *Wireshark Preferences dialog box—Columns* screenshot, you can see there are values of **1** and **2** in the **Field Occurrence** column. When selected, the column headers will appear in this order:
    - **IP Main ICMP** will appear first.
    - **IP Nested ICMP** will appear second.

To add a column, select the plus sign. Identify the name 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 columns by selecting the column you don't want 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:



Aligning columns

Most other column headers are fairly straightforward in how they are used. However, one you may not be familiar with or use very often is **Field Occurrence.** Therefore, let's walk through how and why you would use a **Field occurrence** column header.

Demonstrating how to use field occurrence

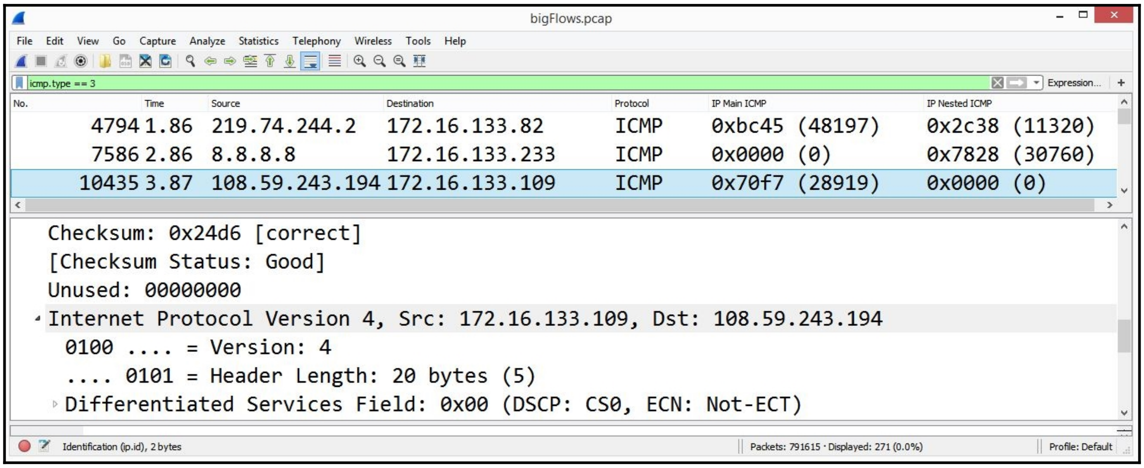
When an ICMP error message is sent, the ICMP packet also includes the IP header and the first 8 bytes (64 bits) of the original datagram that caused the error.

We can use a field occurrence so that we can see the ID field of the first IP header along with the ID field of the nested IP header, so we can see whether the two are the same or different.

To compare the two, complete these steps to modify the column headers:

1. Go to any frame that has an ICMP packet that has an error. For example, use the `icmp.type == 3` display filter to see all ICMP destination unreachable packets.
2. Drop down the *main* IP header and select the **Identification** field; right-click and then select **Apply as Column**. This will add the **Identification** column header.
3. Next, go into **Column Preferences** and then modify the settings for the newly created column header.
    - **Displayed**: Checked
    - **Title**: **IP Main ICMP**
    - **Type**: **Custom** (unchanged)
    - **Fields**: `ip.id` (unchanged)
    - **Field Occurrence**: **1**
4. Drop down the *nested* IP header and select the **Identification** field; right-click and then select **Apply as Column**. This will add the **Identification** column header.
5. I selected the **ID** field in the nested ICMP packet, right-clicked, and then selected **Apply as Column**.
6. Then, go into **Column Preferences** and modify the settings for the newly created column header.
    - **Displayed**: Checked
    - **Title**: **IP Main ICMP**
    - **Type**: **Custom** (unchanged)
    - **Fields**: `ip.id` (unchanged)
    - **Field Occurrence**: **2**

The result is shown in the following screenshot, where, right after the **Protocol** column header, you will see **IP Main ICMP** followed by **IP Nested ICMP**:


bigFlows.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
+

icmp.type == 3

No.	Time	Source	Destination	Protocol	IP Main ICMP	IP Nested ICMP
4794	1.86	219.74.244.2	172.16.133.82	ICMP	0xbc45 (48197)	0x2c38 (11320)
7586	2.86	8.8.8.8	172.16.133.233	ICMP	0x0000 (0)	0x7828 (30760)
10435	3.87	108.59.243.194	172.16.133.109	ICMP	0x70f7 (28919)	0x0000 (0)

```
  Checksum: 0x24d6 [correct]
  [Checksum Status: Good]
  Unused: 00000000
▴ Internet Protocol Version 4, Src: 172.16.133.109, Dst: 108.59.243.194
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▹ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
```

Identification (ip.id), 2 bytes | Packets: 791615 · Displayed: 271 (0.0%) | Profile: Default

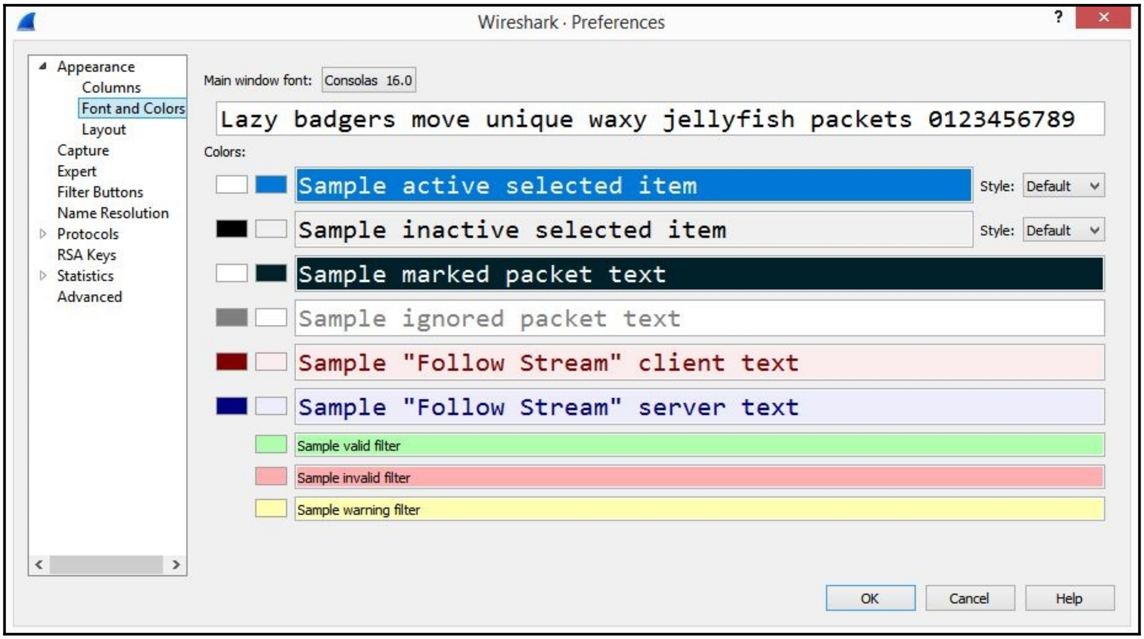
Field occurrence

Now, you can see how versatile Wireshark is 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 classic profile uses Consolas 10 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**, then to **Preferences**, and select **Font and Colors**, as shown in the screenshot here:



Main window font: Consolas 16.0

Lazy badgers move unique waxy jellyfish packets 0123456789

Sample active selected item — Style: Default

Sample inactive selected item — Style: Default

Sample marked packet text

Sample ignored packet text

Sample "Follow Stream" client text

Sample "Follow Stream" server text

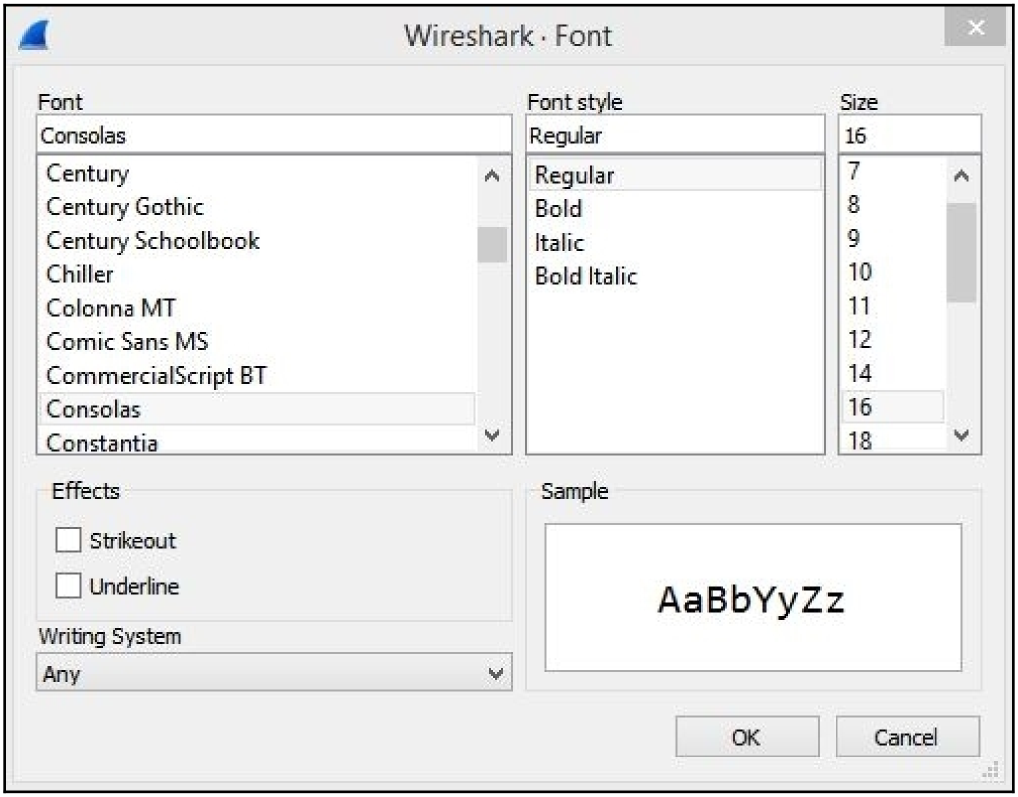
Sample valid filter

Sample invalid filter

Sample warning filter

Wireshark Preferences dialog box—Font and Colors

Along the top, if you select **Main window font** then it will display a dialog box that will allow you to make changes to the font, as shown here:



Wireshark · Font

Font: Consolas

Font style: Regular

Size: 16

Effects

☐ Strikeout

☐ Underline

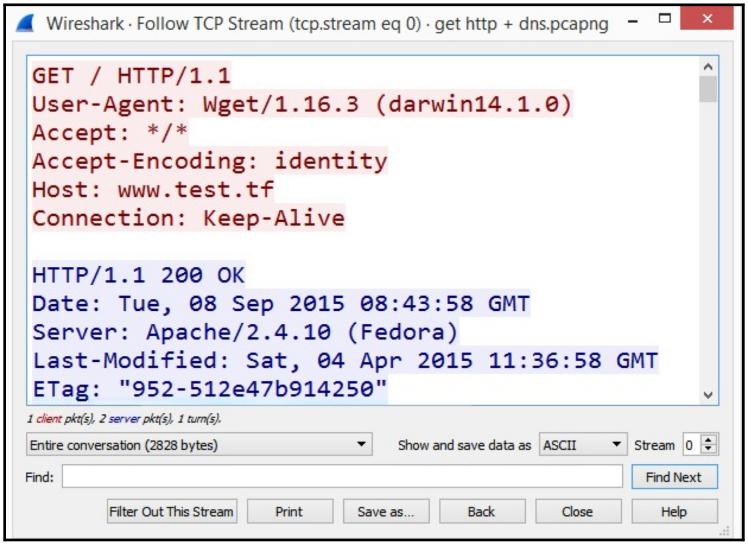
Writing System: Any

Sample

AaBbYyZz

Main window font

Below the **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 Stream**, as shown here:



```
GET / HTTP/1.1
User-Agent: Wget/1.16.3 (darwin14.1.0)
Accept: */*
Accept-Encoding: identity
Host: www.test.tf
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 08 Sep 2015 08:43:58 GMT
Server: Apache/2.4.10 (Fedora)
Last-Modified: Sat, 04 Apr 2015 11:36:58 GMT
ETag: "952-512e47b914250"
```

1 client pkt(s), 2 server pkt(s), 1 turn(s).

Wireshark · Follow TCP Stream (tcp.stream eq 0) · get http + dns.pcapng
Entire conversation (2828 bytes)
Show and save data as
ASCII
Stream
0
Find:
Find Next
Filter Out This Stream
Print
Save as...
Back
Close
Help

Follow the Stream

The colors help identify whether the client or the server is talking. The default value for the client text, when you follow the stream, is red, but this value can be changed.

The last three samples listed refer to the display filters. The syntax checker within Wireshark checks the filter once you enter something in **Display Filter**. By default, a valid filter turns green, an invalid filter turns red, and a warning filter turns yellow. As with all the other choices, you can change this as well.

In Wireshark, we can indicate information on a particular packet or capture 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 you can reference them at a later date. You might need 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, so that you or your coworkers can reference the comments at a later date. While commenting is optional, it is always good practice.

Let's start with adding 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.

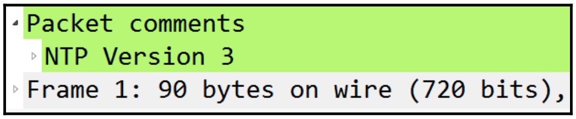
There are a few ways to add a comment to the file. I have listed them here:

- 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 the *Status bar* screenshot.
- Go to **Statistics** and then **Capture File Properties**. 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, go to **Edit** and then **Packet Comment**. A dialog box will be displayed, where you can enter your comment. Once entered, Wireshark will append a note with bright green coloring at the top of the frame, as shown in the following screenshot:



```
Packet comments
  NTP Version 3
Frame 1: 90 bytes on wire (720 bits),
```

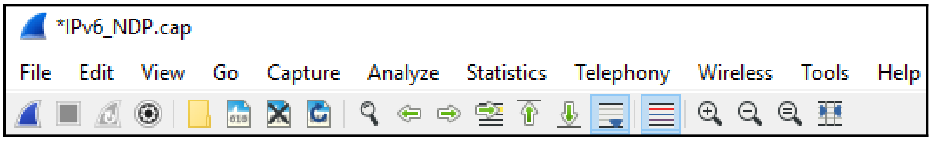
Packet comments in the Packet Details panel

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 captured the packet. However, in order for the comments to be of value, you must save them. Let's discuss how we save our comments.

Once you have created a comment, you will see an asterisk by the title across the top of the Wireshark interface, as shown here:


*IPv6_NDP.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Asterisk indicating that there are file comments

Once you have created a comment, you will see an asterisk by the title across the top of the Wireshark interface.

The asterisk will remain until you save the file. To preserve the comments, the file must be saved in the PCAP next generation (`.pcapng`) format.

If you or your team has taken the time to make a comment, then it's well worth your time to read them. You can view the comments in one of several ways:

- Use the `pkt_comment` display filter to see all packets that have comments.
- Open the **Expert System**.
- Go to the lower right-hand side to the drop-down menu named **Show** and select **Comment** from the list. This will display any comments that are in the capture.
- Go to **Statistics** then **Capture File Properties**, and view the comments in the lower pane.

Hopefully, by now, you can appreciate the many ways in which you can personalize your work area. Most of the time while working with packet captures, there is a need to refine our view by filtering traffic. We can use display filters along with complex expressions. In the next section, let's explore how we can do this, and if you find you're using the same expression repeatedly, then Wireshark can easily create a button that you can place on your toolbar for easy reference.

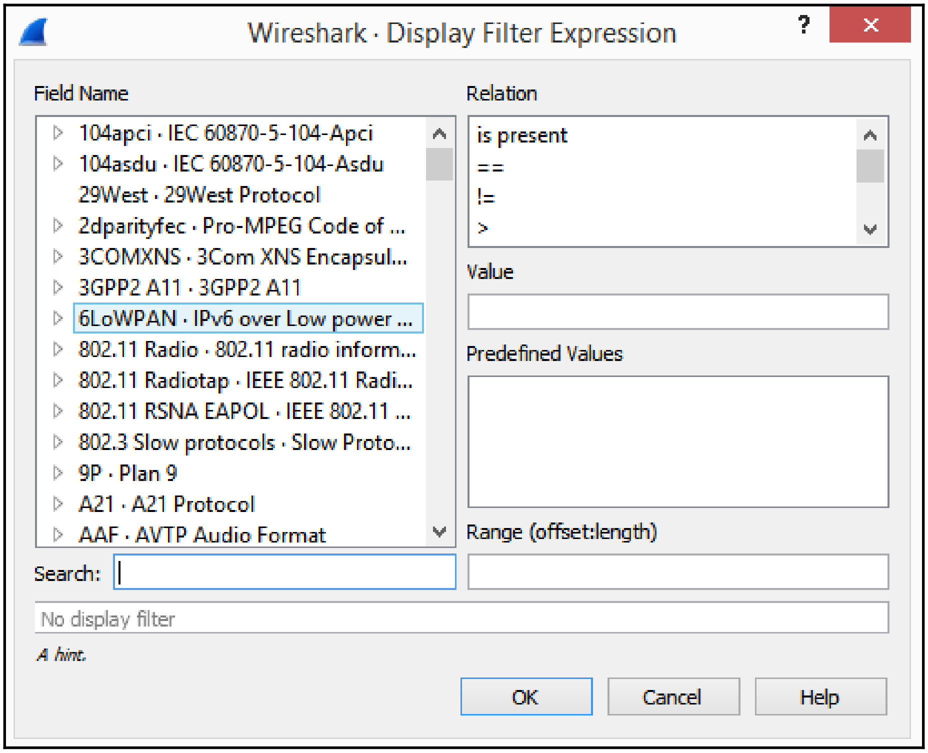
Modifying complex expressions

A display filter allows you to show only specific traffic. However, there are times you may need to use the expression builder so that you can create a more complex filter. For example, you may need to construct a filter that compares a specific protocol field against a value using logical operators. If you use the expression often, then you can create and attach a button to the toolbar. Let's first take a look at how to create an expression.

Creating expressions

In addition to display filters, which allow you to filter only the traffic you want to see, Wireshark has the ability to create a more complex expression. On the right-hand side of the display filter is the **Expression...** link. Click the link to launch the expression builder.

Once open, you will see a list of field names, as shown in the following screenshot:



Wireshark · Display Filter Expression

Field Name

- 104apci · IEC 60870-5-104-Apci
- 104asdu · IEC 60870-5-104-Asdu
- 29West · 29West Protocol
- 2dparityfec · Pro-MPEG Code of ...
- 3COMXNS · 3Com XNS Encapsul...
- 3GPP2 A11 · 3GPP2 A11
- 6LoWPAN · IPv6 over Low power ...
- 802.11 Radio · 802.11 radio inform...
- 802.11 Radiotap · IEEE 802.11 Radi...
- 802.11 RSNA EAPOL · IEEE 802.11 ...
- 802.3 Slow protocols · Slow Proto...
- 9P · Plan 9
- A21 · A21 Protocol
- AAF · AVTP Audio Format

Relation

- is present
- ==
- !=
- >

Value

Predefined Values

Range (offset:length)

Search:

No display filter

A hint.

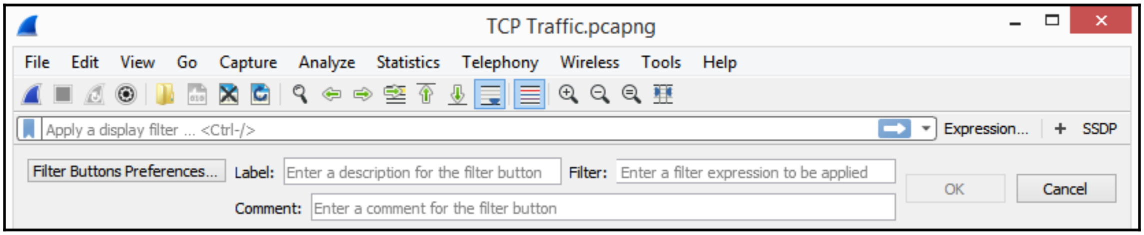
Display Filter Expression

From there, you can select a protocol and, in some cases, drill down for a specific field name. In some cases when in a protocol, you can select from several field values. On the right, you can use a logical operator and, also, a value. For example, if you want to filter all traffic where the TCP SYN flag is present, then the expression builder will create the `tcp.flags.syn` display filter. Once you are confident with your selections, close the expression builder and press *Enter*, and then Wireshark will run the filter.

If you have created a complex filter that you find you will need to run often, then you can create a handy toolbar button. This next section explains how you can easily craft a custom filter button.

Crafting buttons

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

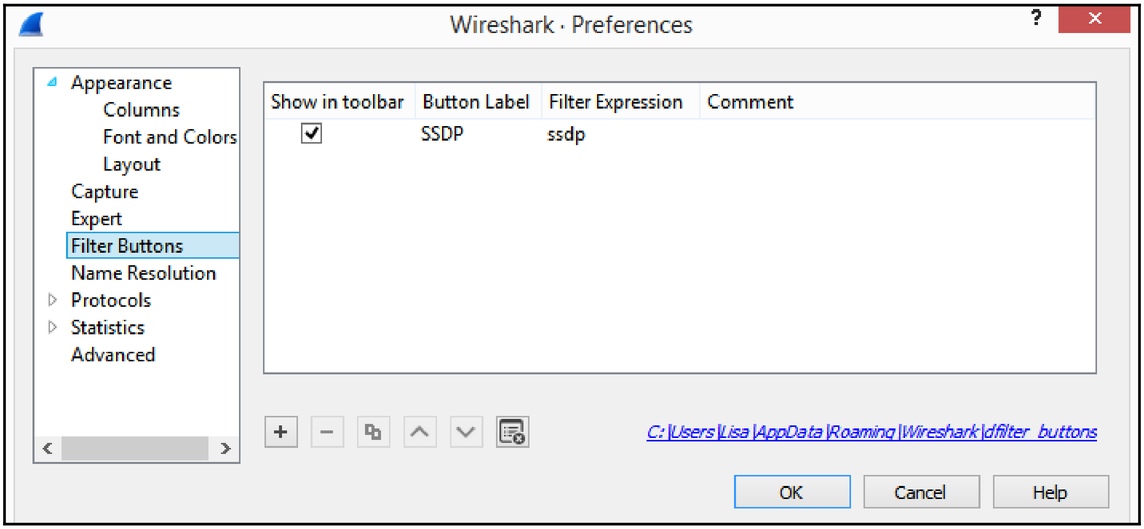

TCP Traffic.pcapng
File
Edit
View
Go
Capture
Analyze
Statistics
Telephony
Wireless
Tools
Help
Apply a display filter ... <Ctrl-/>
Expression...
+
SSDP
Filter Buttons Preferences...
Label:
Enter a description for the filter button
Filter:
Enter a filter expression to be applied
OK
Cancel
Comment:
Enter a comment for the filter button

Filter buttons

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:



Show in toolbar	Button Label	Filter Expression	Comment
☑	SSDP	ssdp	

C:\Users\Lisa\AppData\Roaming\Wireshark\dfilter_buttons

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 would like as a **Button Label**. This (**Button Label**) is not case sensitive.
3. Next, add **Filter Expression**, 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.

Summary

By now, you can see how easy it is to make minor changes in Wireshark to fit your workflow. In this chapter, we examined the many ways to customize the Wireshark interface. We covered how to modify choices such as recent filters and folders, along with personalizing the layout and general appearance. We learned about how easy it is to create personalized configuration profiles to include preferences, coloring rules, and font styles.

Furthermore, we discovered how to adjust columns and column headers, and how to add or remove columns. We learned about how 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 and the default colors for the client and server when you right-click on a packet and select **Follow the Stream**.

We illustrated the ability to add comments to a single packet or to the entire capture, as well as how to communicate issues to team members observed in either a single packet or the entire capture. We then learned about how to create a complex filter expression, and then create a filter button on the toolbar for commonly used filters in Wireshark to manage the workflow. Finally, we saw how to create a filter button to help manage our workflow.

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 display filters. We will then learn about using capture filters, including using default capture filters and how you can build your own. Finally, we will learn about how to use shortcuts to create filters and review some commonly used filters.

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*:

1. Normally, when you open Wireshark, the configuration profile will be the _____ profile.
   1. Marquee
   2. Bluetooth
   3. Classic
   4. Default
2. You can set Wireshark to open files from a specific location. Go to ____, then Preferences, and then select the file location under ____.
   1. Tools and Folder
   2. Edit and Appearance
   3. View and Appearance
   4. View and Folder
3. The default value for the client text for Follow the Stream is _____, but this value can be changed.
   1. Black
   2. Blue
   3. Red
   4. Cyan
4. When working with Wireshark, you can easily create and add a button to automatically run a custom filter when selected. To remove the button, go to _____, then _____, and select Filter Buttons, where you can remove any buttons that you no longer need.
   1. Edit and Preferences
   2. View and Appearance
   3. View and Buttons
   4. Tools and Buttons
5. When you want to add a comment to a packet, select the packet, go to ___, and then go to Packet Comments.
   1. Tools
   2. Edit
   3. View
   4. Analyze

7
Using Display and Capture Filters

Whether you have done an analysis in real time while capturing traffic, or you have analyzed a pre-captured file, you're generally faced with a huge amount of data. How do you make sense of all this data? Most likely, you will benefit from filtering the traffic to narrow the scope. To achieve this, we use filters, so that Wireshark only displays the traffic that you want to see.

This chapter reviews the many ways Wireshark can filter traffic. To help your learning of 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 filtering traffic, we'll review ways to create more complex filters by using the expression builder. We'll then go through capture filters and how they use syntax that is different than display filters. Finally, because filters are so handy, we'll cover some tricks, shortcuts, and common filters that will help you achieve a more effective analysis.

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

- Filtering network traffic
- Comprehending display filters
- Creating capture filters
- Understanding the expression builder
- Discovering shortcuts and handy filters

Filtering network traffic

While in the course of your daily routine, the network starts to experience a significant slowdown. You check your **Intrusion Detection System** (**IDS**) and anti-malware protection, and there is no evidence of intrusion. At that point, you grab a quick capture to determine the source of the slowdown. Wireshark, along with many other packet analysis tools, has the ability to take a large capture, filter on specific traffic, and refine your view to help with analysis. Wireshark has several options to filter traffic:

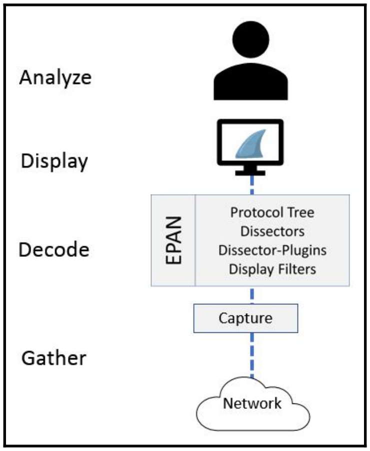
- **Display filters**: Used during an active capture or on a pre-captured packet
- **Capture filters**: Applied prior to capture to only display a certain type of traffic
- **Expressions**: Creates complex filters using logical operators

When filtering traffic, there is a difference between display filters and capture filters. In the next section, let's explore the difference.

Comparing display and capture filters

When working with packet captures, it appears as if capture and display filters are the same. However, although the two work in similar ways, capture and display filters each use their own syntax.

While using Wireshark, there are four main phases of packet analysis, as discussed in `Chapter 2`, *Using Wireshark NG*, which are **Gather**, **Decode**, **Analyze**, and **Display**, as shown in the following diagram:



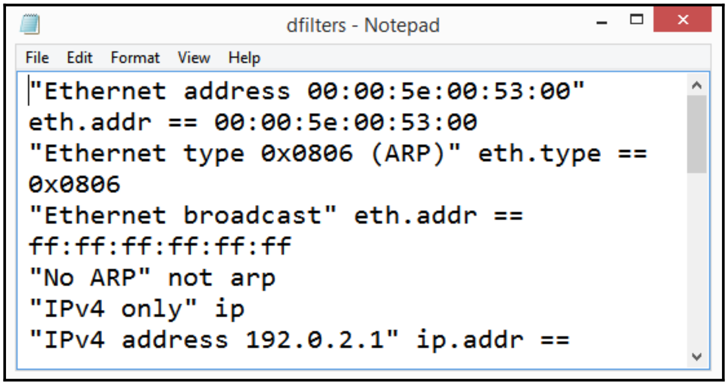
Phases of packet analysis

While gathering network traffic, the packets pass through the appropriate capture engine, such as NPcap or WinPcap. Capture filters use the Berkley packet filter syntax, and when used, Wireshark drops any packets that are not in the filter. You can read more about this in *The BSD Packet Filter: A New Architecture for User-level Packet Capture*, which is found at `https://www.tcpdump.org/papers/bpf-usenix93.pdf`.

Once the packets go through Wireshark's **Enhanced Packet ANalyzer** (**EPAN**), it then passes the dissected traffic through the GUI to be displayed and 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.

Within Wireshark, there are two text files that you can modify or copy to share with a coworker. The filter files include `dfilters.txt` and `cfilters.txt`. These hold a list of filters and are found in the Wireshark folder.

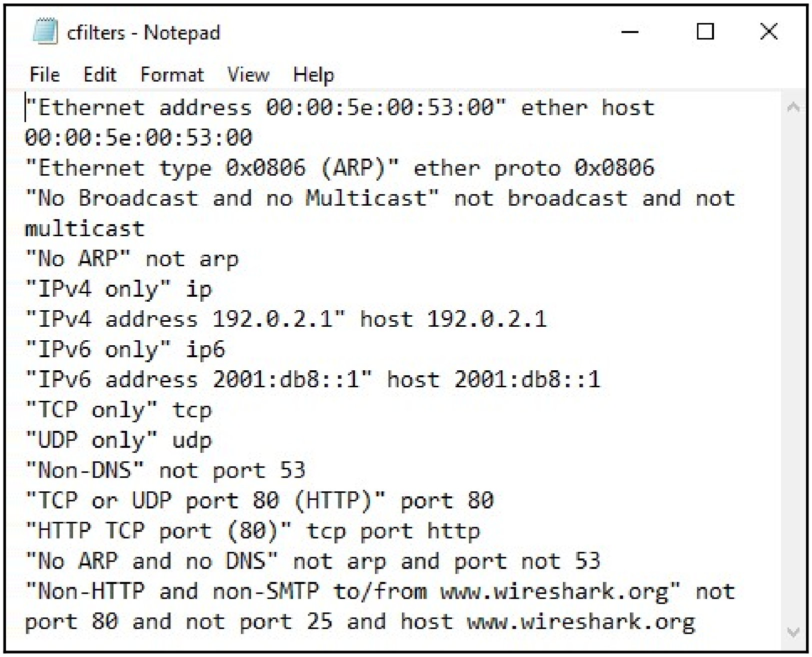
You'll find a list that holds the display filters in `dfilters.txt` and can open the file in Notepad, as shown here:



```
"Ethernet address 00:00:5e:00:53:00"
eth.addr == 00:00:5e:00:53:00
"Ethernet type 0x0806 (ARP)" eth.type ==
0x0806
"Ethernet broadcast" eth.addr ==
ff:ff:ff:ff:ff:ff
"No ARP" not arp
"IPv4 only" ip
"IPv4 address 192.0.2.1" ip.addr ==
```

dfilters.txt

The `cfilters.txt` file holds a list of the capture filters, as shown in the following screenshot:



```
"Ethernet address 00:00:5e:00:53:00" ether host
00:00:5e:00:53:00
"Ethernet type 0x0806 (ARP)" ether proto 0x0806
"No Broadcast and no Multicast" not broadcast and not
multicast
"No ARP" not arp
"IPv4 only" ip
"IPv4 address 192.0.2.1" host 192.0.2.1
"IPv6 only" ip6
"IPv6 address 2001:db8::1" host 2001:db8::1
"TCP only" tcp
"UDP only" udp
"Non-DNS" not port 53
"TCP or UDP port 80 (HTTP)" port 80
"HTTP TCP port (80)" tcp port http
"No ARP and no DNS" not arp and port not 53
"Non-HTTP and non-SMTP to/from www.wireshark.org" not
port 80 and not port 25 and host www.wireshark.org
```

cfilters.txt

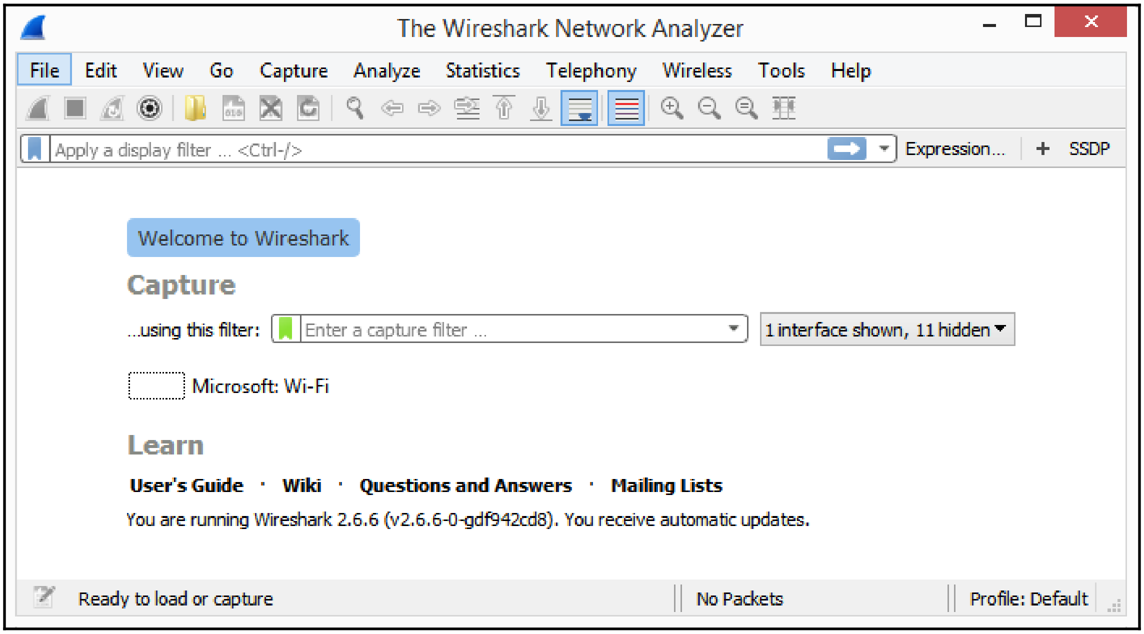
While it is possible to open and modify either file in Notepad, as shown, it's safer and easier to use the Wireshark interface to make changes to the filters.

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 2,000 to 3,000 packets and more, along with many different types of traffic.

When you launch Wireshark, you will see the startup screen, as shown in the following screenshot:


The Wireshark Network Analyzer
File
Edit
View
Go
Capture
Analyze
Statistics
Telephony
Wireless
Tools
Help
Apply a display filter ... <Ctrl-/>
Expression...
+ SSDP
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
1 interface shown, 11 hidden
Microsoft: Wi-Fi
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists
You are running Wireshark 2.6.6 (v2.6.6-0-gdf942cd8). You receive automatic updates.
Ready to load or capture
No Packets
Profile: Default

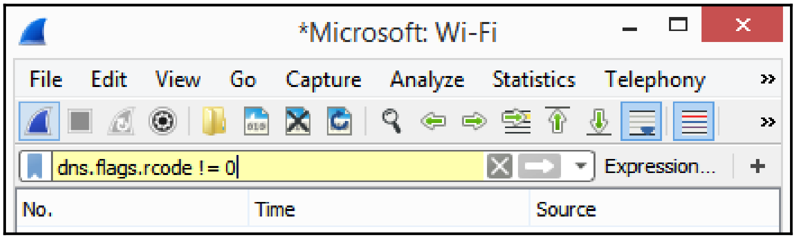
Wireshark startup screen

Across the top, below the icons, you will see the filter toolbar. Within the toolbar is the text `Apply a display filter`, where you can easily apply and edit display filters.

You can create a simple filter on any of the protocols Wireshark supports by using a single protocol or a logical operator. For example, if you want to see **Transmission Control Protocol** (**TCP**) or **Address Resolution Protocol** (**ARP**) traffic, then you would use the `tcp || arp` display filter. While you are building the filter, Wireshark will check the syntax to see whether the string is valid. 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, as shown in the following screenshot, which indicates that you may get unexpected results:



dns.flags.rcode != 0

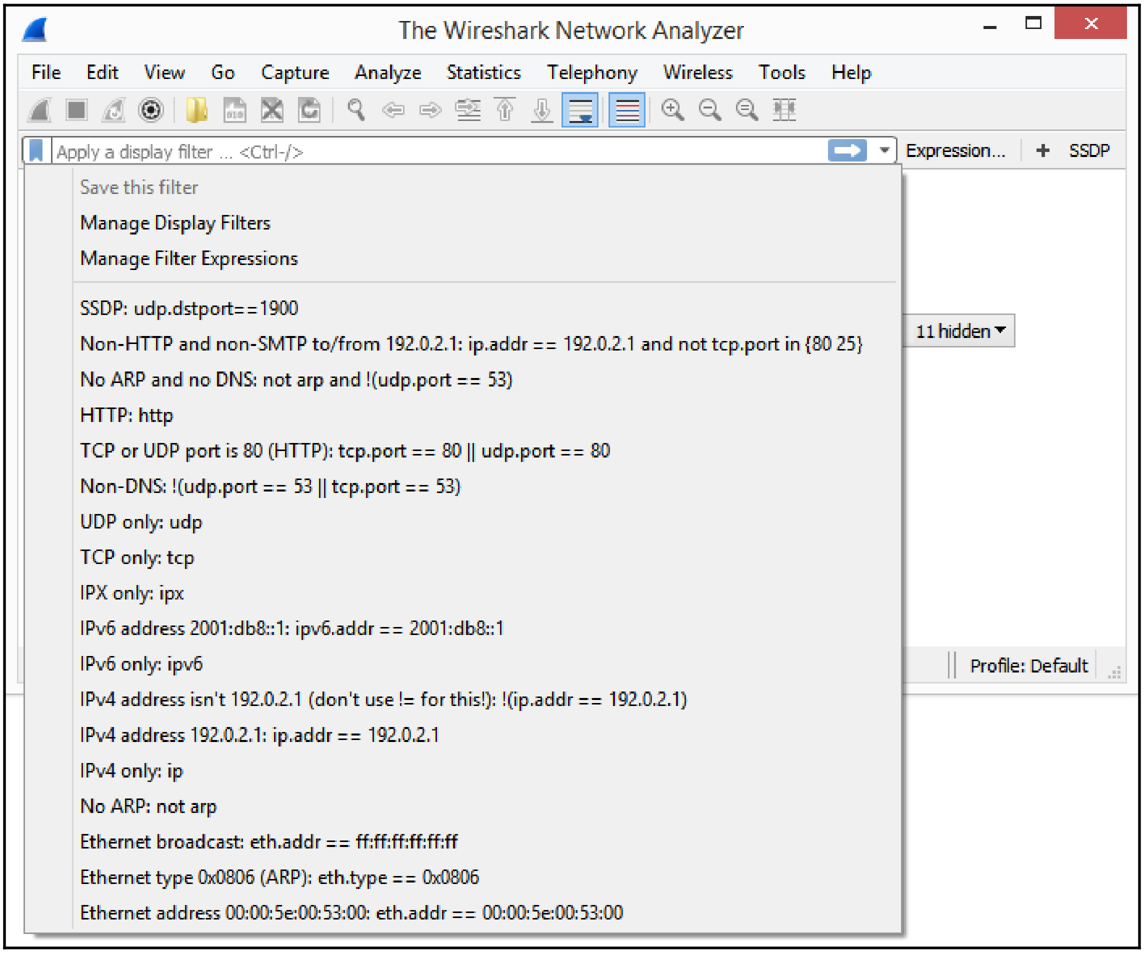
Syntax checker with a yellow background

Working with display filters can be confusing at times, so when you do get a filter that works and you would like to reuse it, you can save it to a bookmark, as discussed in the next section.

Using bookmarks

On the right-hand side of the display filter, is a blue toolbar icon called **bookmarks**, where Wireshark's built-in filters and any saved filters reside. Other choices when working with the bookmark include **Manage Display Filters** and **Manage Filter Expressions**.

Below the save and manage selections, you will see a list of filters. Even if you have never saved a filter, you will see the list, as Wireshark has several pre-loaded filters that you can use, as shown here:



Save this filter
Manage Display Filters
Manage Filter Expressions

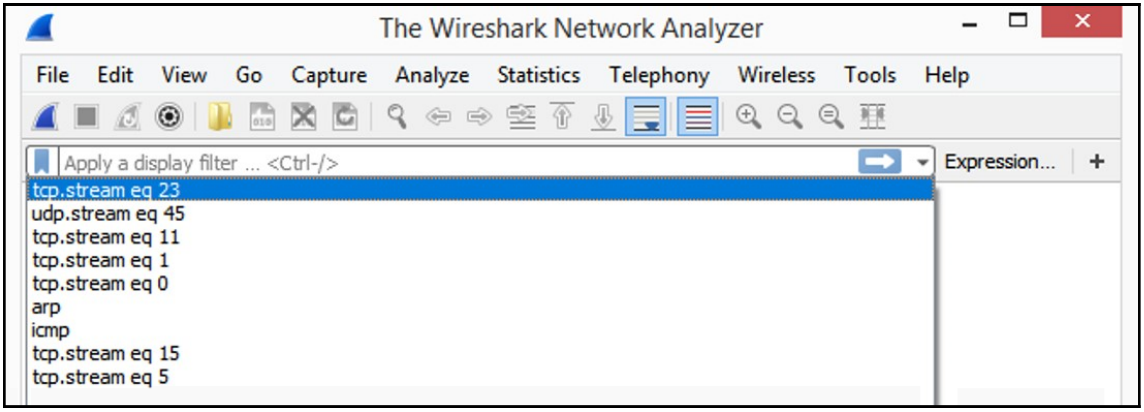
SSDP: udp.dstport==1900
Non-HTTP and non-SMTP to/from 192.0.2.1: ip.addr == 192.0.2.1 and not tcp.port in {80 25}
No ARP and no DNS: not arp and !(udp.port == 53)
HTTP: http
TCP or UDP port is 80 (HTTP): tcp.port == 80 || udp.port == 80
Non-DNS: !(udp.port == 53 || tcp.port == 53)
UDP only: udp
TCP only: tcp
IPX only: ipx
IPv6 address 2001:db8::1: ipv6.addr == 2001:db8::1
IPv6 only: ipv6
IPv4 address isn't 192.0.2.1 (don't use != for this!): !(ip.addr == 192.0.2.1)
IPv4 address 192.0.2.1: ip.addr == 192.0.2.1
IPv4 only: ip
No ARP: not arp
Ethernet broadcast: eth.addr == ff:ff:ff:ff:ff:ff
Ethernet type 0x0806 (ARP): eth.type == 0x0806
Ethernet address 00:00:5e:00:53:00: eth.addr == 00:00:5e:00:53:00

Display filter bookmark drop-down

After you create a filter, you can save the filter to the bookmark by dropping down the bookmark icon and selecting **Save this filter**.

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 an arrow that, when selected, is a drop-down menu where you can see previously used filters, also shown in the following screenshot:


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

tcp.stream eq 23
udp.stream eq 45
tcp.stream eq 11
tcp.stream eq 1
tcp.stream eq 0
arp
icmp
tcp.stream eq 15
tcp.stream eq 5

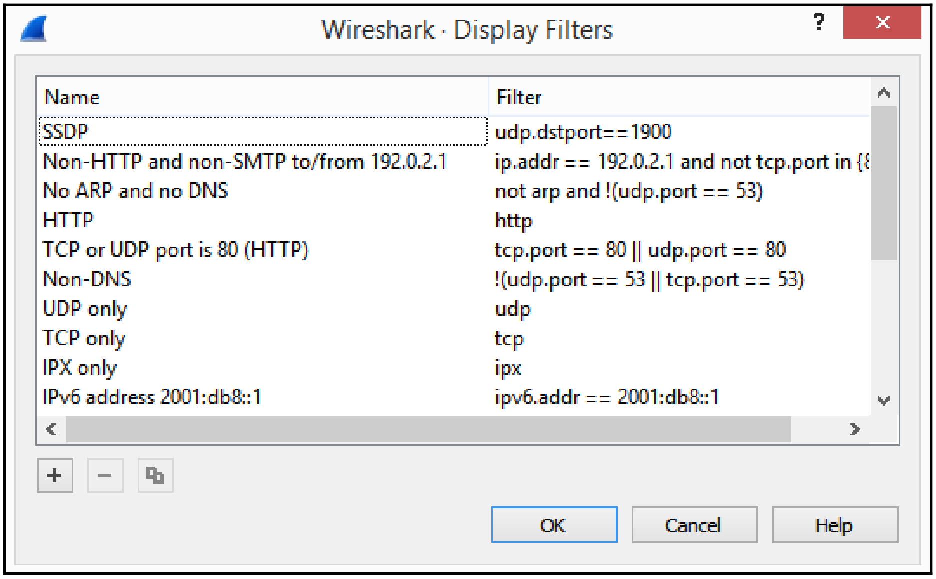
Previously used display filters

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 in the screenshot named *Syntax checker with a yellow background*.

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 blue bookmark on the left of the display filter, and then select **Manage Display Filters**, which will bring up the dialog box, as shown in the following screenshot:



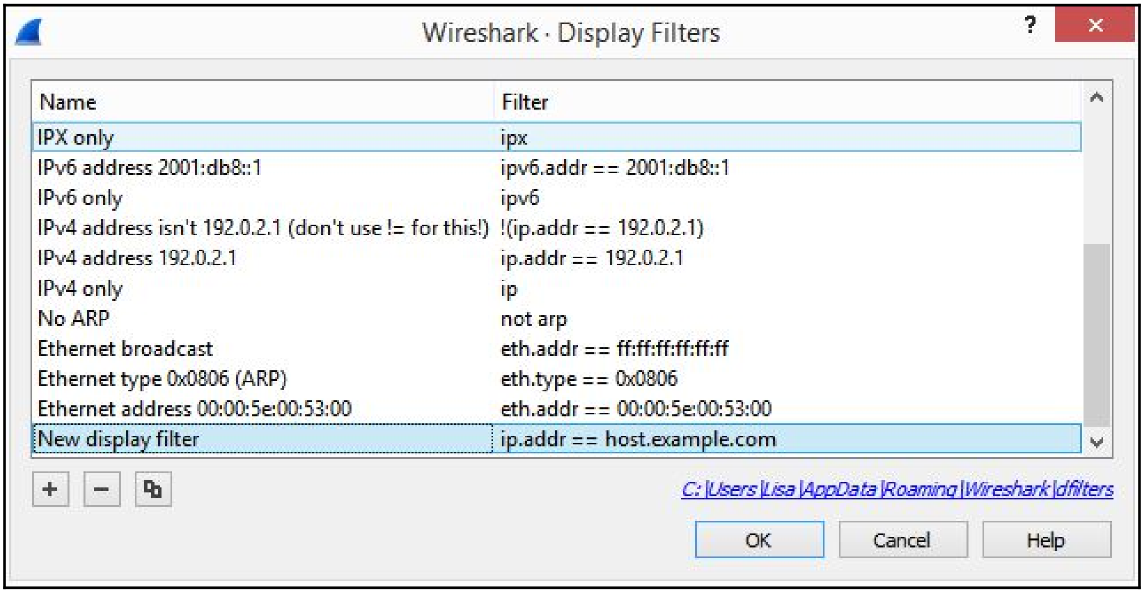
Name	Filter
SSDP	udp.dstport==1900
Non-HTTP and non-SMTP to/from 192.0.2.1	ip.addr == 192.0.2.1 and not tcp.port in {8
No ARP and no DNS	not arp and !(udp.port == 53)
HTTP	http
TCP or UDP port is 80 (HTTP)	tcp.port == 80 || udp.port == 80
Non-DNS	!(udp.port == 53 || tcp.port == 53)
UDP only	udp
TCP only	tcp
IPX only	ipx
IPv6 address 2001:db8::1	ipv6.addr == 2001:db8::1

Display Filters dialog box

Once there, you can select one of the three icons:

- A plus icon to add a new display filter
- A minus icon to delete a display filter
- A copy icon to copy a display filter

When you select the plus icon and add a display filter, Wireshark will create a space in which you can enter a display filter name on the left and the actual filter on the right, as shown here:


Wireshark · Display Filters
?
+
−
C:\Users\Lisa\AppData\Roaming\Wireshark\dfilters
OK
Cancel
Help

Name	Filter
IPX only	ipx
IPv6 address 2001:db8::1	ipv6.addr == 2001:db8::1
IPv6 only	ipv6
IPv4 address isn't 192.0.2.1 (don't use != for this!)	!(ip.addr == 192.0.2.1)
IPv4 address 192.0.2.1	ip.addr == 192.0.2.1
IPv4 only	ip
No ARP	not arp
Ethernet broadcast	eth.addr == ff:ff:ff:ff:ff:ff
Ethernet type 0x0806 (ARP)	eth.type == 0x0806
Ethernet address 00:00:5e:00:53:00	eth.addr == 00:00:5e:00:53:00
New display filter	ip.addr == host.example.com

Add a display filter

When you select **Copy**, this will copy and allow you to modify the filter without changing the original filter.

As we can see, display filters can be very helpful in providing a more targeted view of the capture. However, when capturing traffic for analysis, there may be times that you only want to capture a certain type of traffic. In that case, you would use a capture filter, which we'll discuss in this next section.

Creating capture filters

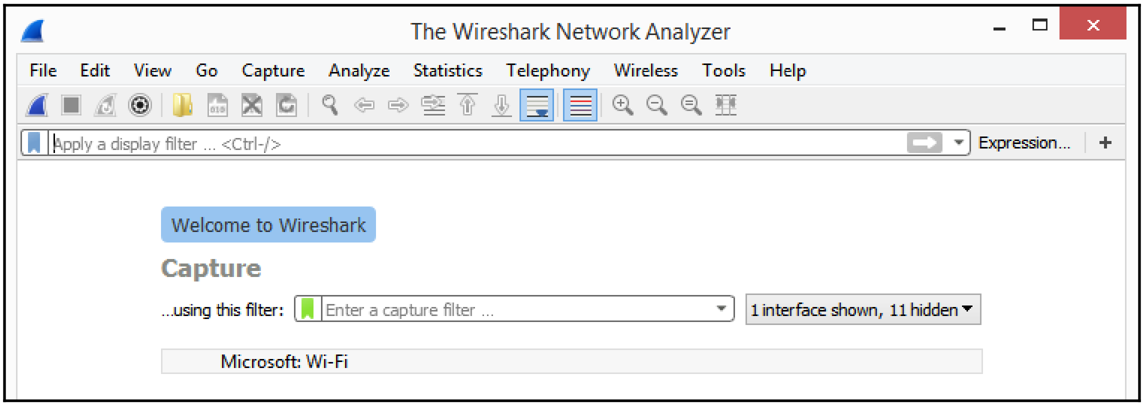
Understanding how to use display filters is important, as you may get a packet capture file from a co-worker who has captured all the traffic coming through the interface, in which case, you would have to make sense of the capture.

However, when you have control of capturing traffic when working on an enterprise network, capture filters help target only the traffic you want to see and remove much of the noise.

There is one thing to keep in mind when using a capture filter; although the capture filters interface may look like the display filter toolbar, the syntax for the capture filters is different. Therefore, when creating a capture filter, you need to be careful that you use the correct syntax.

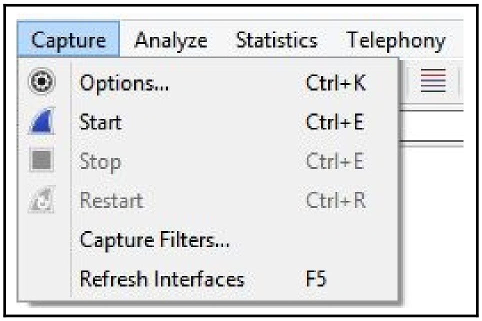
You can create a capture filter in a couple of ways:

- Go to the center of the startup screen and enter the filter in **...using this filter:**, as shown in the following screenshot:


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
1 interface shown, 11 hidden
Microsoft: Wi-Fi

Startup screen capture filter

- Go to the **Capture** menu and then select **Options...**, as shown here:



Capture menu drop-down

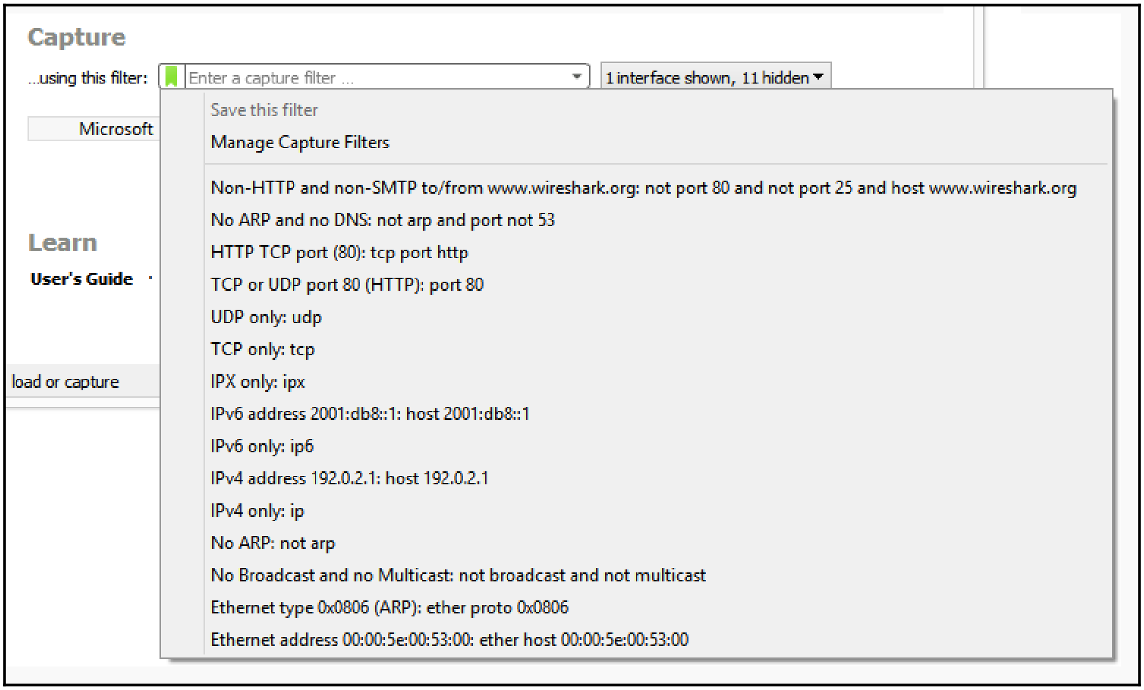
Although both will give you the ability to create a capture filter, I generally go to the **Capture** menu choice and select **Options...**, as this will allow me to see all the interfaces and options.

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.

Saving to bookmarks

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, then you will see a list of capture filters, as shown in the following screenshot:



Non-HTTP and non-SMTP to/from www.wireshark.org: not port 80 and not port 25 and host www.wireshark.org
No ARP and no DNS: not arp and port not 53
HTTP TCP port (80): tcp port http
TCP or UDP port 80 (HTTP): port 80
UDP only: udp
TCP only: tcp
IPX only: ipx
IPv6 address 2001:db8::1: host 2001:db8::1
IPv6 only: ip6
IPv4 address 192.0.2.1: host 192.0.2.1
IPv4 only: ip
No ARP: not arp
No Broadcast and no Multicast: not broadcast and not multicast
Ethernet type 0x0806 (ARP): ether proto 0x0806
Ethernet address 00:00:5e:00:53:00: ether host 00:00:5e:00:53:00

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, then drop down the bookmark icon and select **Save this filter**, and Wireshark will store the filter in the bookmark.

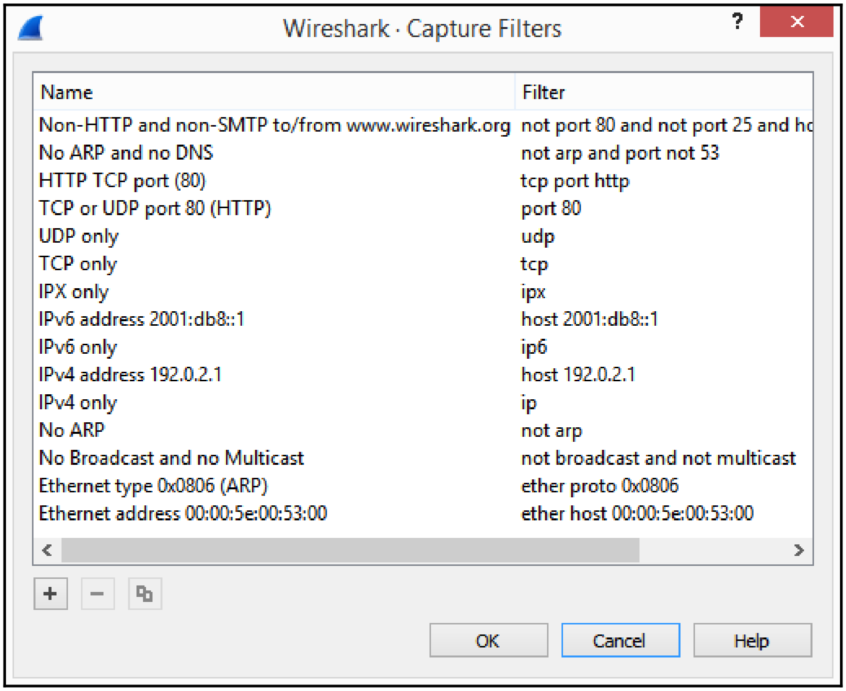
To clear the capture filter, select the **X** on the right-hand side of the capture filter.

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.

Similar to modifying the display filters, you can customize the capture filters, 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:



Name	Filter
Non-HTTP and non-SMTP to/from www.wireshark.org	not port 80 and not port 25 and ho
No ARP and no DNS	not arp and port not 53
HTTP TCP port (80)	tcp port http
TCP or UDP port 80 (HTTP)	port 80
UDP only	udp
TCP only	tcp
IPX only	ipx
IPv6 address 2001:db8::1	host 2001:db8::1
IPv6 only	ip6
IPv4 address 192.0.2.1	host 192.0.2.1
IPv4 only	ip
No ARP	not arp
No Broadcast and no Multicast	not broadcast and not multicast
Ethernet type 0x0806 (ARP)	ether proto 0x0806
Ethernet address 00:00:5e:00:53:00	ether host 00:00:5e:00:53:00

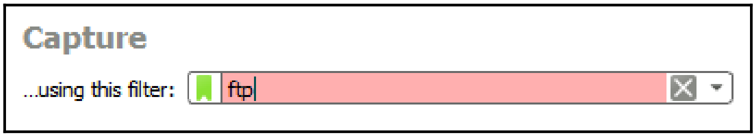
Capture Filters dialog box

At the bottom of the dialog box, there are three icons:

- A plus icon to add a new capture filter
- A minus icon to delete a capture filter
- A copy icon to copy a capture filter

Similar to the plus icon used to add a display filter, you can do the same to add a capture filter. However, you'll need to be careful when crafting a capture filter as it uses different syntax than a display filter.

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, as shown here. Although this filter would work as a display filter, you must write a capture filter that uses the correct syntax:



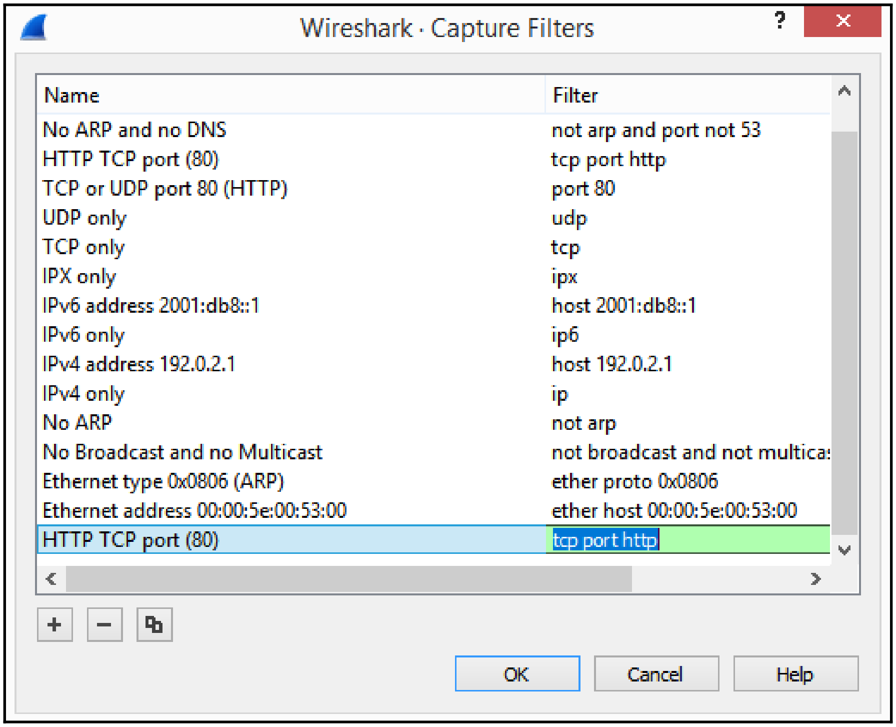
...using this filter: ftp

Invalid capture filter syntax

If you do need to create a new capture filter, try using one of the prebuilt filters as a guide to properly build your 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 the filter

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:



Name	Filter
No ARP and no DNS	not arp and port not 53
HTTP TCP port (80)	tcp port http
TCP or UDP port 80 (HTTP)	port 80
UDP only	udp
TCP only	tcp
IPX only	ipx
IPv6 address 2001:db8::1	host 2001:db8::1
IPv6 only	ip6
IPv4 address 192.0.2.1	host 192.0.2.1
IPv4 only	ip
No ARP	not arp
No Broadcast and no Multicast	not broadcast and not multica
Ethernet type 0x0806 (ARP)	ether proto 0x0806
Ethernet address 00:00:5e:00:53:00	ether host 00:00:5e:00:53:00
HTTP TCP port (80)	tcp port http

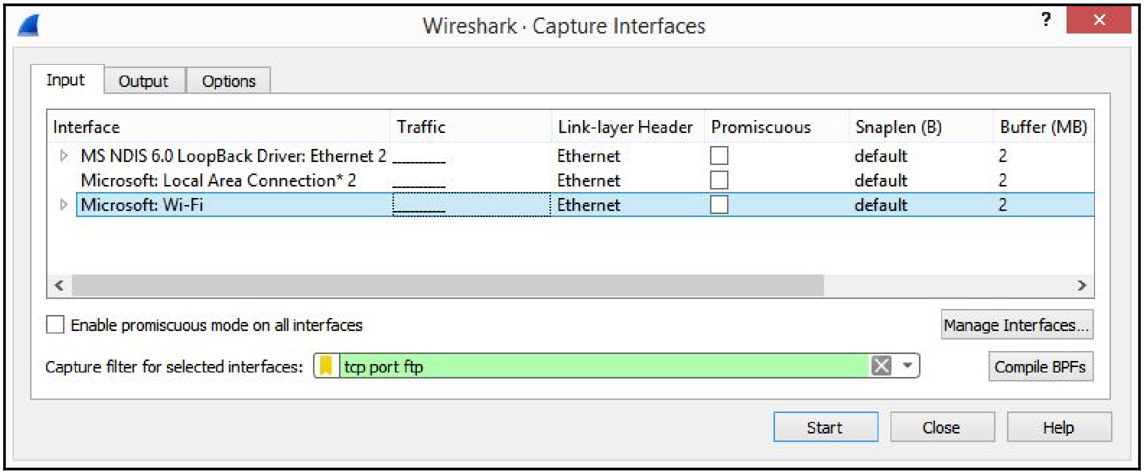
Copy capture filter

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.

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:



Interface	Traffic	Link-layer Header	Promiscuous	Snaplen (B)	Buffer (MB)
▷ MS NDIS 6.0 LoopBack Driver: Ethernet 2		Ethernet	□	default	2
Microsoft: Local Area Connection* 2		Ethernet	□	default	2
▷ Microsoft: Wi-Fi		Ethernet	□	default	2

□ Enable promiscuous mode on all interfaces

Capture filter for selected interfaces: tcp port ftp

FTP capture filter

Once you click **Start** to begin your capture, you will only capture FTP traffic.

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.

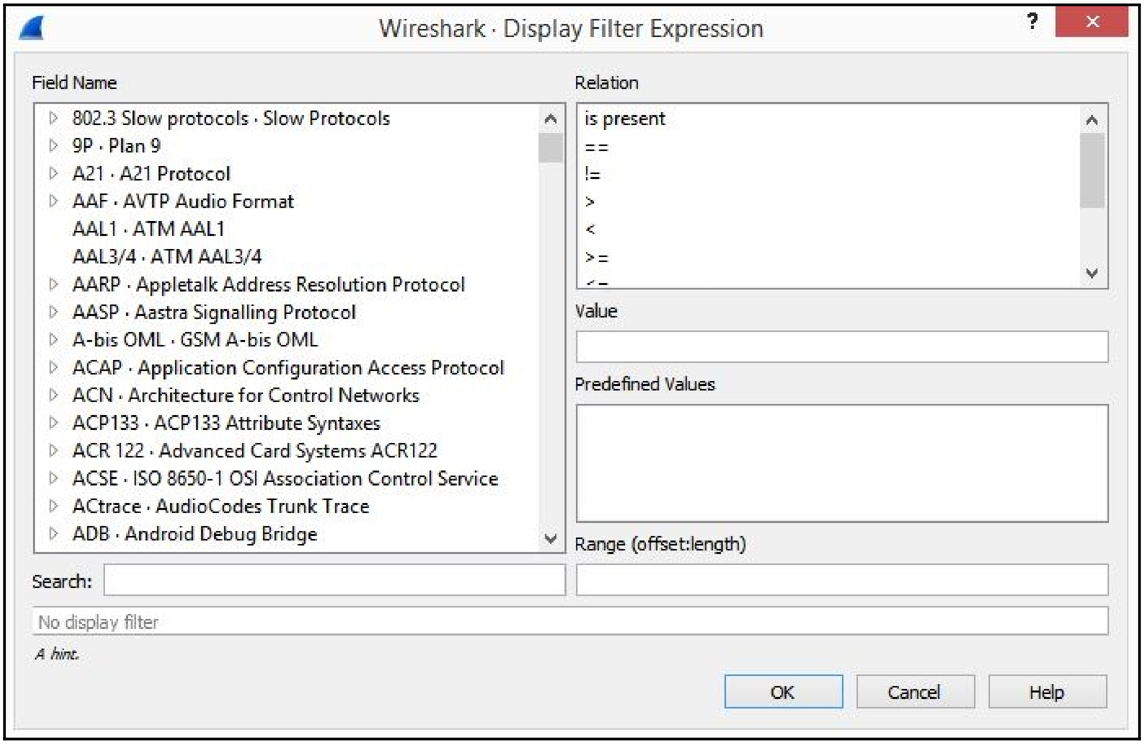
For an extensive list of examples, go to the Wireshark Wiki at `https://wiki.wireshark.org/CaptureFilters`.

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 malware analysis.

After building a few simple filters, you may need to create a more complex filter or expression. The following section outlines how to use the expression builder.

Understanding the expression builder

On the right-hand side of the display filter is the **Expression** button, which, when clicked, will open a dialog box, as shown in the following screenshot:



Wireshark · Display Filter Expression

Field Name

- 802.3 Slow protocols · Slow Protocols
- 9P · Plan 9
- A21 · A21 Protocol
- AAF · AVTP Audio Format
- AAL1 · ATM AAL1
- AAL3/4 · ATM AAL3/4
- AARP · Appletalk Address Resolution Protocol
- AASP · Aastra Signalling Protocol
- A-bis OML · GSM A-bis OML
- ACAP · Application Configuration Access Protocol
- ACN · Architecture for Control Networks
- ACP133 · ACP133 Attribute Syntaxes
- ACR 122 · Advanced Card Systems ACR122
- ACSE · ISO 8650-1 OSI Association Control Service
- ACtrace · AudioCodes Trunk Trace
- ADB · Android Debug Bridge

Relation

- is present
- ==
- !=
- >
- <
- >=

Value

Predefined Values

Range (offset:length)

Search:

No display filter

A hint.

Display Filter Expression

On the left-hand side, you will see a list of all of Wireshark's supported protocols. Wireshark is capable of dissecting hundreds of protocols, with more added all the time, so the list will be long. In order to find a protocol, you can use the search tool. In the preceding screenshot, I have entered `tcp` in the search tool and then expanded the available field names. 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)**: 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.

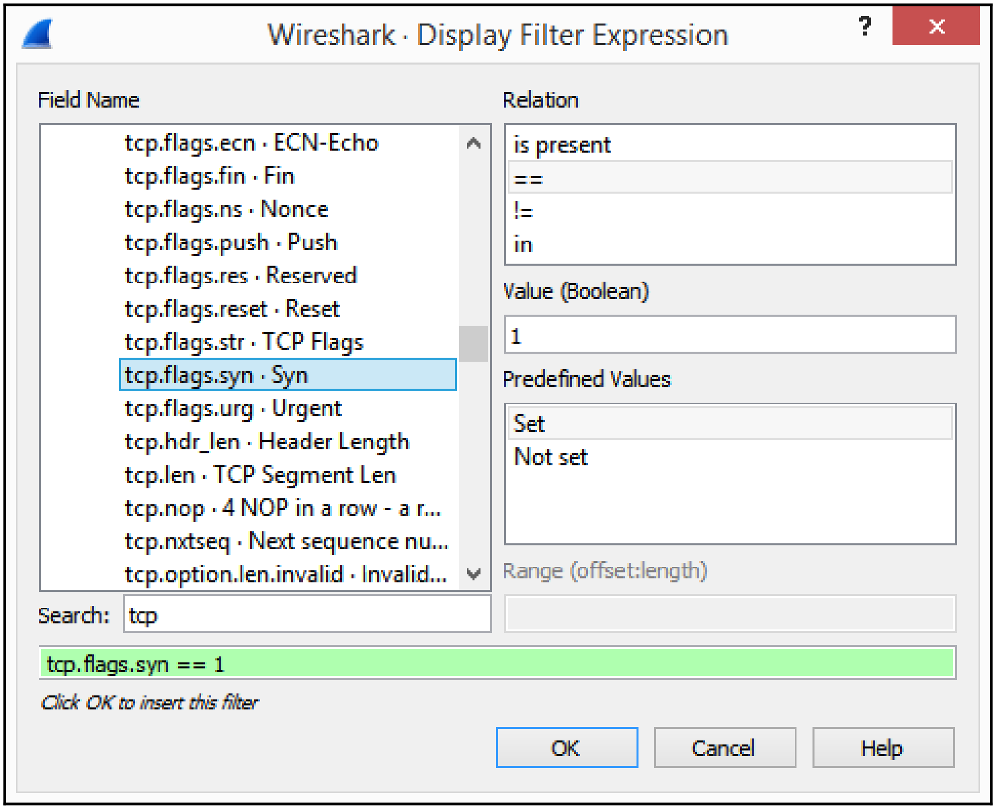
Building an expression

In this example, I want the filter to show me all the packets that have the **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**

The **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:



Wireshark · Display Filter Expression

Field Name

tcp.flags.ecn · ECN-Echo
tcp.flags.fin · Fin
tcp.flags.ns · Nonce
tcp.flags.push · Push
tcp.flags.res · Reserved
tcp.flags.reset · Reset
tcp.flags.str · TCP Flags
tcp.flags.syn · Syn
tcp.flags.urg · Urgent
tcp.hdr_len · Header Length
tcp.len · TCP Segment Len
tcp.nop · 4 NOP in a row - a r...
tcp.nxtseq · Next sequence nu...
tcp.option.len.invalid · Invalid...

Search: tcp

Relation

is present
==
!=
in

Value (Boolean): 1

Predefined Values

Set
Not set

Range (offset:length)

```
tcp.flags.syn == 1
```

Click OK to insert this filter

TCP SYN flag filter

At that 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 complex filter. 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.

While working with packet capture, Wireshark has many shortcuts that allow you to quickly create a filter to streamline your workflow. Let's take a look.

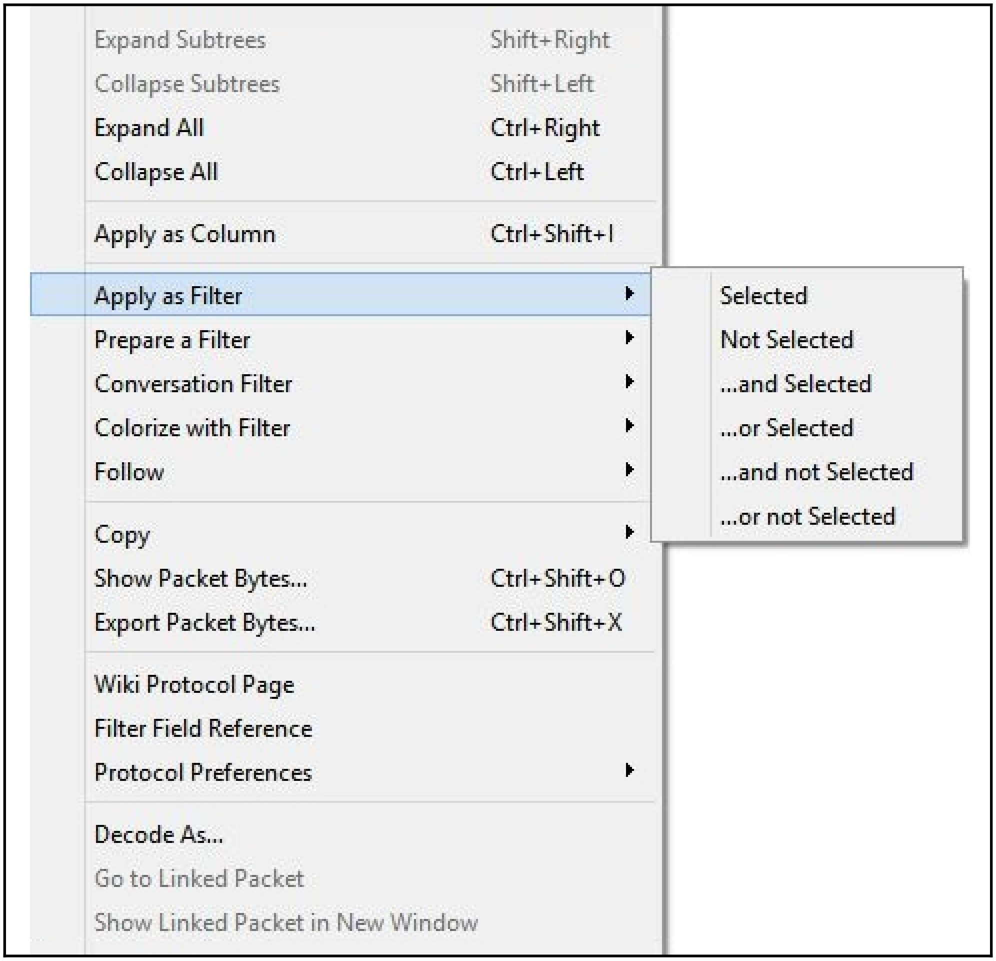
Discovering shortcuts and handy filters

Over the years, Wireshark has evolved. Now, more than ever, it's very easy to create a display filter on the fly while doing analysis by simply right-clicking and choosing to apply or prepare a 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 to view filter shortcuts and you will see several shortcuts, as shown here:

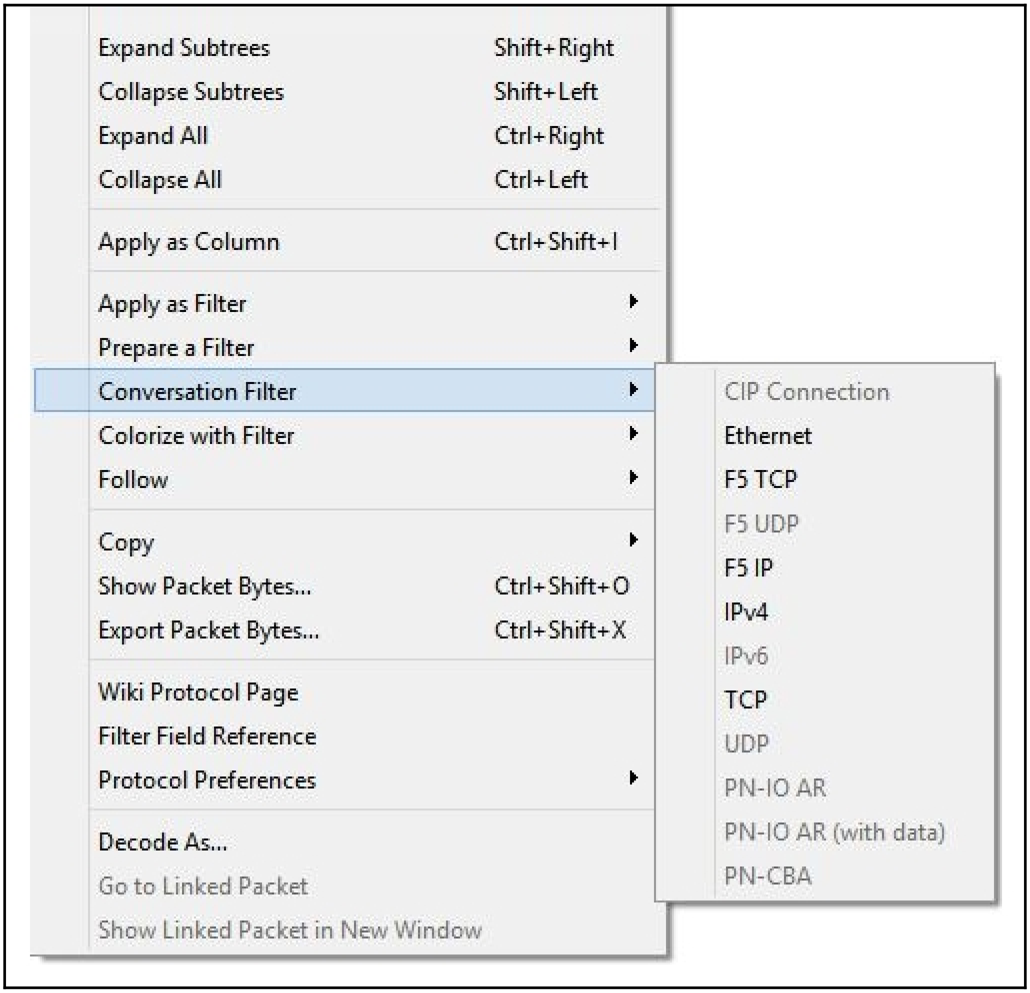


Right-click to view filter shortcuts

Although there are many options when you right-click, in the center are the shortcuts that deal with filters. The following options are available:

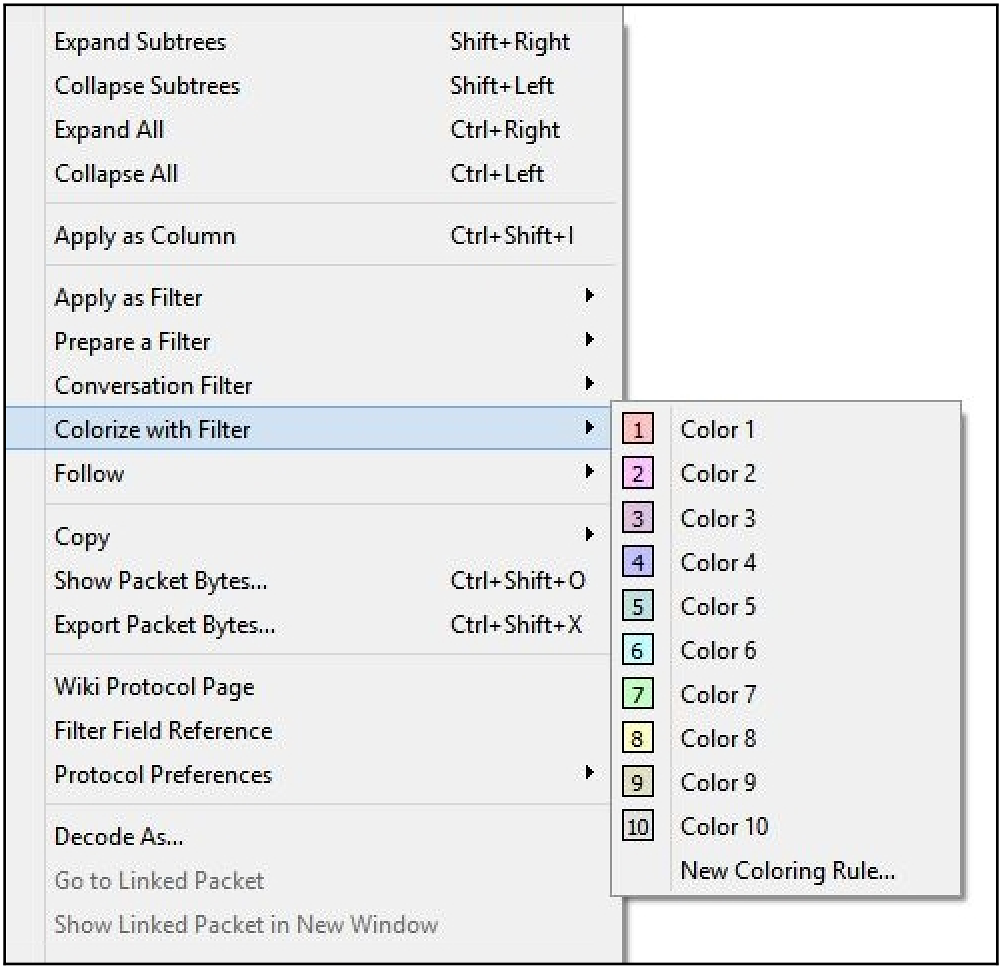
- **Apply as a Filter**: When selected, it will create and run a selected field value.
- **Prepare a Filter**: When selected, it will create and place a selected field value in the display filter area, giving you a chance to make any modifications or add to the filter.

- **Conversation Filter**: When selected, it allows you to follow the conversations according to protocols, such as **Ethernet**, **IPv4**, and **TCP**, as shown in the following screenshot:



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:



Colorize with filter

When you right-click and select either **Apply as a Filter** or **Prepare a Filter**, you will see additional choices, as shown in the screenshot named *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.

After working with Wireshark for a while, you may learn some new techniques that will help improve your workflow. 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.

For example, some handy **display filters** 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 appropriate to collect specific traffic, such as 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.

There are many ways to filter traffic to only display the traffic you want to see, which helps remove the 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 look at traffic more effectively by using the built-in filter functions. We have compared the differences between display and capture filters. In order to filter traffic, we learned how to use a display filter and discussed how it can provide a simple filter showing only a protocol or a combination of field values. We have reviewed how to edit the display or capture filters, as well as creating your own and storing them for easy reference in the bookmarks.

In addition to display filters, we covered capture filters that you apply prior to capture, and the result will display only the traffic that you have captured. To carry out a granular investigation, we have discussed how to create an expression that includes 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 analyses, and then evaluated the benefits of having several useful filters in your arsenal.

In the next chapter, we will take a look at encapsulation in the OSI model, which is an essential concept to grasp in order to be effective at packet analysis. So that you have a better understanding of this important concept, we'll review the seven layers, discuss addressing, the protocol data units, and the protocols in each layer, and the process of encapsulation as the data is readied for frame formation in order to be sent on 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 *Assessment:*

1. When creating a display filter, if the background is ____, then you have entered a valid filter.
    1. Red
    2. Green
    3. Yellow
    4. 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.
    1. Float
    2. Integer
    3. Value
    4. Boolean
3. To create a capture filter to see only DNS requests and responses you would enter _____ in the capture filter.
    1. tcp port 53
    2. DNS
    3. dns
    4. udp port 53
4. If you need to build a complex filter, then use the _____ Builder.
    1. Expression
    2. Berkley
    3. EPAN
    4. Dissector
5. When using either Apply as a Filter or Prepare a Filter, you will see additional choices to create a simple filter or add logical operators. _____ adds a filter that removes the selected field.
    1. ...or Selected
    2. ...and Selected
    3. Not Selected
    4. ...and not Selected

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 OS transforms, encapsulates, and prepares data for transport on the network. In this chapter, we'll cover the seven layers, along with the role and purpose of each layer. So that you understand the significance of a port, IP, and MAC address, we'll go through addressing at the transport, network, and data link layer. We'll then take a look at the **Protocol Data Unit** (**PDU**) for each layer.

Once done, you will be more familiar with the terminology, along with having a better understanding of some of the protocols in each layer. From the HTTP request when retrieving data from a web page, to the bits as it travels across the network, you'll know how data transforms and feeds into the next layer to properly format the frame, so the data can be sent on the appropriate media.

This chapter will cover the following:

- Outlining an overview of the OSI model
- Discovering the purpose, protocols, and PDUs
- Exploring the encapsulation process
- Demonstrating frame formation in Wireshark

Comprehending the OSI model

The OSI model is a reference model that outlines the main functions of each layer. Developing the framework began long ago. We started this journey from the late 1960s to the mid-1970s, where we saw an expansion in computing, along with advances in technology in general. In addition, there was the development of computers, from small personal computers to large supercomputers such as the Cray in 1976, along with 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** (**CCITT**), began working on a reference model to define and standardize networking interoperability. Ultimately, in 1983, the two developed the OSI model.

The OSI model and serves many purposes, such as the following:

- Providing a common framework for developers
- Narrowing down problems for network administrators
- Enabling interoperability among layers and communicating devices
- Breaking down each layer to help students better understand the overall process of data encapsulation.

Developers reference the OSI model to outline how systems communicate with one another. When troubleshooting, it's common to refer to problems according to the layer they feel is responsible for the malfunction. Equipment manufacturers rely on the OSI model to ensure their products will work across all layers.

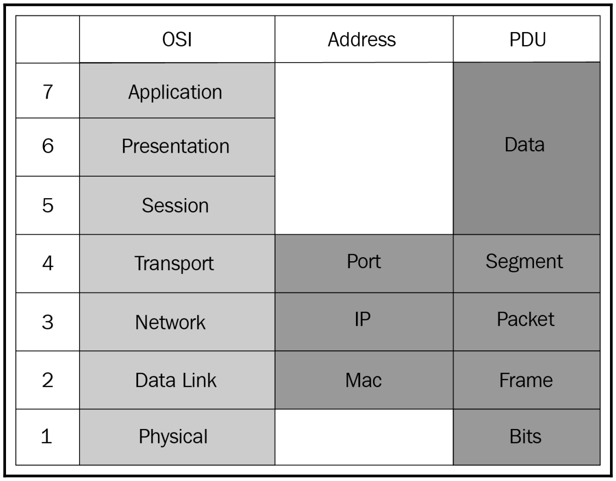
Networking students use the model to begin their journey into networking. A staple of every freshman networking class is an introduction to the OSI model. In most cases, the students have never heard of this, so presenting a complex topic in a simple manner can be difficult. Although this is their first 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, in terms of protocols, PDUs, and the purpose of each layer, as outlined in the following section.

Discovering the purpose, protocols, and PDUs

The OSI model has seven layers. The next diagram shows several elements of the OSI model. From the left, we see the following:

- The number that identifies the layer
- The name of the layer
- An appropriate address for the transport, network, and data link layers
- The corresponding PDU

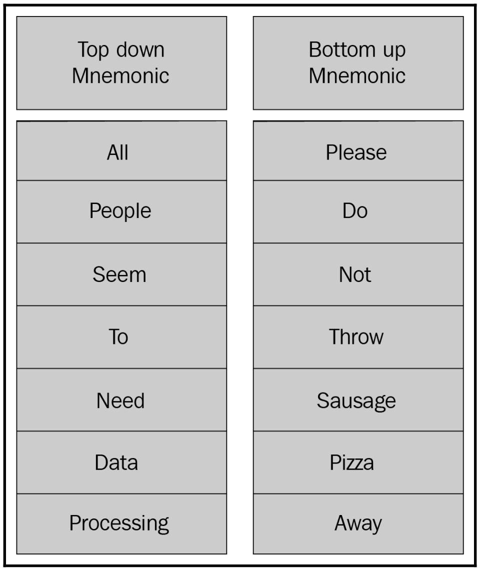


	OSI	Address	PDU
7	Application		Data
6	Presentation		
5	Session		
4	Transport	Port	Segment
3	Network	IP	Packet
2	Data Link	Mac	Frame
1	Physical		Bits

The OSI model

The seven layers are, from layer seven to layer one: **Application**, **Presentation**, **Session**, **Transport**, **Network**, **Data Link**, and **Physical**.

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

OSI mnemonics

During encapsulation, the transport, network, and data link layers use appropriate source and destination addressing:

- **Transport layer**: Uses a port address
- **Network layer**: Uses an IP address
- **Data link layer**: Uses a MAC address

At each layer, the data is in a specific format—called a PDU—that defines what shape the data is in as it is passed to the layer above or the layer below and includes **D**ata, **S**egment, **P**acket, **F**rame, and **B**its. A mnemonic device to remember the PDU is **D**o **S**ome **P**eople **F**ear **B**inary?

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, I commonly start with layer 7, or the application layer, as this is the layer where we initiate contact with the network, as discussed next.

Evaluating the application layer

At the top of the OSI model graphic is the application layer, or layer 7. This layer contains protocols that allow process-to-process communications, and it's where we initiate contact with the network to perform the following:

- Retrieve a web page.
- Fetch or send our email.
- Upload files to an FTP server.
- Request a dynamically assigned IP address.

On the IP network, each application layer protocol follows specific recommendations, requirements, and options, according to its function. Let's talk about a few application layer protocols along with the PDU.

Exploring protocols and the PDU

The application layer has hundreds of protocols. Some common protocols were developed and standardized very early in the 1980s. Some are deprecated and we rarely see them, such as Telnet and CharGEN. New protocols are developed as needed to keep up with today's demands, such as Bitcoin and MQTT. The following is a shortlist of application layer protocols:

Protocol	**Purpose**
Simple Mail Transfer Protocol (**SMTP**)	Transports email
Hypertext Transfer Protocol (**HTTP**)	Ensures delivery (transfer) of web pages the proper format.
File Transfer Protocol (**FTP**)	Transfers files between a client and server across the network
Message Queuing Telemetry Transport (**MQTT**)	Used with IoT devices, sensors, and mobile devices as a lightweight messaging protocol

All protocols have a specific purpose on the network. Today, Wireshark has dissectors for most, but not all protocols. However, new dissectors are added all the time.

The PDU for the application layer is data. In many cases, the protocols in this layer are involved in a series of client requests and server responses.

The header structure will vary as each is application specific. In addition, the header for the client is generally different than the header for the server.

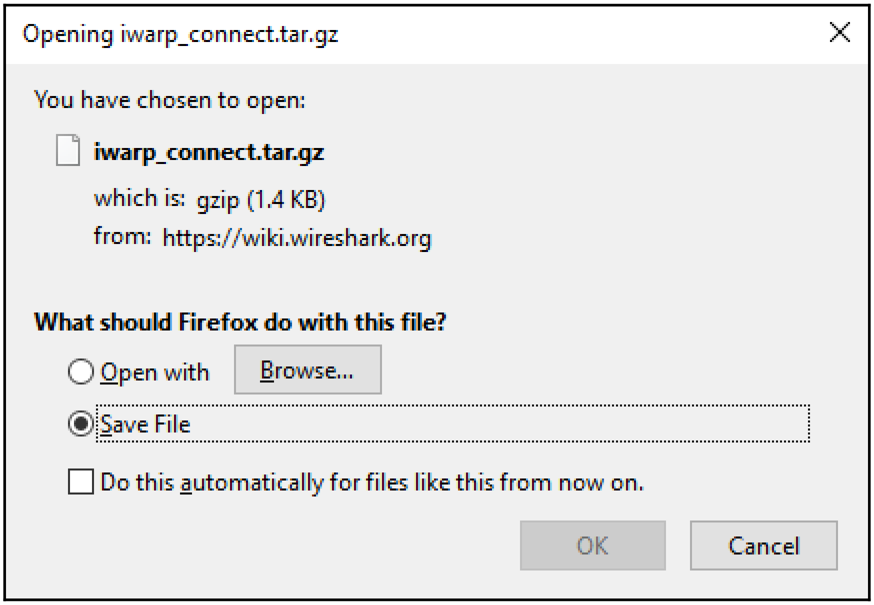
Most protocols will have an associated port, which will be found in the transport layer header. The following is a summary of the application layer:

Layer	**Purpose**	**Protocols**	**PDU**
Application	Initiates contact with the network	HTTP, DNS, FTP	Data

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.

Understanding the presentation layer

Layer 6 is the presentation layer and is responsible for proper data formatting, along with optional compression and encryption. The presentation layer ensures that the data is in the proper format, either before presenting the data to the application, or before sending it to 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:



Opening iwarp_connect.tar.gz

You have chosen to open:

iwarp_connect.tar.gz

which is: gzip (1.4 KB)

from: https://wiki.wireshark.org

What should Firefox do with this file?

○ Open with

● Save File

□ Do this automatically for files like this from now on.

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 to compress and decompress data. Compression removes redundancy and makes data smaller. This function is optional, as not all data is compressed.

This layer also handles encryption, which is scrambling data by using a key so that it is in an unreadable form that does not make sense to anyone unless they have the key. Because encryption is also an optional function, this may not be required.

In the presentation layer, we do see a few protocols, which we'll investigate in the following section, along with the PDU.

Describing the protocols and the PDU

Protocols in the presentation layer deal with proper data translation and encoding/decoding, such as **External Data Representation** (**XDR**). In addition, because of of the presentation layers role in encryption, you'll find the following protocols:

- **Transport Layer Security (TLS)/Secure Socket Layer (SSL)**: Secures end-to-end communications such as bank transactions and web page retrieval using encryption
- **Secure/Multipart Internet Mail Extensions (S/MIME)**: Digitally signs and encrypts email messages

At the presentation layer, the PDU is still data, where we see the data being translated or converted into the correct format. The following is a summary of the presentation layer:

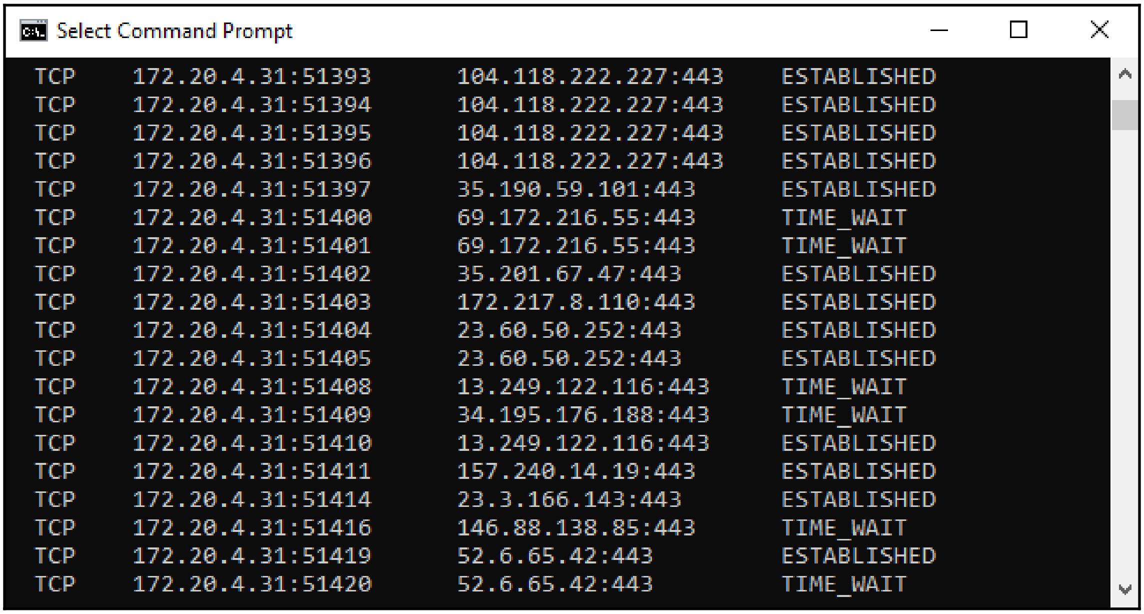
Layer	**Purpose**	**Protocols**	**PDU**
Presentation	Formatting and optional compression and encryption	TLS/SSL, S/MIME	Data

In many ways, the presentation layer is an extension of the application layer. The next layer is the session layer, where we see all 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. The OS creates the appropriate socket, which is an IP address, and a port, so the two endpoints can communicate 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:



```
  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
```

Netstat showing active connections

In this 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.

In addition to setting up a session, this 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 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 may re-request any missing data.

After the session has ended, the session layer safely closes the session. Next, let's look at some key protocols in this layer.

Recognizing protocols and the PDU

There are several protocols that exist in the session layer. Although most protocols may originate in other layers, these protocols begin in part in the session layer:

- **Real-Time Transport Control Protocol (RTCP)**: This works along with RTP to deliver control information to all participants in a 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	Set up, maintain, and tear down a session	DNS, RPC	Data

The session layer manages all aspects of a session that enable 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 selected will depend on the application. Data is transported primarily using either TCP or **User Datagram Protocol** (**UDP**). However, the transport layer has several other protocols. Let's take a look.

Differentiating protocols and the PDU

The transport layer has several protocols to transport data, including the following:

Protocol	**Purpose**
TCP	Connection-oriented protocol that ensures reliable data transfer
UDP	Connectionless protocol used when speed, not reliability, is required
Stream Control Transmission Protocol (**SCTP**)	Reliably transmits data streams that have more than one IP address
Reliable User Datagram Protocol (**RUDP**)	Extends UDP by providing TCP-like qualities, such as flow control, acknowledgments, and re-transmitting lost packets

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.

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 FIN packets. TCP has an 11-field header, and sequences and acknowledges data, to ensure that all the data arrives at the end device.

Once in a connection, TCP progresses through a series of states. For example, in the screenshot named *Netstat showing active connections*, you can see the **ESTABLISHED** and **TIME-WAIT** states.

TCP states are 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**: 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 set to the remote host, or a FIN 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 to make sure 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 can see in this next section.

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 **Dynamic Host Configuration Protocol** (**DHCP**), **Routing Information Protocol** (**RIP**), **Voice over IP** (**VoIP**), or **Trivial File Transfer Protocol** (**TFTP**).

Whether TCP or UDP is in use, the transport layer is a critical component of ensuring 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, as 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 – 1,023 and include protocols such as HTTP, DNS, and SMTP.
- **Registered** ports range between 1,024 – 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 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

After the transport layer, the next layer is the network layer. As we'll see in the next section, the network 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. While the transport layer transports the data, 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.

IP

IP is a best-effort, connectionless protocol that routes packets from source to destination using a logical IP address. The original RFC for IP was written in 1981, as seen at `https://tools.ietf.org/html/rfc791`. IP was standardized shortly after that.

Many of the original protocols in the TCP/IP suite have had minor changes, updates, and modifications over the years. 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 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.

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 IP address to a MAC address on a local area network so the frame can be delivered.

> ARP is an unusual protocol because it is in between layer three and layer two of the OSI model. ARP resolves an IP (network layer) address to a MAC (data link layer) address. However, many consider it to be a layer 3 protocol.

In addition to IP and ARP, we also need ICMP to help report any problems that may have occurred during data transport, as discussed in the following segment.

ICMP

ICMP is another critical network protocol that 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 RFC, which is found at `https://tools.ietf.org/html/rfc792`. ICMP reports on issues encountered during transit such as network unreachable and host 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

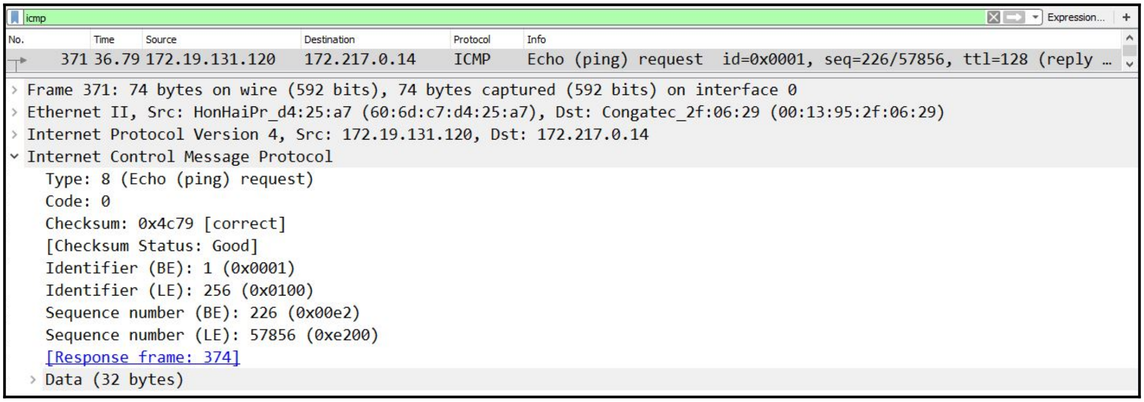
In this layer, the IP header will hold source and destination addresses in either IPv4 or IPv6 format. Both are referred to as logical addresses and are represented as follows:

- 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 these two 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 address because ARP is a service protocol that resolves IPv4 addresses to MAC addresses.
- ICMP is the sister protocol to IP, and reports on matters encountered during transit, such as network unreachable and host 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:



```
No.   Time  Source          Destination   Protocol  Info
  371 36.79 172.19.131.120  172.217.0.14  ICMP      Echo (ping) request  id=0x0001, seq=226/57856, ttl=128 (reply …
> Frame 371: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
> Ethernet II, Src: HonHaiPr_d4:25:a7 (60:6d:c7:d4:25:a7), Dst: Congatec_2f:06:29 (00:13:95:2f:06:29)
> Internet Protocol Version 4, Src: 172.19.131.120, Dst: 172.217.0.14
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4c79 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 226 (0x00e2)
    Sequence number (LE): 57856 (0xe200)
    [Response frame: 374]
  > Data (32 bytes)
```

ICMP echo request

By now, you should understand that the network layer allows hosts on separate networks to communicate with one another by providing logical addressing and path determination.

The following chart outlines a summary of the network layer:

Layer	Purpose	Protocols	PDU
Network	Addressing, routing	IP, ICMP	Packet

Network layer summary

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 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 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.

The PDU at this layer is a frame. Each frame requires an address, as outlined 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 chart provides a summary of the data link layer:

Layer	**Purpose**	**Protocols**	**PDU**
Data link	Frame formation	Ethernet, HLDC	Frame

Data link layer summary

The data link layer ensures proper frame formation and link access, along with error detection, while traveling across the network media. Data then travels on 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. 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

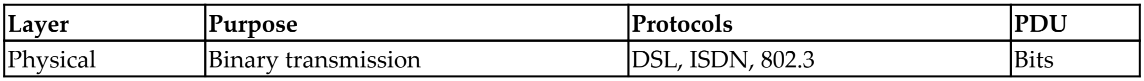
Exemplifying protocols and the PDU

In the physical layer, there are several different protocols used to transmit data across the 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 GB Ethernet.

The physical layer is where binary transmission takes place across the network media. At this layer, the PDU is the most basic form, which is bits.

The following chart provides a summary of the physical layer:



Layer	Purpose	Protocols	PDU
Physical	Binary transmission	DSL, ISDN, 802.3	Bits

Physical layer summary

Prior to traveling across the media, 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 across the media.

Exploring the encapsulation process

Now that we know the layers, let's look at how each of the layers work together during the encapsulation process to create a frame.

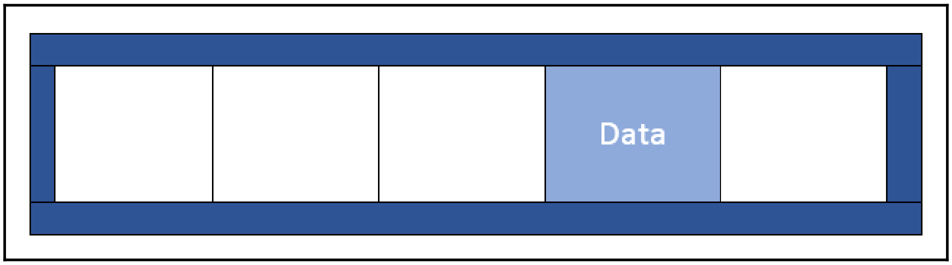
During frame formation, the process begins with data. As the data moves down the layers, a header is added one by one until the frame is complete. Each frame has the following components:

- Data and appropriate application layer header (if applicable)
- Segment header
- Packet header
- Frame header

We'll start with the data portion of the frame.

Viewing the data

In most cases when frame formation begins and the encapsulation takes place, we start with the data, as shown here:



The encapsulation process—data

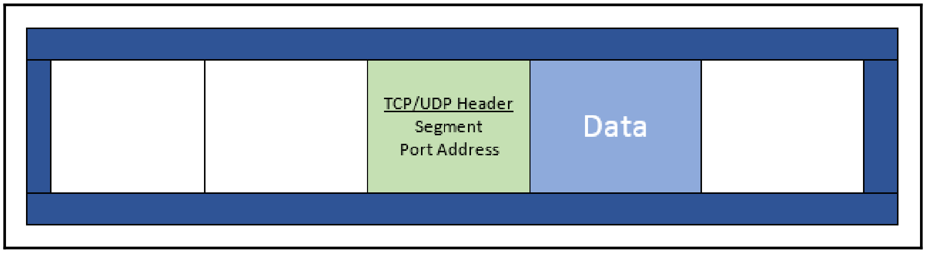
The data might be any of the following:

- 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 or UDP header:

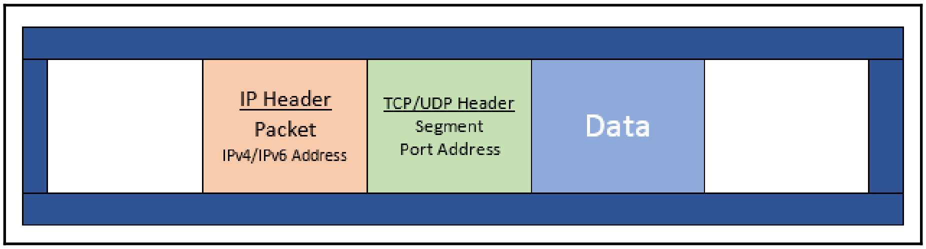


The encapsulation process—segment

The segment holds a source and destination port address, as shown in the preceding diagram. The next stop in the encapsulation process is to add an IP header in order for it to become a packet, as discussed next.

Identifying the packet

As data is encapsulated, we now have data, along with a segment that holds either a TCP or UDP port. The next part of encapsulation is creating a packet by adding the source and destination IPv4 or IPv6 address in the IP header, as shown here:

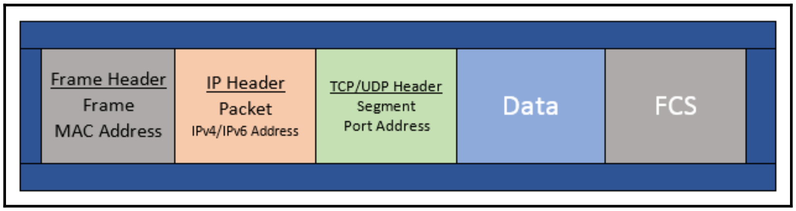


The encapsulation process—packet

The last part of the encapsulation process is the addition of the frame header, as shown 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 data, a transport layer header or segment, and a network layer or packet. And now, we complete the frame by adding the source and destination MAC address, as shown here:



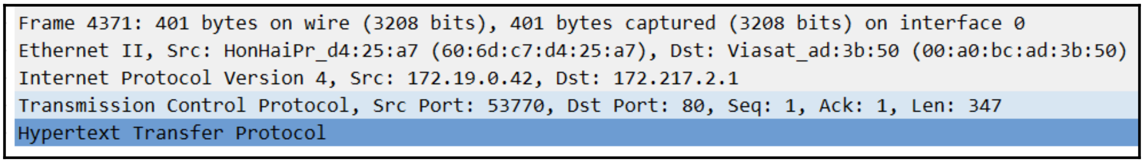
The encapsulation process—frame

With a frame, not only do we also have a header, but we also have a trailer, which is called the frame check sequence. The frame check sequence holds a value called a cyclic redundancy check, which is used for error detection on the network that is checked while traveling along on its journey.

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, this will help you to to learn about 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
```

Frame formation in Wireshark

Not all frames contain data, but this one does, so it is a good example of a fully encapsulated frame.

When looking at a single frame, you will see, at the top in the **Frame 4371** line, the metadata about that single frame. After the frame metadata is the following:

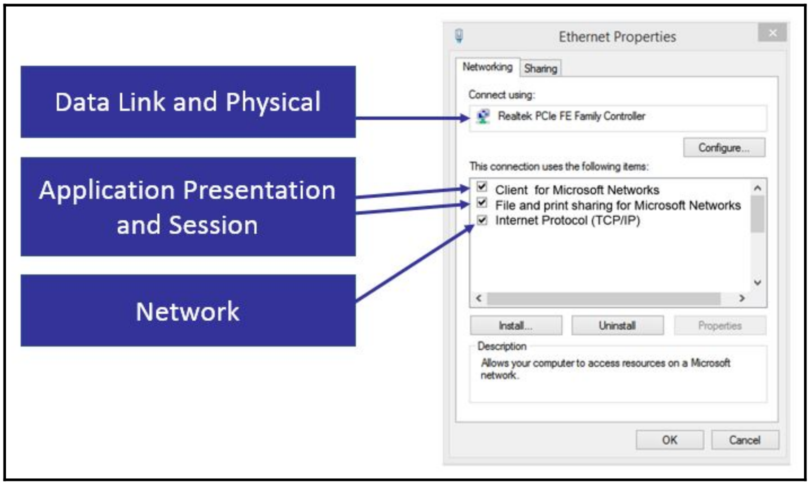
- **Frame**: The frame header shows Ethernet II, and after that are the source and destination MAC addresses.
- **Packet**: The IP header represents the network layer that holds the source and destination IP addresses.
- **Segment**: The TCP header represents the transport layer, which holds the source 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

Within 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 layer in the OSI model are represented:



Connect using:
Realtek PCIe FE Family Controller

This connection uses the following items:

☑ Client for Microsoft Networks
☑ File and print sharing for Microsoft Networks
☑ Internet Protocol (TCP/IP)

Description
Allows your computer to access resources on a Microsoft network.

Network bindings

The following represents the various layers and how they are represented in **Ethernet Properties**:

- Data link and physical layers are represented in the NIC.
- Application, presentation, and session layers are represented in **Client for Microsoft Networks**, along with **File and print sharing for Microsoft Networks**.
- Network layers are shown as **Internet Protocol (TCP/IP)**.

Summary

In this chapter, we took a closer look at an important concept, the OSI model and the encapsulation process. The OSI model is essential and serves many purposes that include providing a common framework for developers and a method to help students understand what process occurs at each layer. Understanding each of the layers, the protocol data unit, and the addressing will help you to better understand the process in Wireshark.

By now, you should have a better understanding of the role, purpose, and protocols, along with the PDU of each layer. We explored the encapsulation process along with taking a look at frame formation as it is seen in Wireshark. To help you learn about 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 decoding two main transport layers: TCP and UDP. We'll review TCP, and then examine the 11-field header format in Wireshark. We'll cover the three-way handshake that is used to start a session along with the four-way FIN exchange to end a session. Then, we'll go through an overview of UDP and examine the four-field header.

Questions

Now it's time to check your knowledge. Select the best responses, and then check your answers with those in the *Assessment*:

1. The _____ layer, or layer 5, is responsible for setting up, maintaining, and tearing down a session.
    1. Transport
    2. Application
    3. Session
    4. Presentation
2. The _____ layer, or layer 4, is responsible for transporting the data, either using a connectionless or connection-oriented protocol.
    1. Transport
    2. Application
    3. Session
    4. Presentation
3. The _____ layer, or layer 6, is responsible for proper data formatting along with optional compression and encryption.
    1. Transport
    2. Application
    3. Session
    4. Presentation
4. TCP port 334 is in the range of the _____ ports.
    1. Ephemeral
    2. Well-known
    3. Registered
    4. Secure
5. The PDU at the transport layer is _____.
    1. Data
    2. Frame
    3. Packet
    4. Segment

3

Section 3: The Internet Suite TCP/IP

The internet suite examines protocols that move data – TCP, UDP, IP, ICMP, and ARP – and takes a close look at the handshake and teardown processes.

This section is comprised of the following chapters:

9

Decoding TCP and UDP

Since its standardization in 1983, 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 System 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 will review the role and purpose of the Transport Layer. So that you have a better understanding of this connection-oriented protocol, we will take a closer look at TCP. We'll examine the header format and field values in detail, such as sequence number, offset, and window size, and review the TCP flags. In addition, we'll review UDP, along with the common uses for this lightweight, connectionless protocol and examine the streamlined four-field header.

The following topics will address all of this:

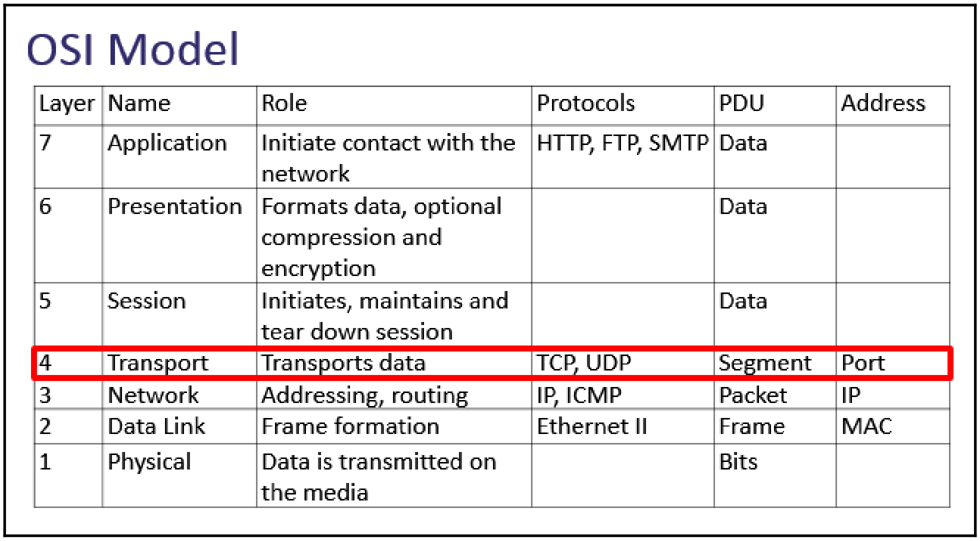
- Reviewing the purpose of the transport layer
- Describing the TCP
- Examining the eleven-field TCP header
- Understanding the UDP
- Discovering the four-field UDP header

Reviewing the purpose of 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 that's selected will depend on the application. There are several protocols in this layer, including the following:

- **Reliable data protocol**: Used to transfer data in a connection-oriented manner
- **Stream control transmission protocol**: Provides the reliable transmission of data streams that have more than one IP address

Although there are other, less well-known transport layer protocols, the two predominant protocols are TCP or UDP, as shown in the following diagram:



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	

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. 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, as evidenced in a draft called *Transport Options for UDP*, which can be found at `https://tools.ietf.org/html/draft-ietf-tsvwg-udp-options-05`.

TCP is a connection-oriented protocol. The eleven-field TCP header tells the story of how impressive this protocol is in its ability to ensure complete delivery of data while monitoring for congestion and providing flow control.

Describing TCP

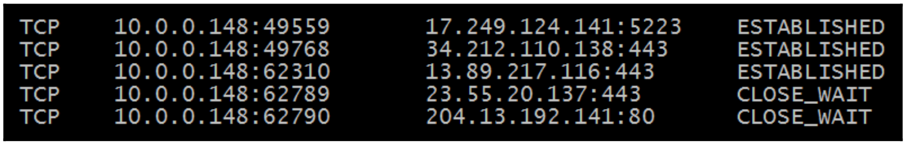
TCP is a connection-oriented protocol that has end-to-end reliability. Connection-oriented means that both endpoints must setup a connection before any data is transferred. To begin a session, TCP starts with a (three-way) handshake.

In many cases, there are TCP header options that outline and further define the parameters of the conversation.

The TCP options are in the first two packets of the three-way handshake and are as follows:

- **Window scaling**: A value that expands the actual Window size by providing a multiplier that more accurately reflects the true Window size
- **Selective Acknowledgements or SACK**: When these are enabled, the receiver will notify the sender if there are any missing packets

Once you have a connection, your operating system creates a socket, which is an IP address and a port. To see all your active TCP connections on a Window machine, open a command-line prompt and run `netstat -anp tcp`, 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
```

Netstat showing TCP connection status

During the conversation, TCP monitors the communication and acknowledges all the data that's received to ensure 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. That is why, in the image, you will see a local IP address and port, along with a sender (or foreign) IP address and port.

Once the conversation is over, TCP ends the session with an exchange of FIN packets.

This powerful protocol also has methods to assist in flow control and congestion control:

- Flow control is an **end-to-end** control method using window size, so the sender doesn't **overwhelm the host**
- Congestion control prevents a node from sending too much data and **overwhelming the network**

There are two state variables involved in congestion control:

- **Congestion Window** (**Cwnd**): The sender-side limit that defines the amount of data a host can send *before receiving* an acknowledgement
- **Receiver Window** (**Rwnd**): The receiver-side limit defines the amount of data a host can receive

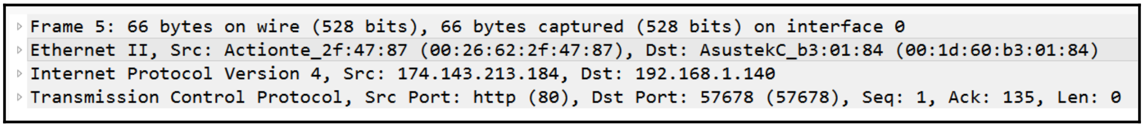
The two variables work together in a TCP connection to regulate the flow of data, minimize congestion, and improve network performance.

It's hard to believe, but there is a great deal of detail in one single frame. Depending on the protocol and the purpose, there are many components, such as the various headers, field values within the headers, along with 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 packet capture file, `HTTP.cap`, in Wireshark.

To follow along, select frame five (5) and focus on the packet details pane, as shown in the following screenshot:

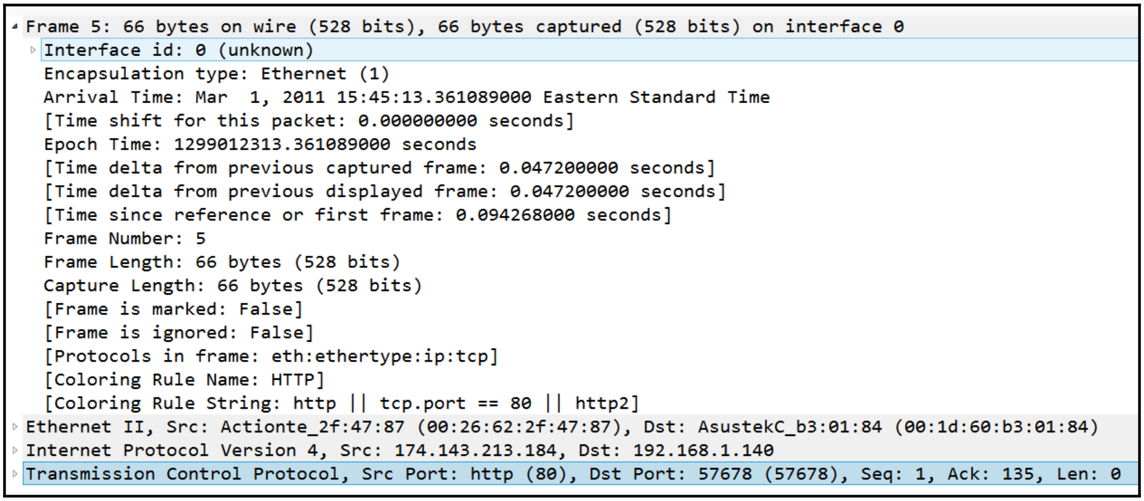


```
Frame 5: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: Actionte_2f:47:87 (00:26:62:2f:47:87), Dst: AsustekC_b3:01:84 (00:1d:60:b3:01:84)
Internet Protocol Version 4, Src: 174.143.213.184, Dst: 192.168.1.140
Transmission Control Protocol, Src Port: http (80), Dst Port: 57678 (57678), Seq: 1, Ack: 135, Len: 0
```

The packet details pane for frame 5

Starting from the top, Wireshark lists the contents of this single frame. Each header has a summary, followed by the details of the header. You can expand the header by clicking on the arrow (or caret, >) on the right-hand side to see the details. In frame 5, we can see the following:

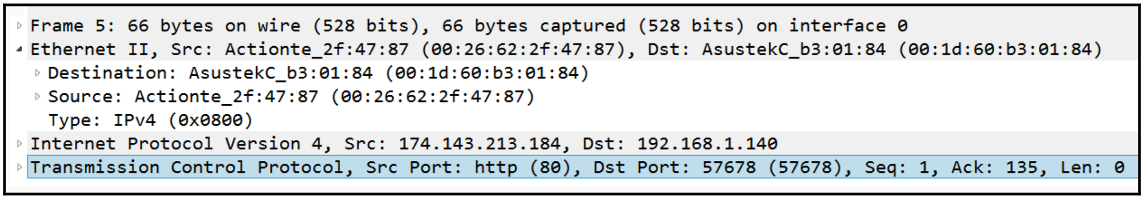
- **Frame 5**: Frame is not a protocol. **Frame** is a list of values generated by Wireshark that describes information about a single frame. Expand the frame by clicking on the arrow on the right-hand side to see the details, as shown in the following screenshot:



```
◂ Frame 5: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
  ▹ Interface id: 0 (unknown)
    Encapsulation type: Ethernet (1)
    Arrival Time: Mar  1, 2011 15:45:13.361089000 Eastern Standard Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1299012313.361089000 seconds
    [Time delta from previous captured frame: 0.047200000 seconds]
    [Time delta from previous displayed frame: 0.047200000 seconds]
    [Time since reference or first frame: 0.094268000 seconds]
    Frame Number: 5
    Frame Length: 66 bytes (528 bits)
    Capture Length: 66 bytes (528 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp]
    [Coloring Rule Name: HTTP]
    [Coloring Rule String: http || tcp.port == 80 || http2]
▹ Ethernet II, Src: Actionte_2f:47:87 (00:26:62:2f:47:87), Dst: AsustekC_b3:01:84 (00:1d:60:b3:01:84)
▹ Internet Protocol Version 4, Src: 174.143.213.184, Dst: 192.168.1.140
▹ Transmission Control Protocol, Src Port: http (80), Dst Port: 57678 (57678), Seq: 1, Ack: 135, Len: 0
```

Frame metadata on a single frame—TCP

- **Ethernet II**: The (true) frame header follows the metadata summary and provides information about the source and destination MAC address, as shown in the following screenshot:



```
▹ Frame 5: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
◂ Ethernet II, Src: Actionte_2f:47:87 (00:26:62:2f:47:87), Dst: AsustekC_b3:01:84 (00:1d:60:b3:01:84)
  ▹ Destination: AsustekC_b3:01:84 (00:1d:60:b3:01:84)
  ▹ Source: Actionte_2f:47:87 (00:26:62:2f:47:87)
    Type: IPv4 (0x0800)
▹ Internet Protocol Version 4, Src: 174.143.213.184, Dst: 192.168.1.140
▹ Transmission Control Protocol, Src Port: http (80), Dst Port: 57678 (57678), Seq: 1, Ack: 135, Len: 0
```

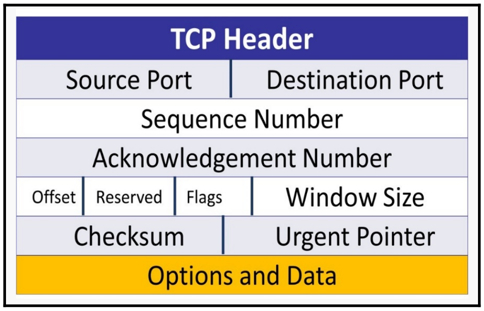
Frame header

- **Internet Protocol Version 4**: The IP header summary includes the source and destination IP address, followed by the IPv4 field values.
- **Transmission Control Protocol**: The TCP header lists the summary, including source and destination ports, sequence and acknowledgement numbers, and length (len), followed by the TCP field values

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.

Examining the eleven-field TCP header

TCP has an eleven-field header that holds the values that keep track of the conversation, as shown in the following diagram:



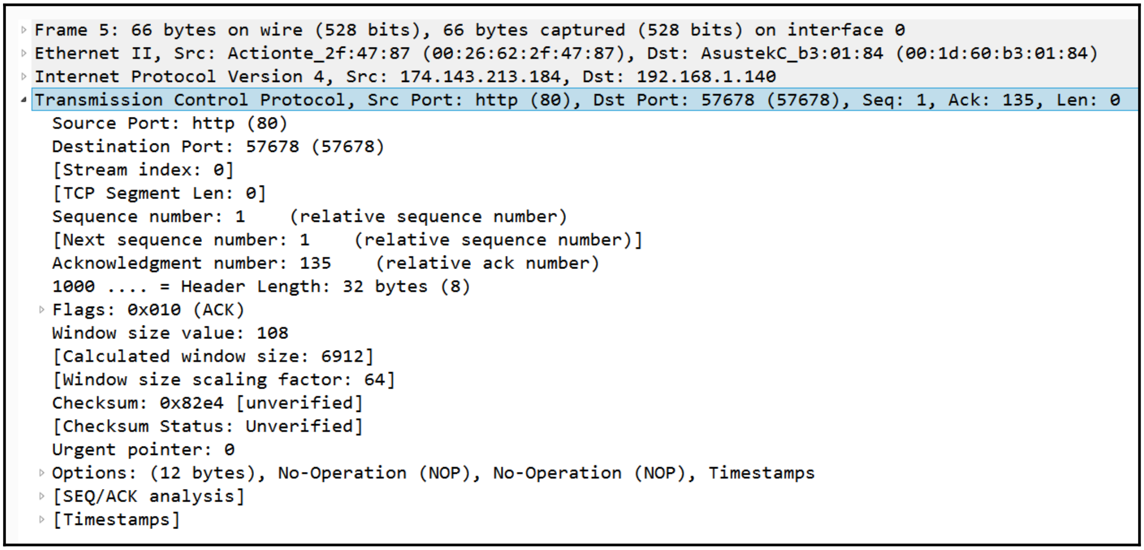
The TCP header

TCP uses the field values to monitor communication. 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 other hosts of any missing segments.

Next, 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.

Navigating the TCP header fields

Starting at the top of the TCP header, we can see the **Transmission Control Protocol**, followed by a summary of what the header represents, as shown in the following screenshot:



```
▹ Frame 5: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▹ Ethernet II, Src: Actionte_2f:47:87 (00:26:62:2f:47:87), Dst: AsustekC_b3:01:84 (00:1d:60:b3:01:84)
▹ Internet Protocol Version 4, Src: 174.143.213.184, Dst: 192.168.1.140
◢ Transmission Control Protocol, Src Port: http (80), Dst Port: 57678 (57678), Seq: 1, Ack: 135, Len: 0
    Source Port: http (80)
    Destination Port: 57678 (57678)
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 135    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▹ Flags: 0x010 (ACK)
    Window size value: 108
    [Calculated window size: 6912]
    [Window size scaling factor: 64]
    Checksum: 0x82e4 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▹ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▹ [SEQ/ACK analysis]
  ▹ [Timestamps]
```

TCP header

Below the summary is the TCP header fields. Whenever there is additional information, you will see an arrow on the right-hand side of the field, which you can expand to see the details.

When looking at the TCP header contents, any information in brackets is generated by Wireshark to help you better understand the details of the header, such as `[SEQ/ACK analysis]` and `[Timestamps]`, as shown near the bottom of the preceding screenshot.

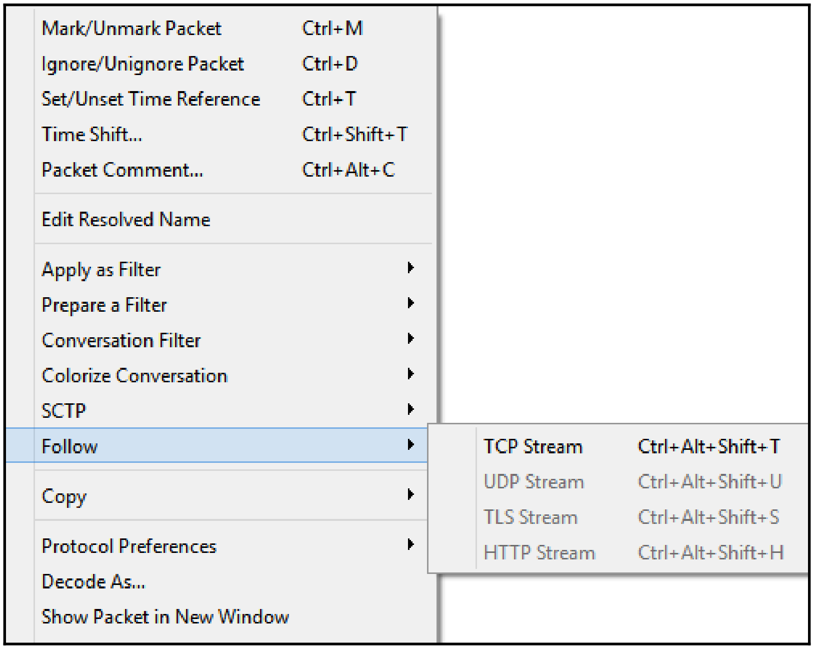
Let's step through the header fields, starting with the ports.

Exploring TCP ports

In Wireshark, you can resolve physical, network, and transport addresses. The packet capture, `HTTP.cap`, uses transport layer name resolution so that whenever a well-known or registered port is used, Wireshark will identify the application associated with the port number.

The following lists the port numbers, along with the generated information Wireshark provides, such as **Stream Index** and **Segment Length**:

- **Source port 16-bit**: This is the port on the sender side. In frame 5, the sender is most likely a web server, as the value is Source Port: http (80).
- **Destination port 16-bit**: This is the port on the receiver (client) side that tells the sender, *When you deliver the data, use this port*. In this case, the value is **Destination 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.
- **Stream index**: This value is shown in brackets, as Wireshark calculates this to keep track of the streams. A stream is a communication between two endpoints. In frame 5, 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 | [TCP, UDP, SSL, HTTP] stream**, as shown in the following screenshot:


Mark/Unmark Packet Ctrl+M
Ignore/Unignore Packet Ctrl+D
Set/Unset Time Reference Ctrl+T
Time Shift... Ctrl+Shift+T
Packet Comment... Ctrl+Alt+C
Edit Resolved Name
Apply as Filter
Prepare a Filter
Conversation Filter
Colorize Conversation
SCTP
Follow
Copy
Protocol Preferences
Decode As...
Show Packet in New Window
TCP Stream Ctrl+Alt+Shift+T
UDP Stream Ctrl+Alt+Shift+U
TLS Stream Ctrl+Alt+Shift+S
HTTP Stream Ctrl+Alt+Shift+H

Following the stream

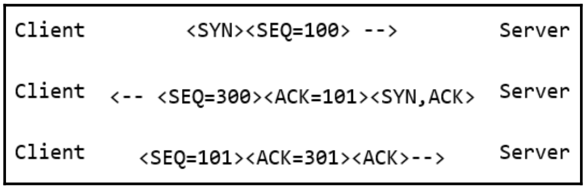
- **TCP segment length:** In the transport layer, the PDU is a segment. The segment length is the value of the TCP payload, which is the data that follows the TCP header, and any options. This value is in brackets, as it is calculated by Wireshark. In frame 5, we can see **[TCP Segment Length: 0].** This means there is no data following the header, which would make sense, as frame 5 is an acknowledgment of the data received in frame 4.

Next, we'll take a look at the fields that keep track of the data that's sent and received during data transmission.

Sequencing and acknowledging data

Because TCP is a connection-oriented protocol, the operating system keeps track of every byte (or octet) of data. Each byte is sequenced and, once received, is acknowledged. The following are the fields that help provide a snapshot of the data that's exchanged during a TCP connection:

- **Sequence number 32-bit**: The three-way handshake starts the sequencing. The **Synchronization** (**SYN**) packets found in the first two packets of the three-way handshake are responsible for synchronizing the sequence numbers that are used during the connection.
  For example, as shown in the following screenshot, a client sends a SYN packet to the server with a sequence number of `100`.
  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`:

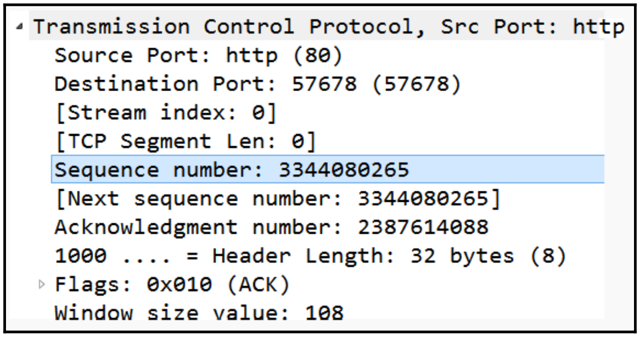


```
Client        <SYN><SEQ=100> -->          Server

Client  <-- <SEQ=300><ACK=101><SYN,ACK>   Server

Client    <SEQ=101><ACK=301><ACK>-->      Server
```

The three-way handshake

After the handshake, the data flow begins. In frame 5, we can see **Sequence number: 1 (relative sequence number).** Relative sequence numbers are generated by Wireshark, mainly because the actual sequence number is very large. The relative sequence number is easy to understand and represents a value *in relation* to this conversation.

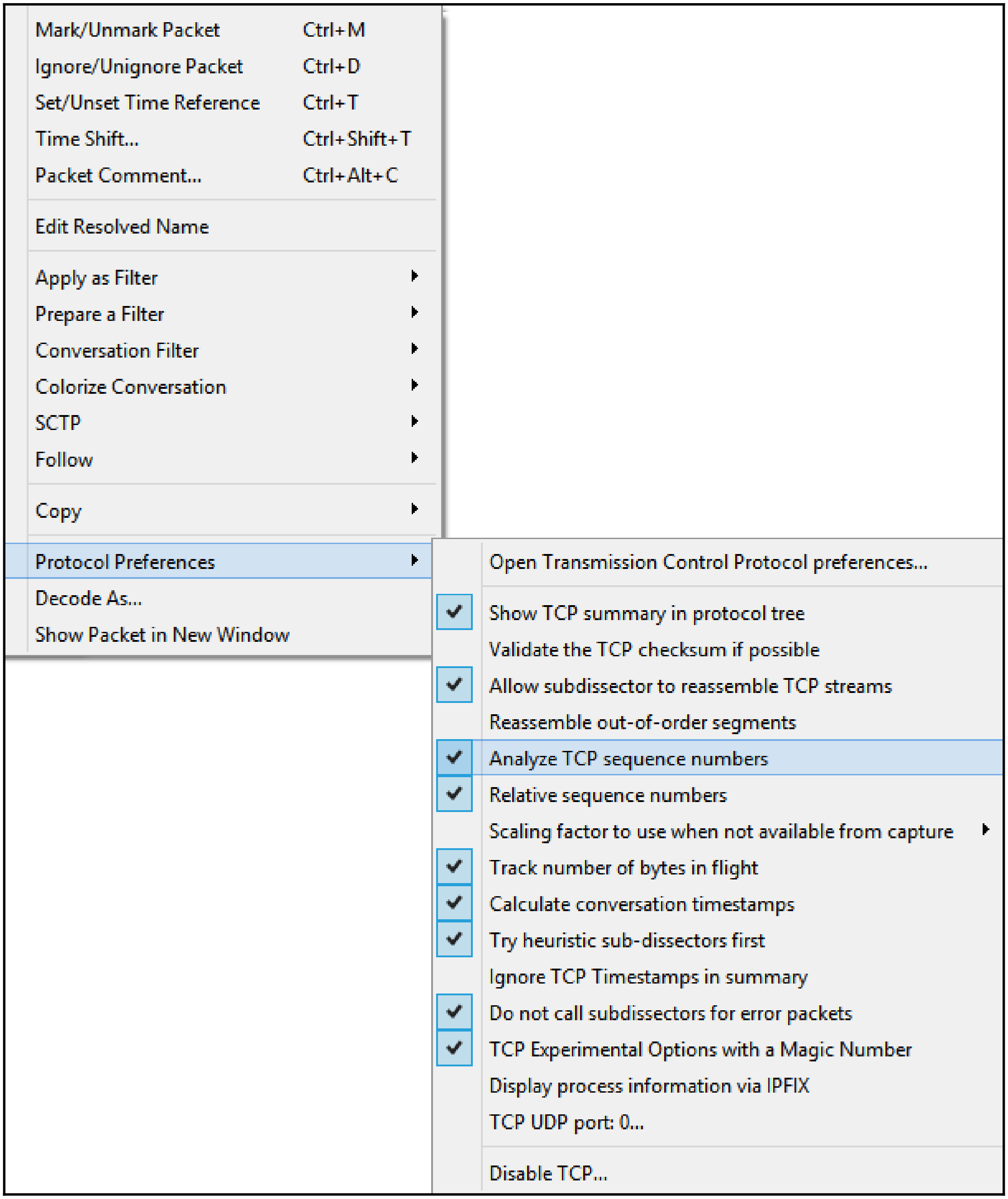
Without using a relative sequence number, the absolute sequence number is Sequence number: 3344080265, as shown in the following screenshot:



```
◂ Transmission Control Protocol, Src Port: http
    Source Port: http (80)
    Destination Port: 57678 (57678)
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 3344080265
    [Next sequence number: 3344080265]
    Acknowledgment number: 2387614088
    1000 .... = Header Length: 32 bytes (8)
  ▹ Flags: 0x010 (ACK)
    Window size value: 108
```

Absolute sequence numbers

If you would like to use relative rather than actual sequence numbers, right-click on anywhere in the TCP header, select **Protocol Preferences**, and then select **Relative Sequence Numbers**, as shown in the following screenshot:


Mark/Unmark Packet Ctrl+M
Ignore/Unignore Packet Ctrl+D
Set/Unset Time Reference Ctrl+T
Time Shift... Ctrl+Shift+T
Packet Comment... Ctrl+Alt+C
Edit Resolved Name
Apply as Filter
Prepare a Filter
Conversation Filter
Colorize Conversation
SCTP
Follow
Copy
Protocol Preferences
Decode As...
Show Packet in New Window

Open Transmission Control Protocol preferences...

✓ Show TCP summary in protocol tree

Validate the TCP checksum if possible

✓ Allow subdissector to reassemble TCP streams

Reassemble out-of-order segments

✓ Analyze TCP sequence numbers

✓ Relative sequence numbers

Scaling factor to use when not available from capture

✓ Track number of bytes in flight

✓ Calculate conversation timestamps

✓ Try heuristic sub-dissectors first

Ignore TCP Timestamps in summary

✓ Do not call subdissectors for error packets

✓ TCP Experimental Options with a Magic Number

Display process information via IPFIX

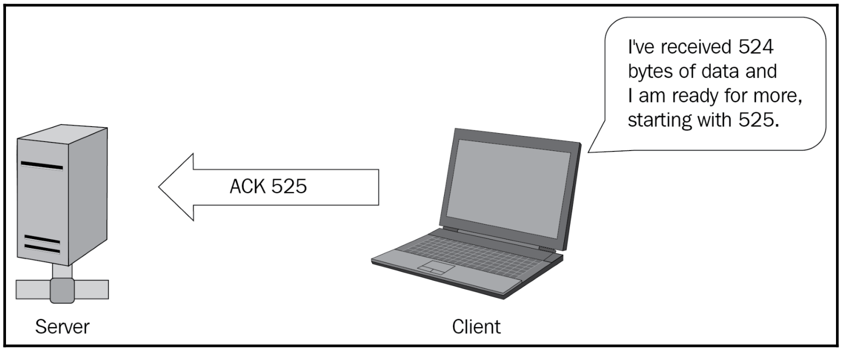
TCP UDP port: 0...

Disable TCP...

Protocol preferences—relative sequence numbers

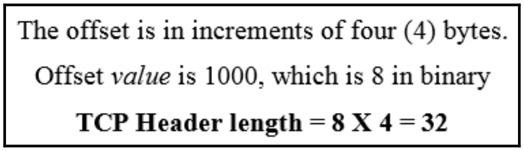
This will adjust the sequence numbers to a more understandable value. To see the use of relative sequence numbers, refer to the figure *TCP header*, where you will see **Sequence number 1**.

- **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.
- **Acknowledgment number 32-bit**: During data transfer, the operating system keeps track of all bytes and reordering by using the sequence numbers. Every time the TCP 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:


I've received 524 bytes of data and I am ready for more, starting with 525.
ACK 525
Server
Client

Acknowledging the data

- **Offset**: The line right after the **Acknowledgement number** is `1000`. This line is the data offset field, which indicates the length of the TCP header. After the TCP header, the data begins. In this case, the offset value is 32 bytes. The following diagram shows how this value is calculated:


The offset is in increments of four (4) bytes.

Offset *value* is 1000, which is 8 in binary

TCP Header length = 8 X 4 = 32

Offset value calculation

- The size of a fixed TCP header field is 20 bytes. However, many times in today's networks, the TCP header has additional options, so the value is not always consistent.

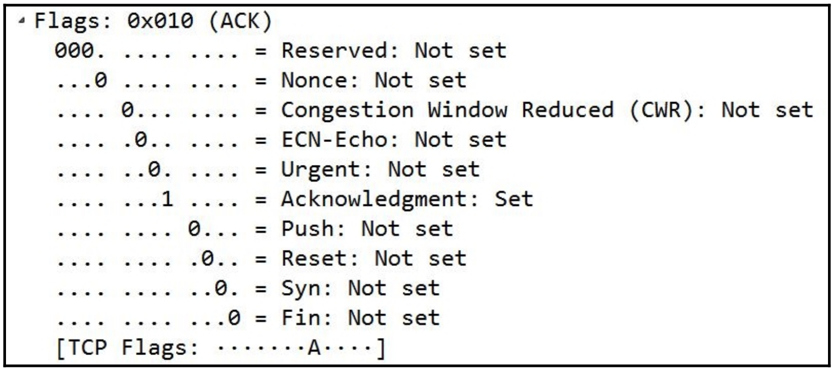
While keeping track of the data exchange, another important element in the TCP header is the use of flags, as discussed in the next section.

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 (8) control flags, as shown here:

Control flag	**Bits**	**Function**
Reserved	3	The **Reserved** flag is for future use and should be set to zero.
Nonce	1	**Nonce** is experimental—possibly use with ECN.
CWR	1	**Congestion Window Reduced** when set indicates that the sender is responding to indications of network congestion with congestion avoidance.
ECE	1	The **Explicit Congestion Notification Echo Explicit Congestion Notification** (**ECN**) 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, this means the endpoint is ECN capable.
URG	1	**Urgent** indicates a packet that should have priority. Rarely seen.
ACK	1	**Acknowledgment** acknowledges that the data was received and that the client is ready to accept more. All packets after the initial SYN packet sent by the client should have this flag set.
PSH	1	Normally, a buffer will hold data until it has a decent-sized packet to send. **Push** informs TCP that data should be sent immediately and not wait until the buffer is full.
RST	1	When set, the sender and receiver will abort the TCP connection. A **Reset** can happen for a number of reasons; many times, it is used to close an abnormal or malicious connection.
SYN	1	**Synchronization** synchronizes the sequence numbers. Only the first two packets of the handshake will have this flag set.
FIN	1	**Finish** means the communication has ended and there is no more data—close the connection.

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: 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····]
```

TCP flags

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.

As we can see, 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 window size is used to notify the sender about just how much data that 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 communicate with each other. Flow control is an end-to-end control method where a host transmits a window size, with every acknowledgement indicating how many bytes it can accept so that the sender does not transmit too much data and overwhelm the host. The following steps through values provide an indication of how much data the client can receive, which can change at any time during a conversation:

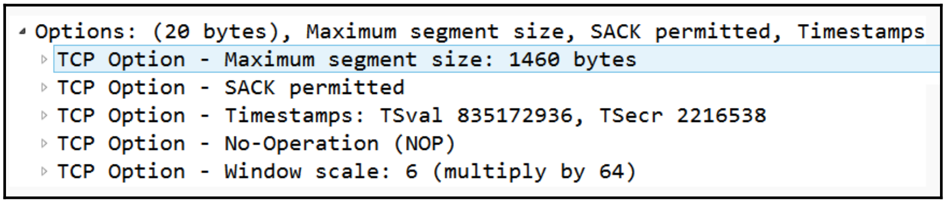
- **Window size 16-bit**: During an active connection, the server sends data to the client. The client responds with an ACK and the window size value, which will indicate how much data they can accept. In the case of frame 5, we can see **Window size value: 108.**
    - If the client cannot process all the data, the client will send an ACK with a lower Window size value.
    - Once the client advertises a smaller window size, the server throttles back the data transfer.
    - 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.
    - The server then can continue to send data.

A related term is called **sliding window**, as the value will slide back and forth as the end point adjusts the amount of traffic it can accept.

- **Calculated Window size**: In brackets, we can see **[Calculated Window Size 6912]**. This value is larger than the actual Window size because this stream uses a scaling factor, which changes the value of the Window Size. Let's talk about why this value is different:
    - 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. Over time, options were used to expand the Window size value in the TCP header.
    - In the early 1990's, RFCs were written to address the larger buffer sizes, and a window scaling option provides a way to address the actual window size.
    - As a result, Wireshark calculates the Window size by multiplying the scaling factor of 64 by the listed window size field value 108, which gives us a calculated window size of 6,912.

Next, let's take a look at how the scaling factor is determined.

- **Window size scaling factor:** The `[Window Size scaling factor: 64]` is calculated by Wireshark. This reflects the scaling factor from the TCP options exchanged during the three-way handshake. As shown in the following screenshot, we can see that the TCP options sent by the server lists the last option as the **Window scale is 6 (multiply by 64)**:



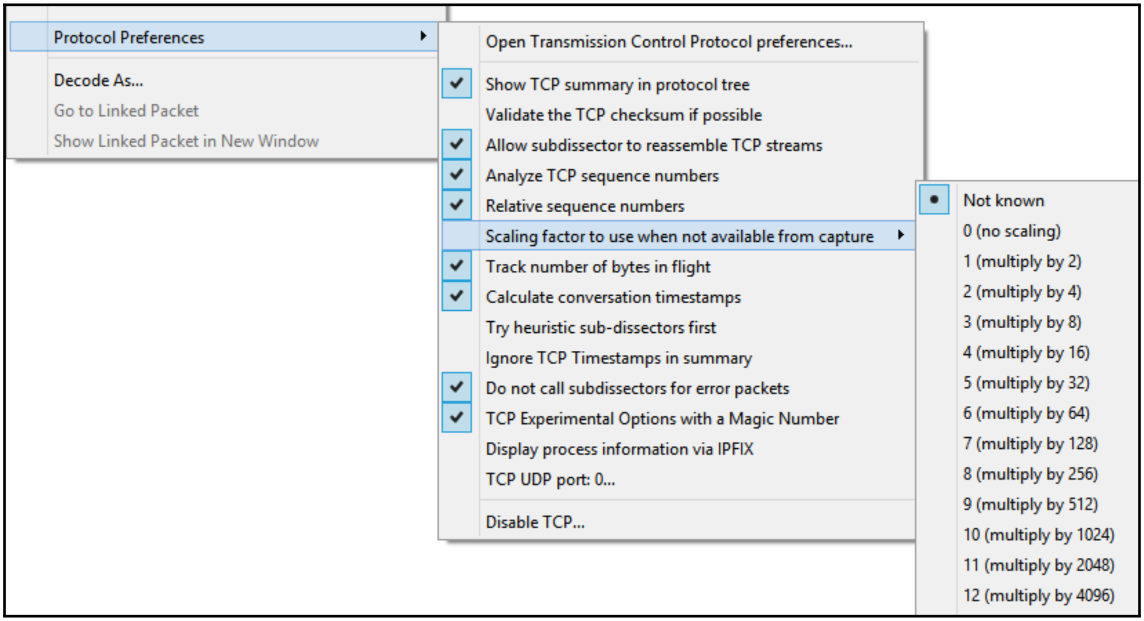
```
◢ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps
  ▷ TCP Option - Maximum segment size: 1460 bytes
  ▷ TCP Option - SACK permitted
  ▷ TCP Option - Timestamps: TSval 835172936, TSecr 2216538
  ▷ TCP Option - No-Operation (NOP)
  ▷ TCP Option - Window scale: 6 (multiply by 64)
```

TCP options

Wireshark will determine the calculated window size 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 know the scaling factor, you can modify the value by right-clicking anywhere in the TCP header, selecting **Protocol Preferences** | **Scaling factor to use when not available from capture**, and then selecting the appropriate value, as shown in the following screenshot:



Protocol Preferences ▸

Decode As...

Go to Linked Packet

Show Linked Packet in New Window

Open Transmission Control Protocol preferences...

☑ Show TCP summary in protocol tree

☐ Validate the TCP checksum if possible

☑ Allow subdissector to reassemble TCP streams

☑ Analyze TCP sequence numbers

☑ Relative sequence numbers

Scaling factor to use when not available from capture ▸

☑ Track number of bytes in flight

☑ Calculate conversation timestamps

☐ Try heuristic sub-dissectors first

☐ Ignore TCP Timestamps in summary

☑ Do not call subdissectors for error packets

☑ TCP Experimental Options with a Magic Number

☐ Display process information via IPFIX

TCP UDP port: 0...

Disable TCP...

● Not known

○ 0 (no scaling)

○ 1 (multiply by 2)

○ 2 (multiply by 4)

○ 3 (multiply by 8)

○ 4 (multiply by 16)

○ 5 (multiply by 32)

○ 6 (multiply by 64)

○ 7 (multiply by 128)

○ 8 (multiply by 256)

○ 9 (multiply by 512)

○ 10 (multiply by 1024)

○ 11 (multiply by 2048)

○ 12 (multiply by 4096)

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.

Additional header values

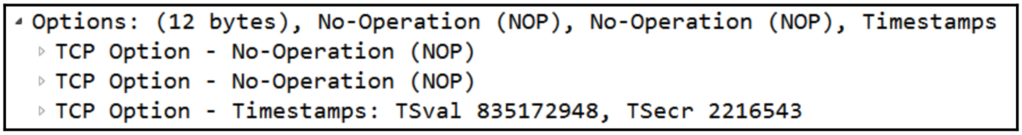
The last part of the header lists additional values and options. The following are the field values and information that helps keep a TCP connection on track:

- **Checksum and Status**: 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 5, we can see the value of the checksum as **Checksum: 0x82e4 [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**.

- **Urgent pointer**: The TCP flags, which include the **urgent** (**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 **push** (**PSH**), as it informs the TCP that data should be sent up the stack immediately.
- **Options**: Before any TCP conversation, there is a three-way handshake. During the handshake, TCP has several options. The options will be listed during the SYN packet exchange. In frame 5, the options include timestamps and no-operation, as shown in the following screenshot:

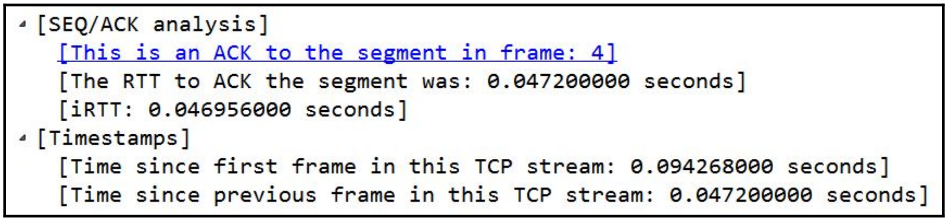


```
◢ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▷ TCP Option - No-Operation (NOP)
  ▷ TCP Option - No-Operation (NOP)
  ▷ TCP Option - Timestamps: TSval 835172948, TSecr 2216543
```

TCP header options

These options and others will be covered in detail in `Chapter 10`, Managing TCP Connections.

- **Additional Packet Details:** At the bottom of the TCP header, there are two other calculations: `[SEQ/ACK analysis]` and `[Timestamps]`, as shown in the following screenshot:



```
◢ [SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 4]
    [The RTT to ACK the segment was: 0.047200000 seconds]
    [iRTT: 0.046956000 seconds]
◢ [Timestamps]
    [Time since first frame in this TCP stream: 0.094268000 seconds]
    [Time since previous frame in this TCP stream: 0.047200000 seconds]
```

Additional Packet Details

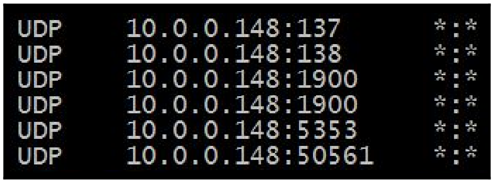
- `[SEQ/ACK analysis]` is a calculated field that includes information such as what frame was acknowledged and the **Round-Trip Time** (**RTT**), which is used in function such as the time-sequence graphs found under **Statistics** | **TCP Stream Graphs**. The `[Timestamps]` calculation indicates the elapsed time and is used to provide details about the capture found in **Statistics** | **Capture File Properties**.

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 the **User Datagram Protocol** (**UDP**), a transport protocol to use when speed—not reliability—is required for data transport.

Understanding UDP

UDP is a lightweight, connectionless, TCP used for data transfer. UDP does not have a handshake or connection process, nor does it have a teardown.

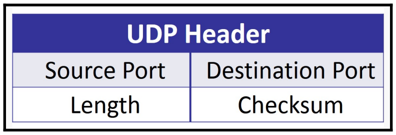
To see all your active UDP connections on a Windows machine, open a command line and run **netstat -anp udp**, as shown in the following screenshot:



```
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      *:*
```

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 sender (or foreign) IP address and port. As a result, as shown in the following diagram, you will see only a *local* IP address and port for UDP:



UDP Header	
Source Port	Destination Port
Length	Checksum

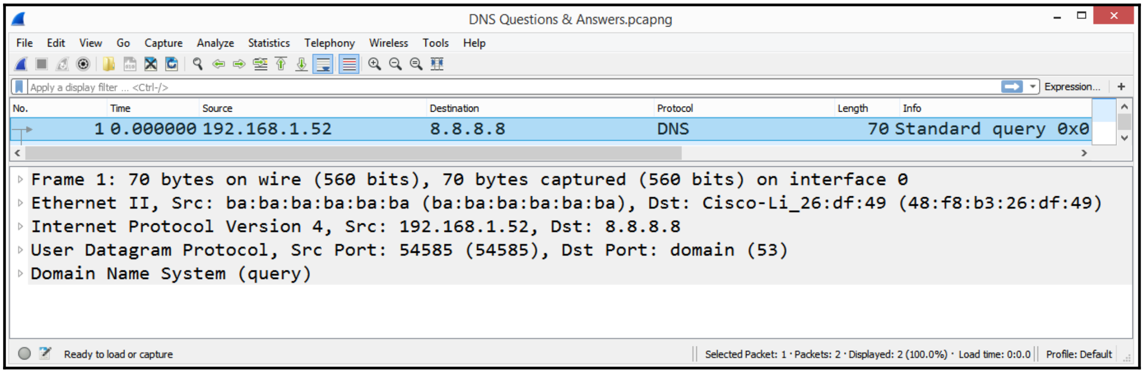
The UDP header

Because of UDP's streamlined nature, it is an appropriate protocol for time-sensitive applications such as **Dynamic Host Configuration Protocol** (**DHCP**), **Domain Name System** (**DNS**), **Trivial File Transfer Protocol** (**TFTP**), **Voice over IP** (**VoIP**), and other protocols that require speed. Let's take a look at how a single UDP frame works.

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 one (1) so you can follow along. Once selected, you will see the following details:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.52	8.8.8.8	DNS	70	Standard query 0x0

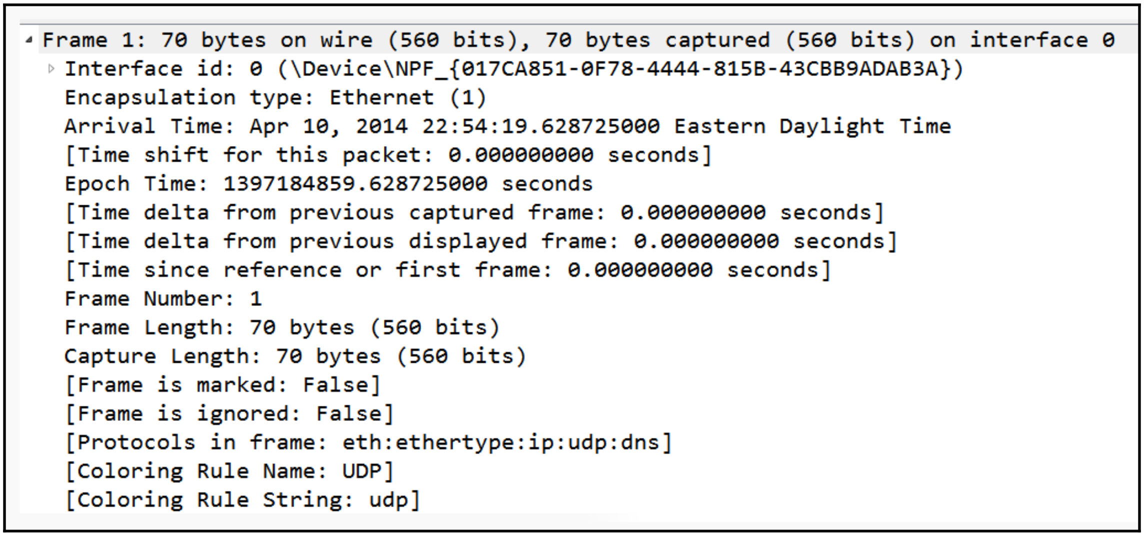
```
▹ Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
▹ Ethernet II, Src: ba:ba:ba:ba:ba:ba (ba:ba:ba:ba:ba:ba), Dst: Cisco-Li_26:df:49 (48:f8:b3:26:df:49)
▹ Internet Protocol Version 4, Src: 192.168.1.52, Dst: 8.8.8.8
▹ User Datagram Protocol, Src Port: 54585 (54585), Dst Port: domain (53)
▹ Domain Name System (query)
```

Ready to load or capture

Selected Packet: 1 · Packets: 2 · Displayed: 2 (100.0%) · Load time: 0:0.0 | Profile: Default

Frame one (1), as shown in the packet details pane

Starting from the top, Wireshark lists the contents of this single frame. Expand the frame metadata by clicking on the arrow (or caret, >) on the right-hand side to see the details, as shown in the following screenshot:



```
◢ Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
  ▷ Interface id: 0 (\Device\NPF_{017CA851-0F78-4444-815B-43CBB9ADAB3A})
    Encapsulation type: Ethernet (1)
    Arrival Time: Apr 10, 2014 22:54:19.628725000 Eastern Daylight Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1397184859.628725000 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 70 bytes (560 bits)
    Capture Length: 70 bytes (560 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:udp:dns]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
```

Frame metadata on a single frame—UDP

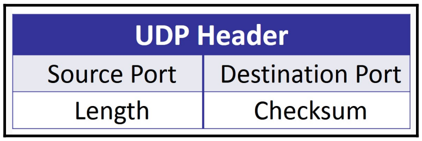
In frame 1, we can see the following:

- **Frame 1**: Frame is not a protocol. **Frame** is a list of values 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. It is the same as the information found in the section A single TCP frame.
- **Internet Protocol Version 4**: Provides the details of the IP protocol used and includes the source and destination IP address. It is the same as the information found in the section A single TCP frame.
- **User Datagram Protocol**: Provides the details of the UDP header as shown in the figure *The UDP header*. When looking at the UDP header contents, any information in brackets is generated by Wireshark to help you better understand the details of the header, such as **Checksum Status** and **Stream index**.

Now that we have taken a look at the information that's found 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:



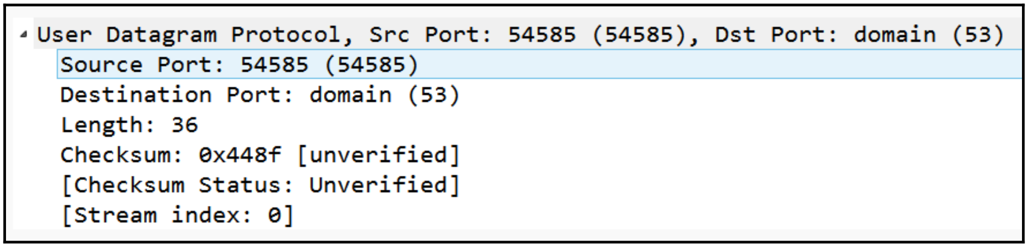
UDP Header	
Source Port	Destination Port
Length	Checksum

The UDP header

UDP is always eight bytes long as it does not have any header options. UDP is connectionless; 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. 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. Unlike TCP, UDP has a simple header, with no additional communication details listed, such as **Timestamps** or **SEQ/ACK analysis**:



```
◂ 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]
```

The UDP header

Starting at the top of the UDP header, we can see the UDP, followed by a summary of what the header represents with information on the source and destination ports:

- **Source Port 16-bit**: The source port field is the port on the receiver's side, and this field. In this case, the sender is a DNS client, as the source port in frame 1 is **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 sender's side, and this field is 16bit. In this case, the port in frame 1 is **Destination Port: domain (53)**. Port `53` is associated with DNS, and is the port on the DNS server accepting resolution 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. The UDP checksum is optional in IPv4; however, it is required in IPv6. The checksum is required in IPv6 primarily because IPv6 does not have a checksum, and the value in the UDP header is used to ensure data integrity.

Summary

Anyone working on a network should have a solid understanding of the protocols that traverse the network. Although there are hundreds of protocols, this chapter focused on the functions of the transport layer of the OSI model, specifically the two predominant protocols, TCP and UDP.

We evaluated TCP, a connection-oriented protocol. You now understand that, in order to achieve reliability, TCP sequences and acknowledges every octet. We saw that, in addition to transporting data, TCP monitors the transmission and provides not only flow control, but congestion control as well. So that you grasped how TCP is so effective in providing reliable data transport, we took a closer look at the header, along with the eight control flags.

Along with TCP, we looked at the other transport layer protocol, UDP, which ensures fast transportation of time-sensitive data and protocols such as DHCP and DNS. We discovered 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 and examine the TCP three-way handshake and resultant socket creation. We will look at the packet exchange and have a closer look at the TCP options. Finally, we'll study 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*:

1. A _____ is defined as an IP address and a port.
    1. Window
    2. Socket
    3. Checksum
    4. Sequence
2. ACK 725 means I have received _____ bytes of data and I am ready for more, starting with _____.
    1. 700 and 800
    2. 724 and 725
    3. 725 and 726
    4. 725 and 725
3. The _____ flag informs TCP that data should be sent immediately.
    1. RST
    2. SYN
    3. ACK
    4. PSH
4. If the offset value is 0101, the TCP header length = ___.
    1. 32
    2. 8
    3. 20
    4. 64
5. In a normal connection, ___ will use UDP.
    1. HTTP
    2. SMTP
    3. DHCP
    4. FTP

10
Managing TCP Connections

One of the most important, yet least understood, 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 **Finish (FIN)** packets, that indicates the session is complete.

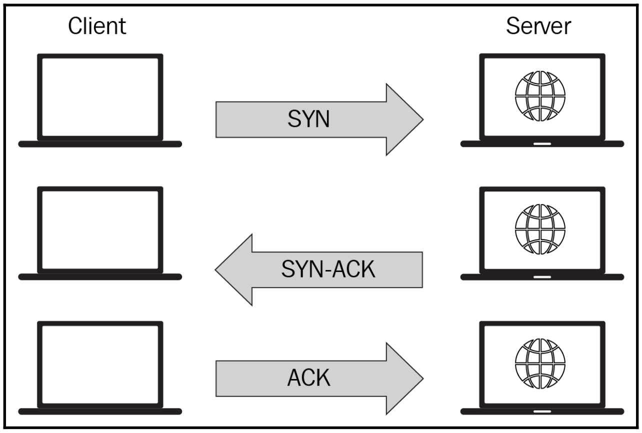
In this chapter we'll take a more detailed look at the handshake and resultant socket creation. So that you can home in on a single TCP stream, we'll take a large capture, subset, mark and filter the packets, so we can examine the TCP handshake. As you traverse the chapter, you'll have a greater understanding of the TCP options exchanged during the handshake. You'll learn what they mean and why they are required to have a conversation on today's networks. In addition, you'll see how you can easily modify protocol preferences, such as analyze 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

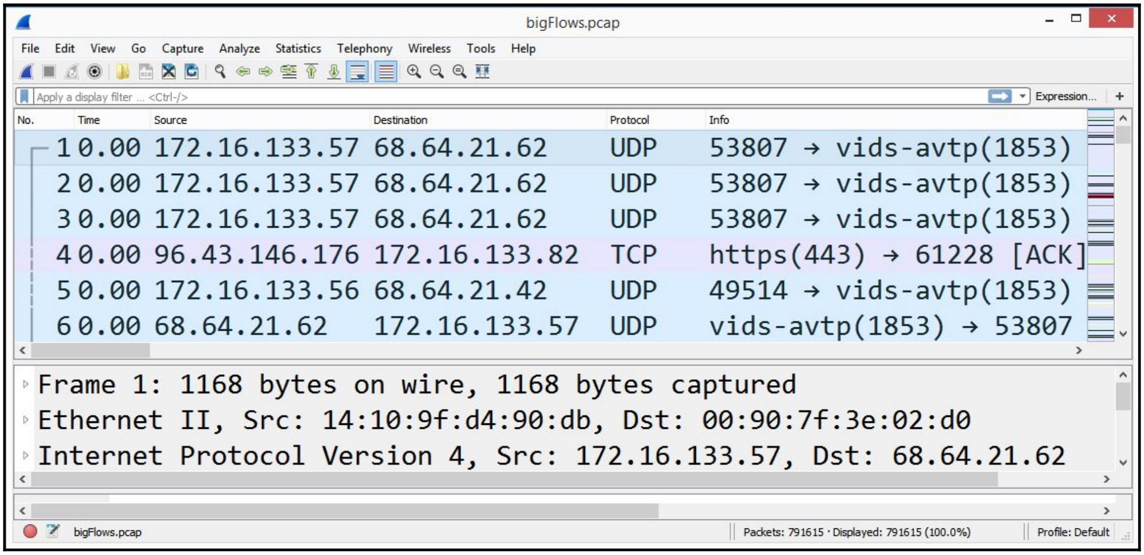
In computing, a handshake is an exchange of information between devices that sets up the parameters of the conversation. Each side sends what is available and the two endpoints agree on the terms before any data is exchanged. The topic of the three-way handshake is outlined in detail in the original TCP RFC 793 found at `https://tools.ietf.org/html/rfc793`. The TCP handshake is as follows:



The TCP three-way handshake

In most cases, the client initiates the conversation with a **synchronization** (**SYN**) packet, the server responds with a **synchronization acknowledgment** (**SYN-ACK**), and the client then completes the handshake with an **acknowledgment** (**ACK**). 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.cap` so you can follow along. Bigflows is a large capture that has many protocols and conversations. Bigflows has 791,615 packets, as shown in the lower right-hand corner of the following screenshot:


bigFlows.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

No.	Time	Source	Destination	Protocol	Info
1	0.00	172.16.133.57	68.64.21.62	UDP	53807 → vids-avtp(1853)
2	0.00	172.16.133.57	68.64.21.62	UDP	53807 → vids-avtp(1853)
3	0.00	172.16.133.57	68.64.21.62	UDP	53807 → vids-avtp(1853)
4	0.00	96.43.146.176	172.16.133.82	TCP	https(443) → 61228 [ACK]
5	0.00	172.16.133.56	68.64.21.42	UDP	49514 → vids-avtp(1853)
6	0.00	68.64.21.62	172.16.133.57	UDP	vids-avtp(1853) → 53807

```
Frame 1: 1168 bytes on wire, 1168 bytes captured
Ethernet II, Src: 14:10:9f:d4:90:db, Dst: 00:90:7f:3e:02:d0
Internet Protocol Version 4, Src: 172.16.133.57, Dst: 68.64.21.62
```

bigFlows.pcap | Packets: 791615 · Displayed: 791615 (100.0%) | Profile: Default

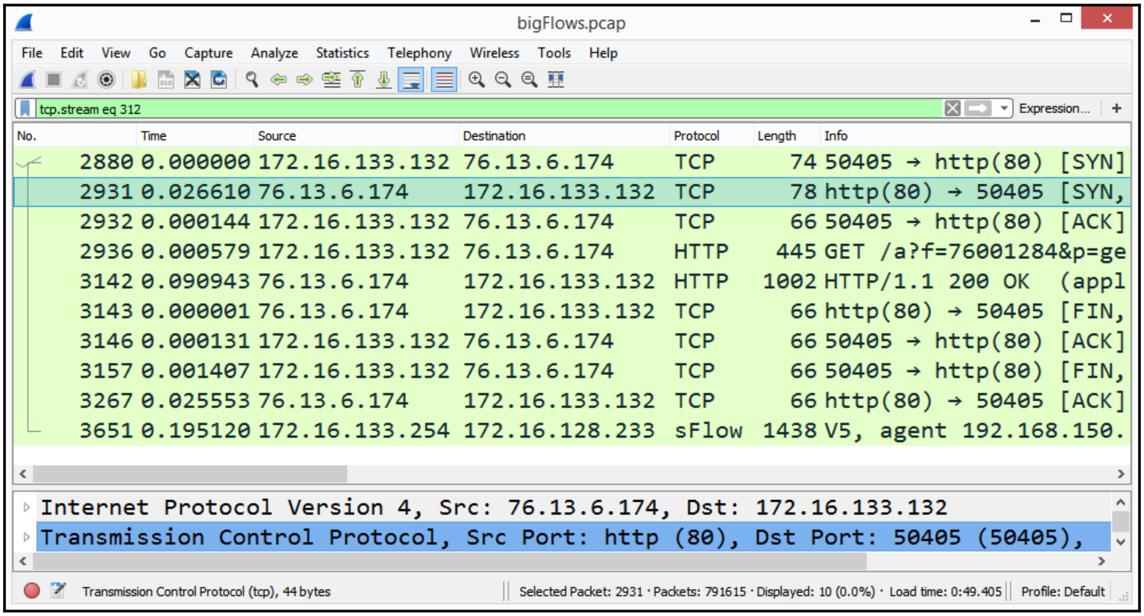
BigFlows

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 UDP streams. Although we can filter on any of the streams, for now, let's use 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:



tcp.stream eq 312

No.	Time	Source	Destination	Protocol	Length	Info
2880	0.000000	172.16.133.132	76.13.6.174	TCP	74	50405 → http(80) [SYN]
2931	0.026610	76.13.6.174	172.16.133.132	TCP	78	http(80) → 50405 [SYN,
2932	0.000144	172.16.133.132	76.13.6.174	TCP	66	50405 → http(80) [ACK]
2936	0.000579	172.16.133.132	76.13.6.174	HTTP	445	GET /a?f=76001284&p=ge
3142	0.090943	76.13.6.174	172.16.133.132	HTTP	1002	HTTP/1.1 200 OK (appl
3143	0.000001	76.13.6.174	172.16.133.132	TCP	66	http(80) → 50405 [FIN,
3146	0.000131	172.16.133.132	76.13.6.174	TCP	66	50405 → http(80) [ACK]
3157	0.001407	172.16.133.132	76.13.6.174	TCP	66	50405 → http(80) [FIN,
3267	0.025553	76.13.6.174	172.16.133.132	TCP	66	http(80) → 50405 [ACK]
3651	0.195120	172.16.133.254	172.16.128.233	sFlow	1438	V5, agent 192.168.150.

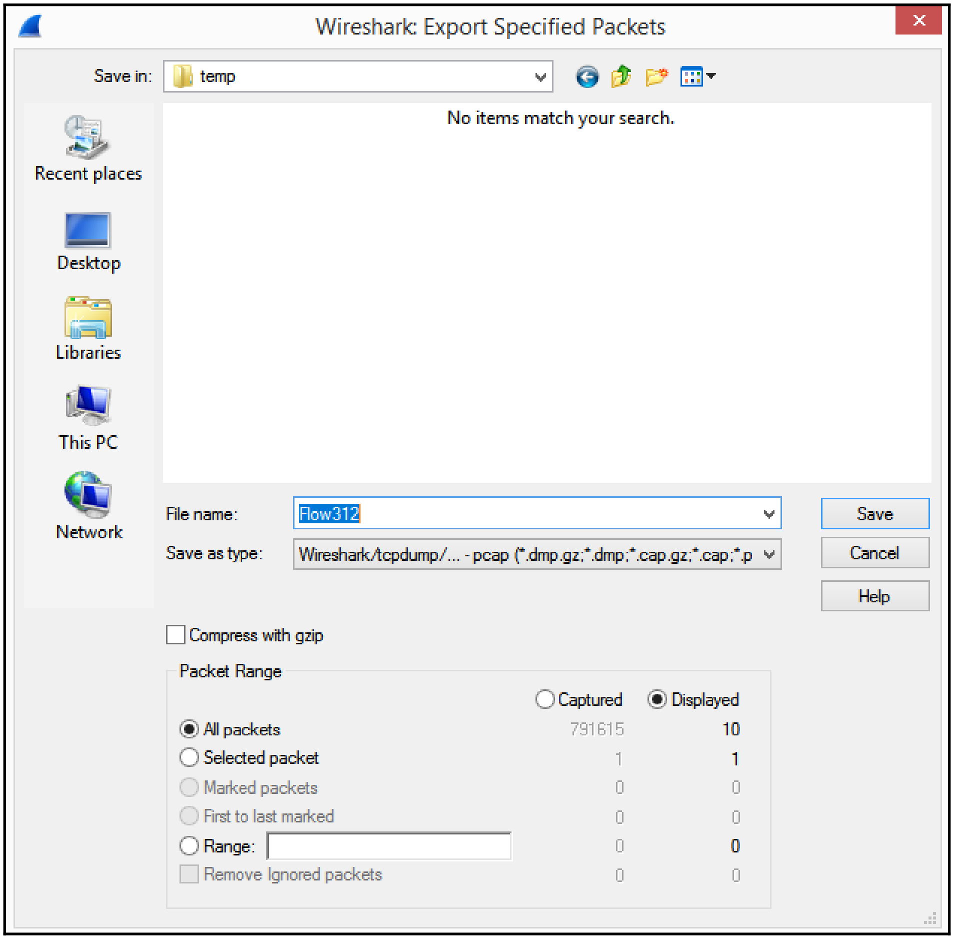
```
▷ Internet Protocol Version 4, Src: 76.13.6.174, Dst: 172.16.133.132
▷ Transmission Control Protocol, Src Port: http (80), Dst Port: 50405 (50405),
```

Transmission Control Protocol (tcp), 44 bytes || Selected Packet: 2931 · Packets: 791615 · Displayed: 10 (0.0%) · Load time: 0:49.405 || Profile: Default

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 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 graphic:



Wireshark: Export Specified Packets

Save in: temp

No items match your search.

File name: Flow312

Save as type: Wireshark/tcpdump/... - pcap (*.dmp.gz;*.dmp;*.cap.gz;*.cap;*.p

□ Compress with gzip

Packet Range

	○ Captured	● Displayed
● All packets	791615	10
○ Selected packet	1	1
○ Marked packets	0	0
○ First to last marked	0	0
○ Range:	0	0
□ Remove Ignored packets	0	0

Select All packets and Displayed

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. When we mark the packets, Wireshark will modify the packet to have a black background with white text. Once we mark them, we'll filter according to the marked packets to focus in on the handshake.

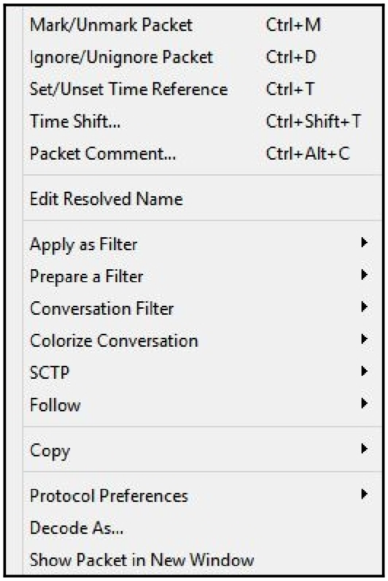
In the file, we'll identify the handshake by marking the packets. We know that to begin a session, TCP starts with a handshake that uses three packets as follows:

- The client sends a SYN packet to the server.
- The server responds by sending a SYN ACK packet.
- The client sends a final ACK packet.

Once the handshake is complete, the data flow begins.

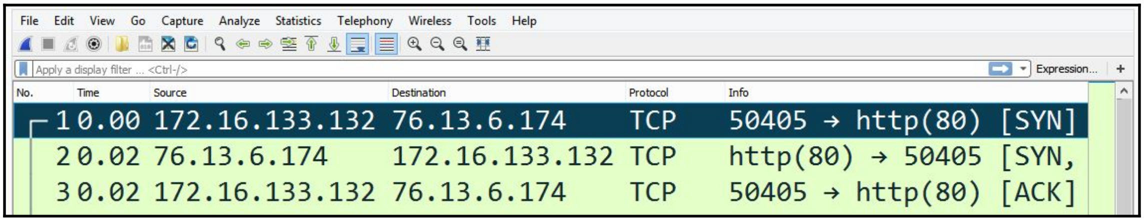
Wireshark will identify the three-way handshake and the exchange of packets by showing the transaction details in the info column, (if you have this column header active). In the capture `Flow312.pcapng`, packets 1, 2, and 3 represents handshake.

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 sub menu choice **Mark/Unmark Packet** as shown in the graphic:


Mark/Unmark Packet Ctrl+M
Ignore/Unignore Packet Ctrl+D
Set/Unset Time Reference Ctrl+T
Time Shift... Ctrl+Shift+T
Packet Comment... Ctrl+Alt+C
Edit Resolved Name
Apply as Filter
Prepare a Filter
Conversation Filter
Colorize Conversation
SCTP
Follow
Copy
Protocol Preferences
Decode As...
Show Packet in New Window

Mark/Unmark Packet

Once you have marked the packet, the background will turn black and the text will be white, as shown here:



File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

No.	Time	Source	Destination	Protocol	Info
1	0.00	172.16.133.132	76.13.6.174	TCP	50405 → http(80) [SYN]
2	0.02	76.13.6.174	172.16.133.132	TCP	http(80) → 50405 [SYN,
3	0.02	172.16.133.132	76.13.6.174	TCP	50405 → http(80) [ACK]

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.

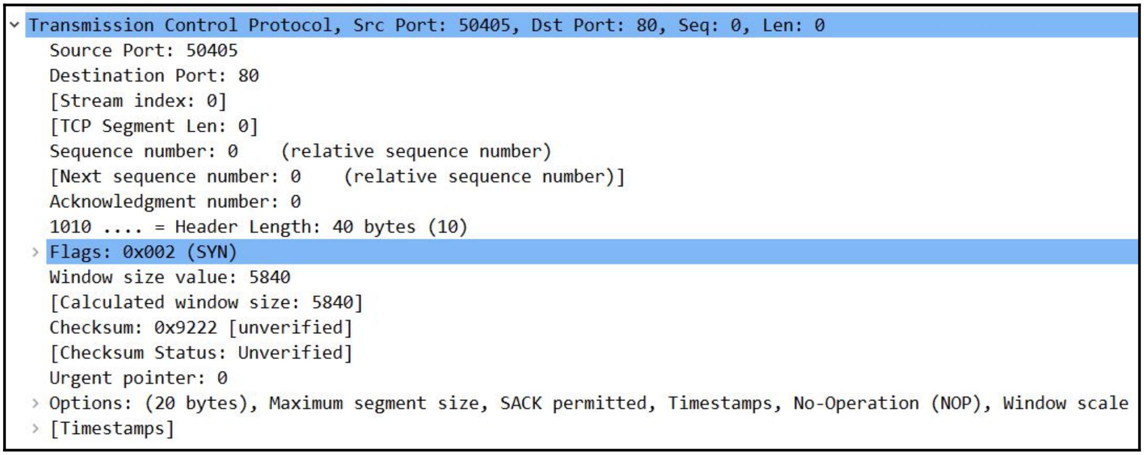
Identifying the handshake packets

In this section, we'll take a look at each of the packets, examine the flags, along with the sequence and acknowledgement 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 **Synchronization** (**SYN**) packet to the server and then wait for a response.

To see all the field values as shown, go to frame 1, and expand the TCP Header as shown:



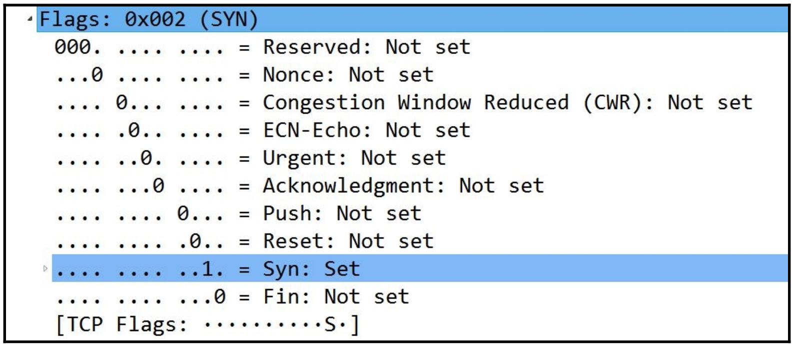
```
Transmission Control Protocol, Src Port: 50405, Dst Port: 80, Seq: 0, Len: 0
  Source Port: 50405
  Destination Port: 80
  [Stream index: 0]
  [TCP Segment Len: 0]
  Sequence number: 0    (relative sequence number)
  [Next sequence number: 0    (relative sequence number)]
  Acknowledgment number: 0
  1010 .... = Header Length: 40 bytes (10)
  Flags: 0x002 (SYN)
  Window size value: 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), Window scale
  [Timestamps]
```

Flows312.pcapng—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 acknowledgment numbers, as follows:

- `Sequence number: 0 (relative sequence number)`
- `Acknowledgment number: 0`

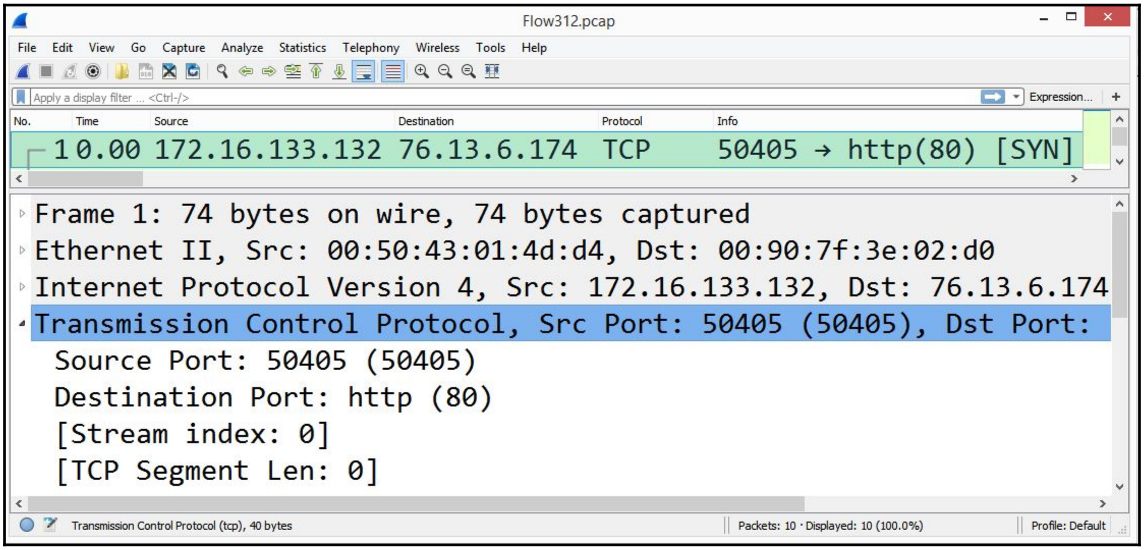
If we expand the flags, we see that the `Syn` flag is set as shown in the following screenshot. Keep in mind that the `Syn` flag will *only* be used to synchronize the sequence numbers during the first two packets of the handshake:



```
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·]
```

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:


Flow312.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No. Time  Source          Destination  Protocol  Info
1   0.00  172.16.133.132  76.13.6.174  TCP       50405 → http(80) [SYN]
```

```
▹ Frame 1: 74 bytes on wire, 74 bytes captured
▹ Ethernet II, Src: 00:50:43:01:4d:d4, Dst: 00:90:7f:3e:02:d0
▹ Internet Protocol Version 4, Src: 172.16.133.132, Dst: 76.13.6.174
◢ Transmission Control Protocol, Src Port: 50405 (50405), Dst Port:
    Source Port: 50405 (50405)
    Destination Port: http (80)
    [Stream index: 0]
    [TCP Segment Len: 0]
```

Transmission Control Protocol (tcp), 40 bytes | Packets: 10 · Displayed: 10 (100.0%) | Profile: Default

TCP header—40 bytes

While a normal TCP header is 20 bytes, this header is 40 bytes. The header size is larger because it contains options that are added to the header. 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 as well if 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` once the handshake completes. After the handshake, the data flow begins.

Now that we have seen the first packet of the three-way handshake, let's examine the second packet, the SYN-ACK.

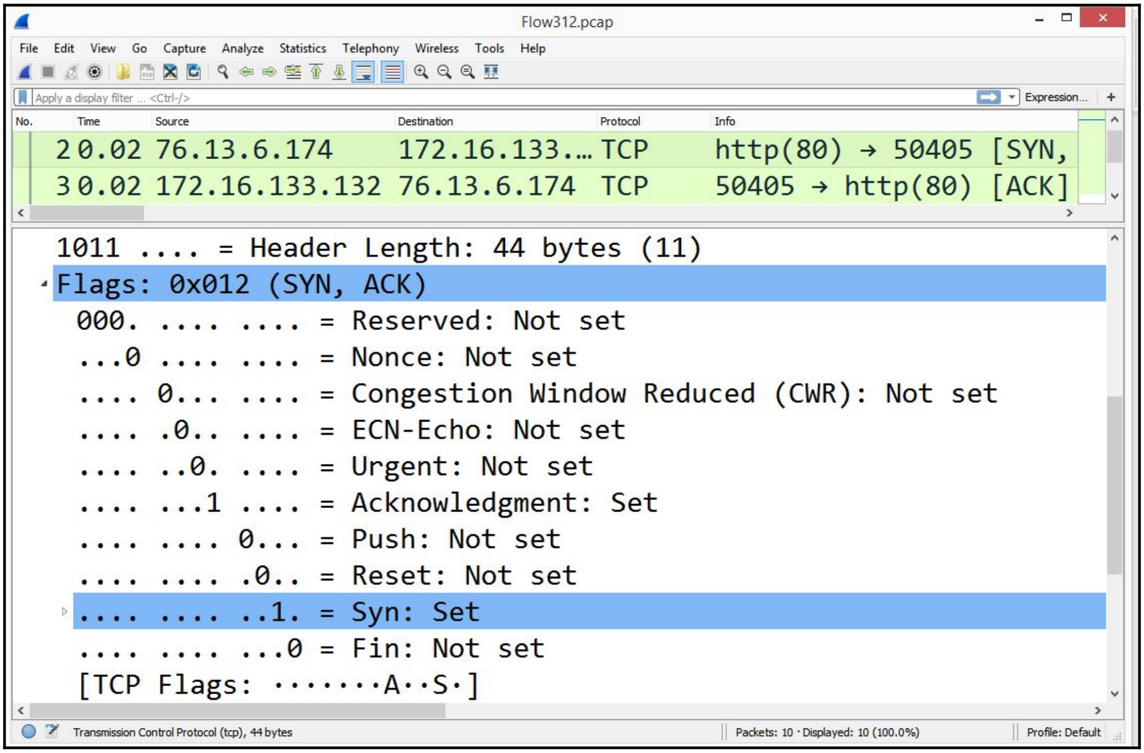
Returning the SYN-ACK packet

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, which are similar, although the acknowledgment number has changed. The sequence and acknowledgment 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:



Flow312.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No. Time Source         Destination     Protocol Info
2   0.02 76.13.6.174    172.16.133.…    TCP      http(80) → 50405 [SYN,
3   0.02 172.16.133.132 76.13.6.174     TCP      50405 → http(80) [ACK]
```

```
  1011 .... = Header Length: 44 bytes (11)
  Flags: 0x012 (SYN, 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
   .... .... ..1. = Syn: Set
   .... .... ...0 = Fin: Not set
   [TCP Flags: ·······A··S·]
```

Transmission Control Protocol (tcp), 44 bytes
Packets: 10 · Displayed: 10 (100.0%)
Profile: Default

TCP SYN-ACK flags set

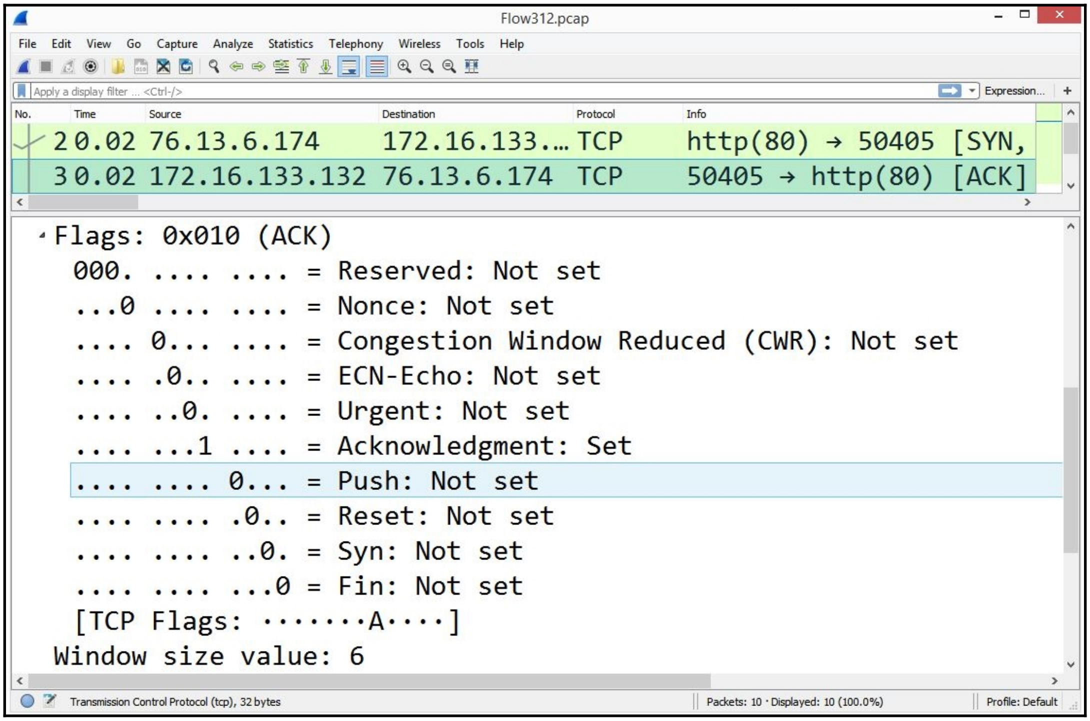
Before any data is exchanged, the handshake must complete with the acknowledgment packet, as discussed next.

Finalizing with an ACK packet

Frame 3 is the final packet in the three-way handshake. At this point, the sequence and acknowledgment 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:


Flow312.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
+

```
No. Time Source          Destination    Protocol Info
2   0.02 76.13.6.174     172.16.133.…   TCP      http(80) → 50405 [SYN,
3   0.02 172.16.133.132  76.13.6.174    TCP      50405 → http(80) [ACK]
```

```
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····]
Window size value: 6
```

Transmission Control Protocol (tcp), 32 bytes | Packets: 10 · Displayed: 10 (100.0%) | Profile: Default

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 this TCP conversation.

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 January 2019, can be found at `https://www.iana.org/assignments/tcp-parameters/tcp-parameters.txt`.

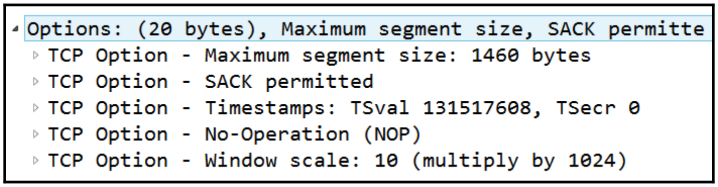
Not all options are used. Some of the options are experimental, some are used for specific reasons and do not have an associated RFC, and some have been developed and used without proper IANA assignment. The seven most common options are listed in the following table:

Kind	Length	Meaning	Reference
0	1	**End of Option List** (**EOL**)	`[RFC793]`
1	1	**No-Operation** (**NOP**)	`[RFC793]`
2	4	**Maximum segment size** (**MSS**)	`[RFC793]`
3	3	Window scale	`[RFC7323]`
4	2	**Selective acknowledgment (SACK)** permitted	`[RFC2018]`
5	N	SACK	`[RFC2018]`
8	10	Timestamps	`[RFC7323]`

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-bit, or 1 byte. The entire header must be a multiple of 32-bit, or four bytes, for memory alignment. Therefore, in some cases, padding is required to ensure that the header is a multiple of four bytes. The entire TCP header can be up to 40 bytes.

To see the TCP options for `Flow312.pcap`, select the options header in frame 1, where additional conversation parameters are listed, as shown here:



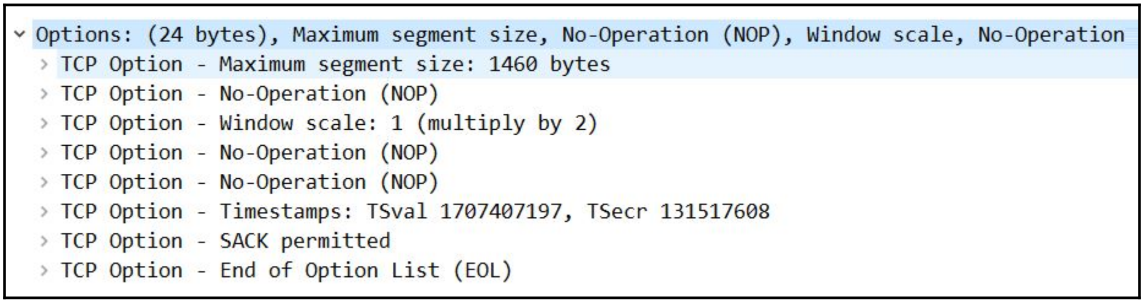
```
◢ Options: (20 bytes), Maximum segment size, SACK permitte
  ▷ TCP Option - Maximum segment size: 1460 bytes
  ▷ TCP Option - SACK permitted
  ▷ TCP Option - Timestamps: TSval 131517608, TSecr 0
  ▷ TCP Option - No-Operation (NOP)
  ▷ TCP Option - Window scale: 10 (multiply by 1024)
```

Flows312 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 End of Option List.

Grasping the EOL

EOL is a single byte used at the end of the options. 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)
```

Flows312 frame 2 options list

Another TCP option is NOP, which isn't an option at all, but a placeholder, as we'll see next.

Using NOP

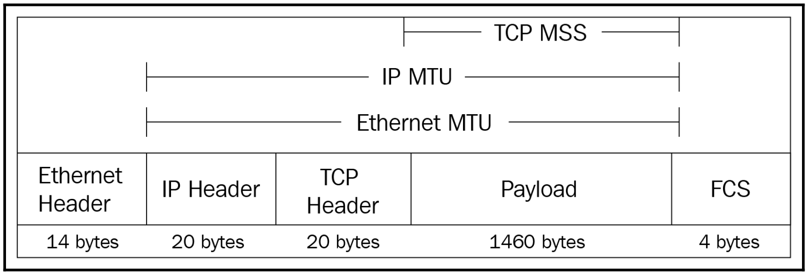
NOP is essentially a placeholder to separate different options. How and where NOP is used is dependent on the operating system. For example, as shown in the preceding screenshot, we can see that two NOPs are placed between the **Window scale** and **Timestamps** options.

In addition to using NOP to separate the various options, NOP is used to ensure that the options header is a multiple of 32-bit, or four bytes, for memory alignment. As a result, if there is an option with three bytes, such as a window scale, a single one-byte NOP will be added so that the option's length totals four 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

MSS is an option that defines the maximum receive segment size. This value is important for several reasons:



MSS and MTU

During a conversation between endpoints, TCP monitors the connection to ensure that the optimal data is sent, so as not to waste bandwidth. In addition, TCP keeps track of the following:

- **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.
- **Maximum transfer unit** (**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 several values, including the WS, CWND, and the MTU, 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, WS.

Scaling the window size

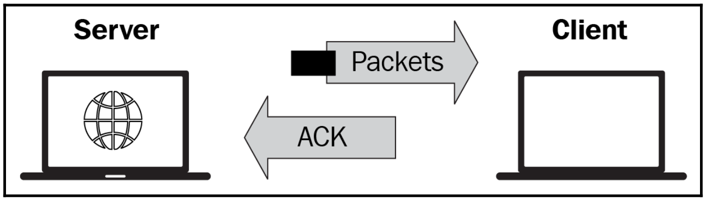
TCP is a full-duplex communication protocol, in which the sender and receiver communicate with each other. Flow control is an end-to-end control method where a host transmits a **window size** with every acknowledgment, indicating how many bytes it can accept, so the sender does not transmit too much data and overwhelm the host.

Window scale (WS) is an option that allows the window size to be expanded, according to a scaling factor that is obtained from the TCP options exchanged during the three-way handshake. The WS value is used to increase the maximum window size that is allowed. Although optional, this provides information to the server of a more accurate Window Size value.

Let's outline why this is an important option. In the TCP header, the Windows field value is 16-bit, which permits a maximum size of 65,535 bytes. The original RFC for TCP was written in 1981. In the early 1980s, 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.

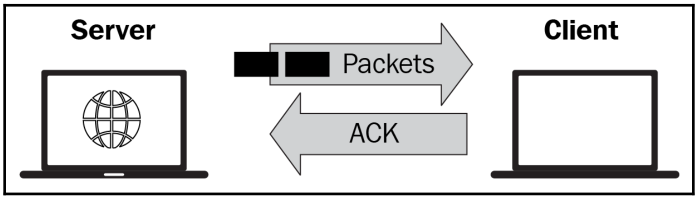
When using the Window scale option, the total value can be up to 2^{30}. This value as a metric is beneficial when data travels, especially on high-bandwidth WAN links, because a larger WS will improve performance. Intermediary devices can be tuned to accept larger WS; for example, when configuring a 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:



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 and utilize the available bandwidth for a more efficient data transfer:



A larger WS allows the server to send more data

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 protocols. One such improvement is **Selective Acknowledgment** (**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 complete transfer of all data required for that session.

After the server sends data to the client, the client will acknowledge that the data was received and is ready to accept more by sending an ACK to the server with 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 gap may be for the following reasons:

- Some of the data did not arrive.
- The ACK did not reach the server.

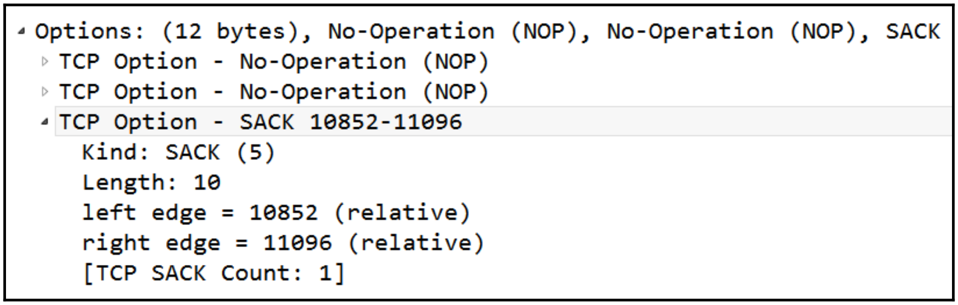
The server has no idea why there is a gap in transmission and will resend all the data that it believes is missing. It may be because the client did not receive the data. However, the gap may be because the client received the data, but the server did not receive the acknowledgment. This can lead to unnecessary retransmission of data.

By using SACK, the client selectively acknowledges what data it has received, so the server only has to resend the 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, in the `Bigflows.pcap` capture, in frame 4006, the client used the **TCP option SACK 10852-11096** option. Therefore, the server only needs to send the data from sequence number 10852 to 11096:



```
◢ 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]
```

TCP SACK option in Bigflows.pcap

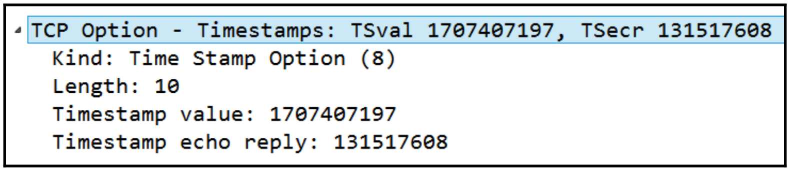
As you can see, SACK can prevent unnecessary retransmissions and keeps the data flowing. Another TCP option is timestamp, which monitors the transmission and keeps track of the round-trip time during the data exchange.

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 timestamp options, TCP can monitor the round-trip times because the sending host may need to retransmit a packet if it does not receive an acknowledgment in a timely manner.

In the `Flow312.pcap` capture in frame 2, open the options to view the timestamp option, as shown in the following screenshot:



```
TCP Option - Timestamps: TSval 1707407197, TSecr 131517608
  Kind: Time Stamp Option (8)
  Length: 10
  Timestamp value: 1707407197
  Timestamp echo reply: 131517608
```

TCP timestamp option

Within this option, there are the following:

- **Kind**: (one byte) `8`. The `Kind` field indicates the type of option, in this case `8`, which is the timestamp option.
- **Length**: (one byte) `10`. This indicates the length of this options header; in this case, it is 10 bytes.
- **Timestamp value (TSval)**: (four bytes) `1707407197`, which is the timestamp clock on the sender's side.
- **Timestamp echo reply (TSecr)**: (four bytes) `131517608`, which is the echo reply sent by the remote host.

TCP uses the timestamp value to monitor the round-trip time in various segments in the path. The timestamp option must be set during the handshake, but you will see the options reporting during the conversation.

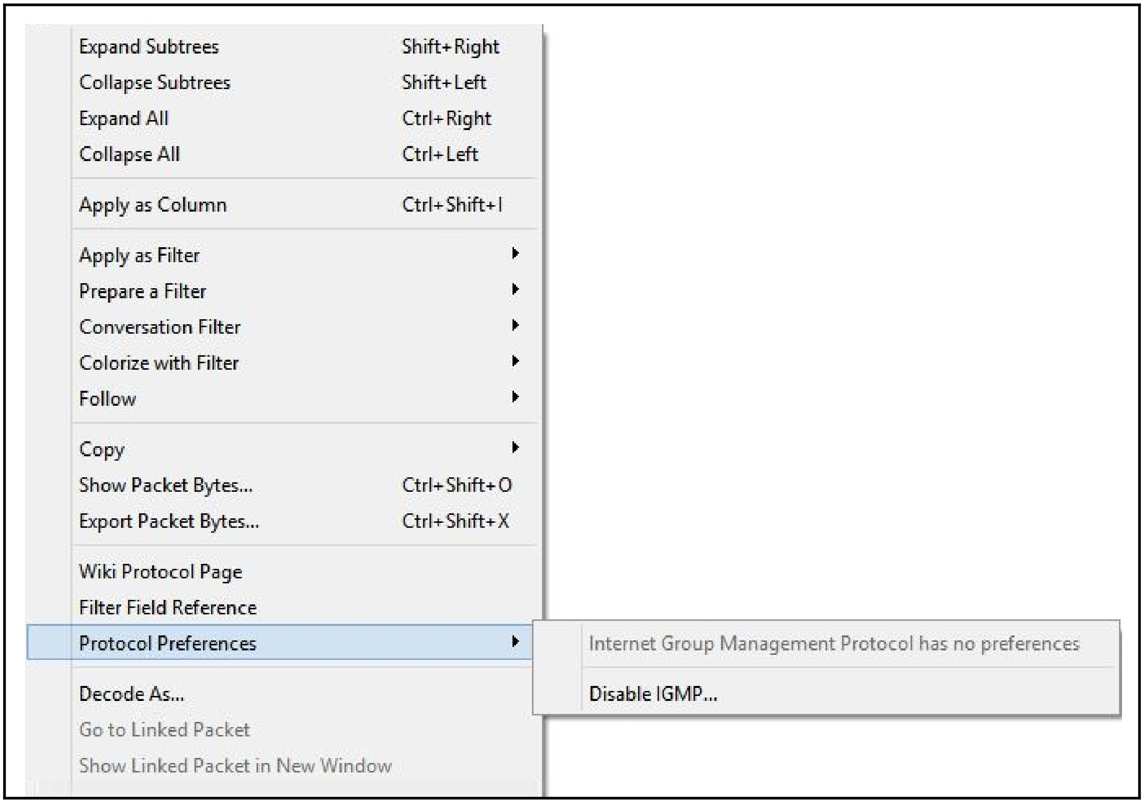
As we can see, TCP can set various options during the handshake that further define the parameters of the conversation.

While working with Wireshark, there may be a preference in the way the protocol responds or is configured. We can modify may of the protocol preferences, as we'll see next.

Understanding TCP protocol preferences

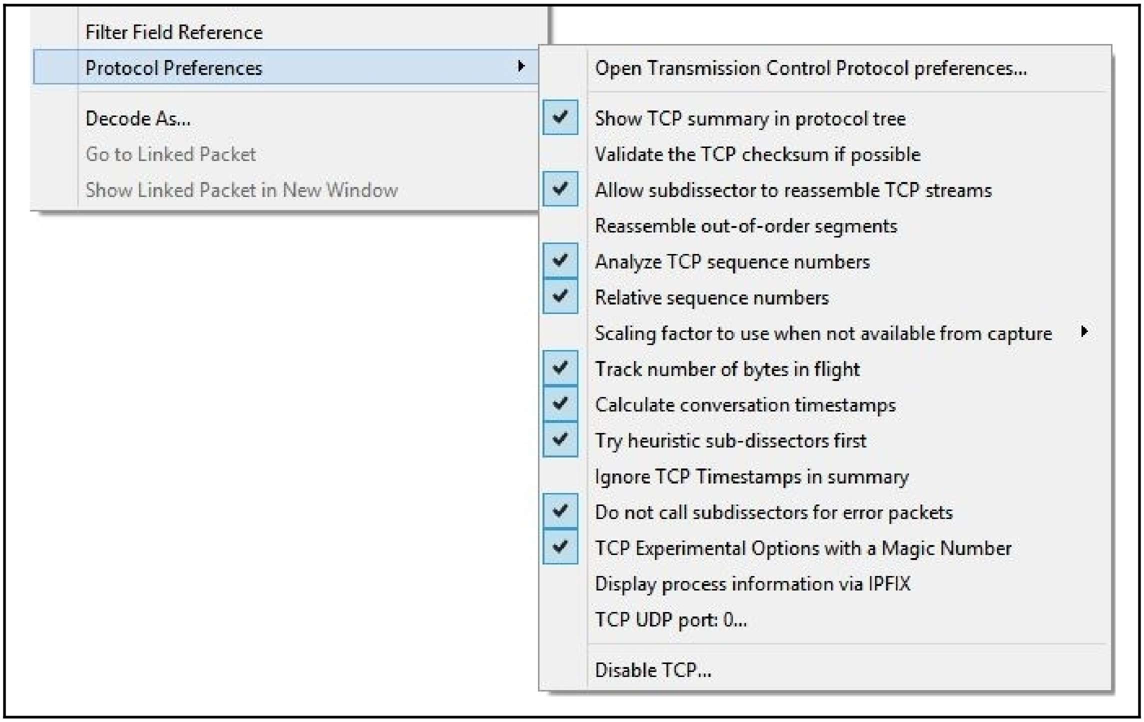
In Wireshark, there are several protocols that we can modify so that Wireshark can display the data more in line with the way the protocol should be used, according to our preferences.

In some cases, a protocol won't have any protocol preferences. For example, when selecting IGMP, Wireshark states **Internet Group Management Protocol has no preferences,** as shown in the following screenshot. However, there are preferences for many protocols, including TCP:


Expand Subtrees Shift+Right
Collapse Subtrees Shift+Left
Expand All Ctrl+Right
Collapse All Ctrl+Left
Apply as Column Ctrl+Shift+I
Apply as Filter
Prepare a Filter
Conversation Filter
Colorize with Filter
Follow
Copy
Show Packet Bytes... Ctrl+Shift+O
Export Packet Bytes... Ctrl+Shift+X
Wiki Protocol Page
Filter Field Reference
Protocol Preferences
Decode As...
Go to Linked Packet
Show Linked Packet in New Window
Internet Group Management Protocol has no preferences
Disable IGMP...

Protocol preferences—IGMP

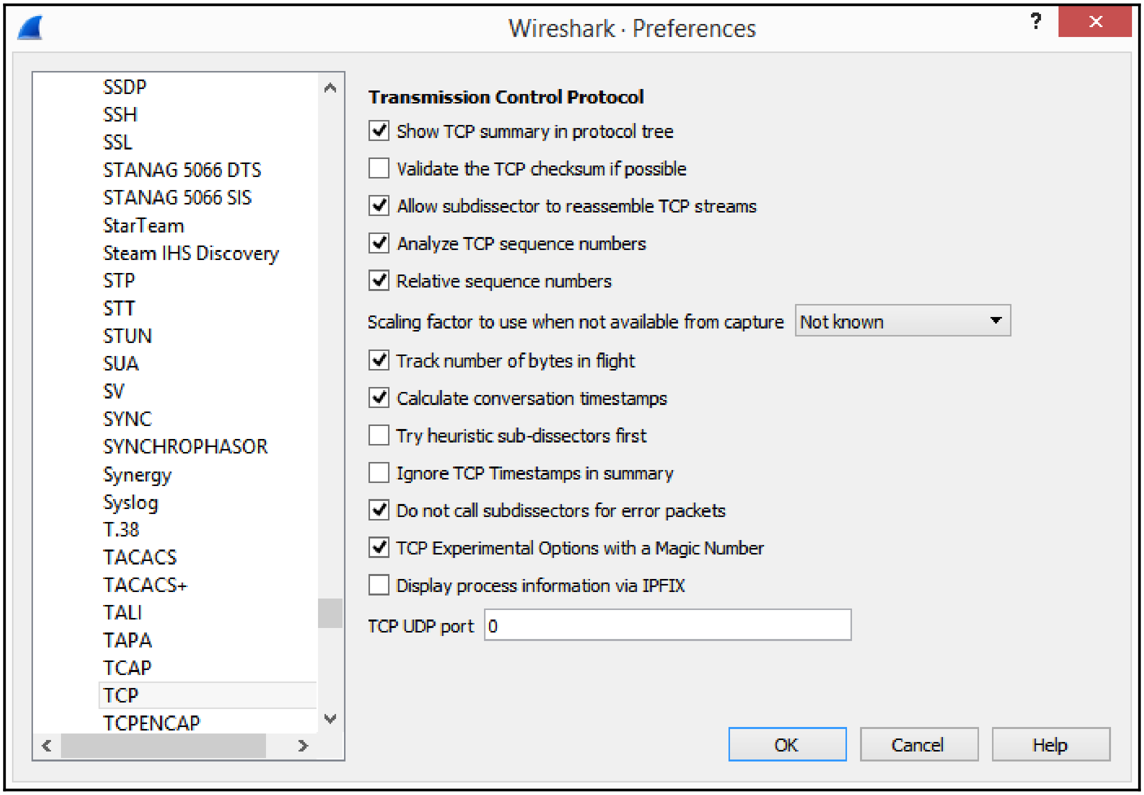
To modify TCP preferences, go to frame 1 and select the TCP header. Right-click and select **Protocol Preferences**, as shown in the following screenshot:


Filter Field Reference
Protocol Preferences
Decode As...
Go to Linked Packet
Show Linked Packet in New Window
Open Transmission Control Protocol preferences...

☑ Show TCP summary in protocol tree
□ Validate the TCP checksum if possible
☑ Allow subdissector to reassemble TCP streams
□ Reassemble out-of-order segments
☑ Analyze TCP sequence numbers
☑ Relative sequence numbers
Scaling factor to use when not available from capture
☑ Track number of bytes in flight
☑ Calculate conversation timestamps
☑ Try heuristic sub-dissectors first
□ Ignore TCP Timestamps in summary
☑ Do not call subdissectors for error packets
☑ TCP Experimental Options with a Magic Number
□ Display process information via IPFIX
TCP UDP port: 0...

Protocol preferences—TCP

Above **Protocol Preferences**, there is another option, **Open Transmission Control Protocol preferences...**, which will open the Wireshark preferences:



Transmission Control Protocol

☑ Show TCP summary in protocol tree
□ Validate the TCP checksum if possible
☑ Allow subdissector to reassemble TCP streams
☑ Analyze TCP sequence numbers
☑ Relative sequence numbers
Scaling factor to use when not available from capture: Not known
☑ Track number of bytes in flight
☑ Calculate conversation timestamps
□ Try heuristic sub-dissectors first
□ Ignore TCP Timestamps in summary
☑ Do not call subdissectors for error packets
☑ TCP Experimental Options with a Magic Number
□ Display process information via IPFIX
TCP UDP port: 0

TCP preferences

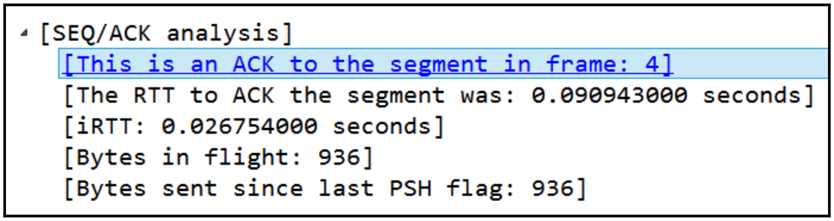
Once the **Preferences** dialog box is open, you can select and modify some of the TCP preferences.

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.
- **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, TCP duplicate acknowledgments, and TCP zero window.
- **Relative sequence numbers**: When used, this feature helps make the sequence numbers easier to read and compare. The relative sequence numbers start with 0 for the first packet in each stream and then increment from that point.
- **Scaling factor to use not available from capture**: Window scale is used to increase the maximum WS that is allowed. There are times that the scaling factor is not known, for example, when the capture started mid-stream and the handshake was not captured. This option allows you to enter a scaling factor, if known.
- **Track number of bytes in flight**: To see the bytes in flight, in `Flow312`, 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]
```

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. By selecting **Try heuristic sub-dissectors first**, Wireshark will dissect the packet 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.

- **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, in some cases, 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**: IP flow information export 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.
- **TCP UDP port**: Use this option if you want to change the protocol's behavior. For example, the simple service discovery protocol uses UDP port `1900`. If you modify this and enter TCP UDP port `1900`, Wireshark will recognize and identify UDP port `1900` as TCP.

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.

Doing analysis will involve investigating all aspects of a protocol's behavior. Now, you can see how you can personalize your preferences when working with Wireshark. This final section provides an overview of TCP teardown, which properly closes the connection between two endpoints.

Tearing down a connection

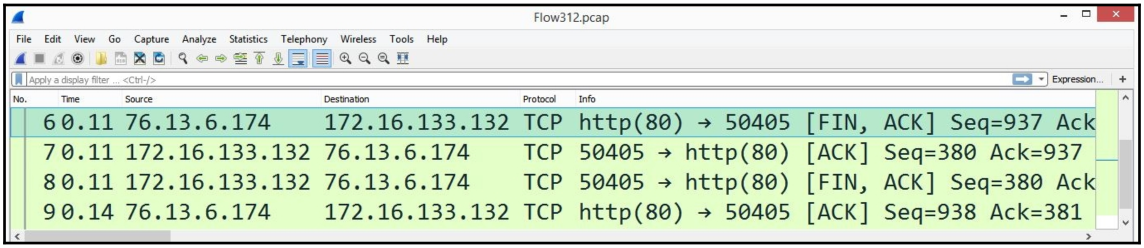
When a TCP connection is complete, TCP tears down the connection by exchanging a series of FIN packets, closing the port and refusing any more requests to communicate. Let's walk through the entire process.

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 outline and further define the parameters of the conversation.
- During the conversation, TCP monitors the communication and acknowledges all data received to ensure 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.

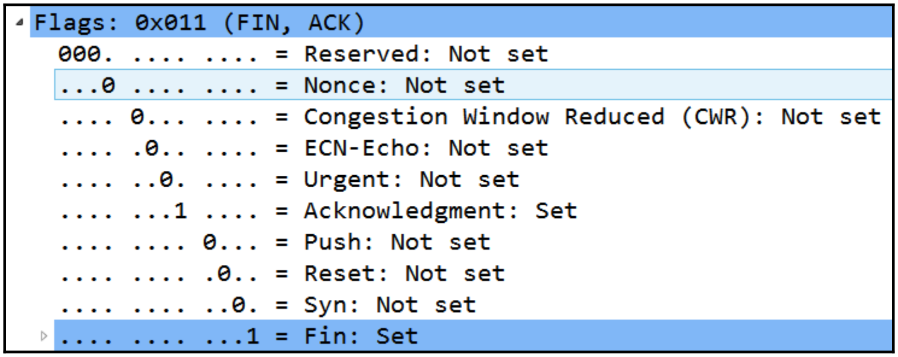
In the `Flows312.pcapng` capture, packets 6, 7, 8, and 9 represent the session teardown, as shown here:



```
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
```

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:



```
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
```

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.

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 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 protocol preferences in Wireshark. Then, we saw 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. Because IPv4 is a completely different format to 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 *Assessment:*

1. To filter only on packets that you have marked in Wireshark, use _____ in the display filter.
    1. marked:all
    2. frame =black
    3. frame.marked==1
    4. marked: on
2. ____ is used to increase the maximum WS that is allowed.
    1. NOP
    2. Window scale
    3. Timestamp
    4. SACK
3. When using _____, the receiver will notify the sender if there are any missing packets.
    1. NOP
    2. Window scale
    3. Timestamp
    4. SACK
4. TCP ends the session by exchanging packets indicating that each side should close their respective socket. TCP uses the _____ flag to indicate that this is the end of a conversation.
    1. END
    2. SYN
    3. FIN
    4. 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.
    1. 32
    2. 380
    3. 936
    4. 33,304

11
Analyzing IPv4 and IPv6

Anyone who works in networking will, at some point, work with the **Internet Protocol** (**IP**), as it is responsible for delivering data over a network. For that reason, it's important to have a solid understanding of the IP. In this chapter, we'll take a closer look at the IP, which is responsible for two key roles: addressing and routing data.

To strengthen your analytical skills, we'll start with a thorough overview of IPv4 and IPv6 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, along with 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 compare the different address types used in IPv6.

So that you can learn how to customize IPv4 and IPv6 in Wireshark, we'll evaluate the protocol preferences. Finally, because IPv4 is a completely different format to 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:

- Understanding the purpose of the IP
- Outlining IPv4
- Exploring IPv6
- Editing protocol preferences
- Discovering tunneling protocols

Understanding the purpose of the IP

The IP 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, if they have to pass through other networks to reach their destination.

One of the main protocols in the network layer is the 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 the IP does not retain any state information; that process is left to the higher-level protocols.

Although IP can't guarantee delivery, it can prioritize traffic, so that the data can be delivered faster. Because of the unpredictable nature of the internet, IP can be prioritized so that time-sensitive data such as VoIP and streaming media is delivered at a higher precedence than email or web pages. The priority is marked in a field value in IPv4 using the **differentiated services** (**DiffServ**) field, or in IPv6 using the traffic class field.

Wireshark provides exceptional support for IP. In this chapter, we compare IPv4 with IPv6, as both protocols are currently in use. To examine the 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 in Wireshark.

Let's begin with an evaluation of IPv4.

Outlining IPv4

In 1981, RFC 791 outlined the specifications for IPv4, which had two principal tasks: addressing and fragmentation, as defined in section 1.4, operation, 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, on today's networks, fragmentation is rarely used.

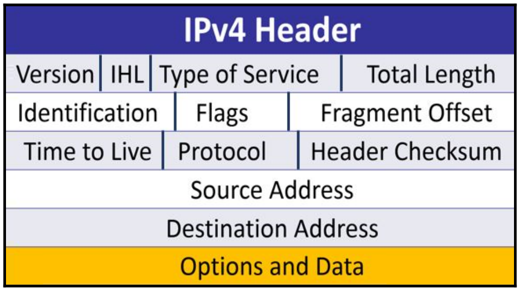
As time has passed, we can see that IPv4 is still influential in addressing, along with the role of routing, in order to get data to its final destination.

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 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 this next section, we will examine the header and the field values so that you can be confident when looking at a packet capture that you can 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 | Type of Service | Total Length

Identification | Flags | Fragment Offset

Time to Live | Protocol | Header Checksum

Source Address

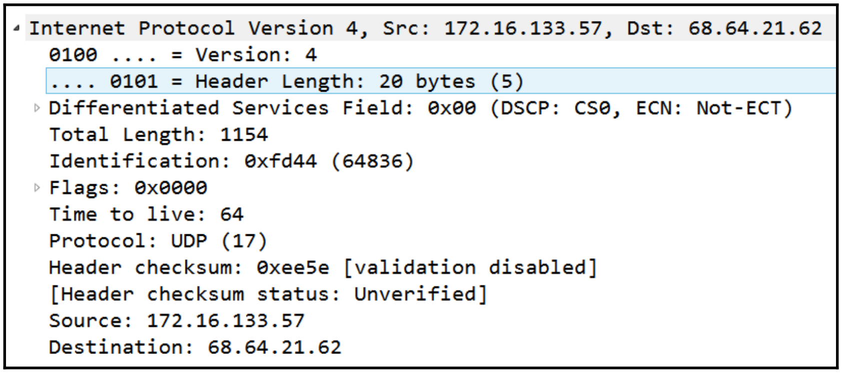
Destination Address

Options and Data

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:



```
◂ 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
```

bigFlows frame 1—IPv4 header

The following section will list each field, and how many bits or bytes are used in each field, 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.**
- **Header length 4-bit**: The header length is in multiples of four 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**. Although the IPv4 header is shown in the *Type of Service* graphic lists, Wireshark will decode an IPv4 **Type of Service** (**TOS**) field as a DiffServ field, 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. In IPv4, the TOS field can be used to prioritize traffic so that it is delivered at a higher precedence than email or web pages.

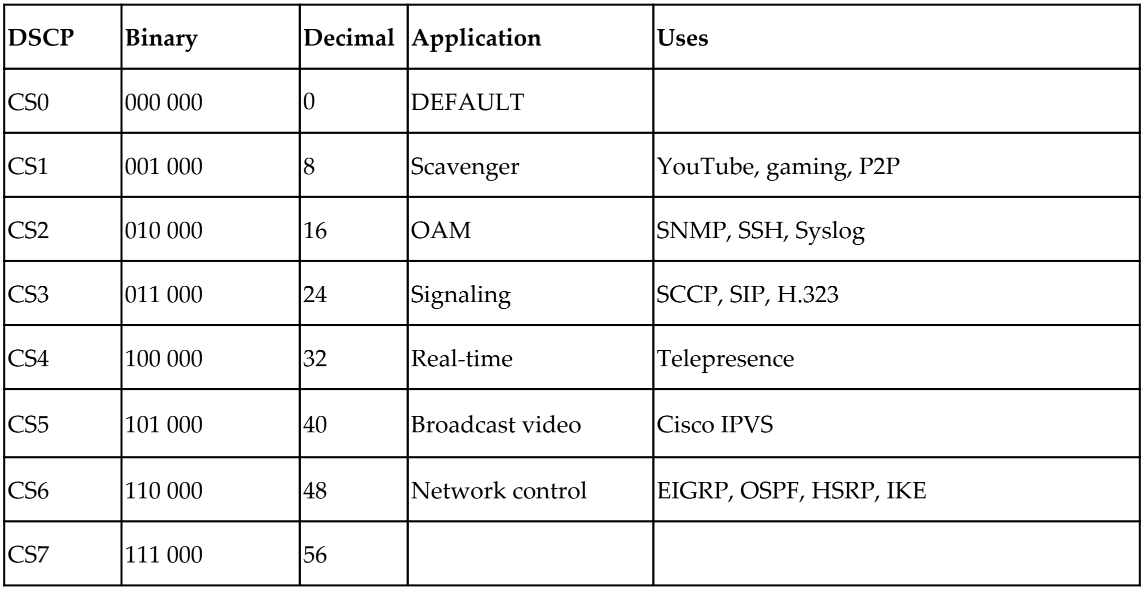
Wireshark presents the TOS field as DiffServ, which is similar to TOS in offering prioritization, but with subtle differences and improvements to handle the real-time protocols in use today:

- **DiffServ 8-bit**: This field is separated into two functions: **Quality of Service** (**QoS**) and **Explicit Congestion Notification** (**ECN**).

In frame 1, we can see the **Differentiated Services Field: 0X00 (DSCP: CSO, ECN: Not-ECT)**. Let's step through what this represents. We'll start with the first 6-bit 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 the following **class selector** (**CS**) values:

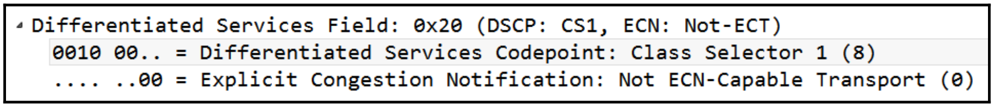


DSCP	Binary	Decimal	Application	Uses
CS0	000 000	0	DEFAULT	
CS1	001 000	8	Scavenger	YouTube, gaming, P2P
CS2	010 000	16	OAM	SNMP, SSH, Syslog
CS3	011 000	24	Signaling	SCCP, SIP, H.323
CS4	100 000	32	Real-time	Telepresence
CS5	101 000	40	Broadcast video	Cisco IPVS
CS6	110 000	48	Network control	EIGRP, OSPF, HSRP, IKE
CS7	111 000	56		

Differentiated services field values

The first column shows the **Differentiated Services Code Point** (**DSCP**), which lists the CS. As shown in frame 1, 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 display filter, `ip.dsfield.dscp > 0`. 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)
```

CS 1

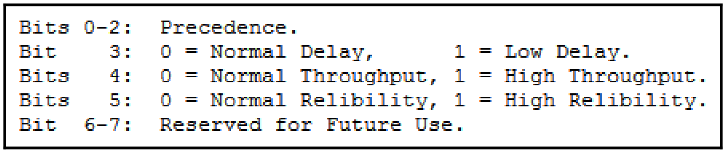
`Class Selector 1 (8)` is used in scavenger applications such as YouTube, gaming, and P2P, as this traffic would benefit from having a (slightly) higher priority when traveling over the internet.

The last two bits of the DiffServ field are used to identify ECN, which helps to manage congestion on the network.

Sending an ECN

You may not have been aware of the ECN and its significance; however, this can have an impact in how devices communicate congestion on the network. Let's take a look at how these two bits can improve data flow.

In the original RFC 791, the last two bits of the DiffServ field are `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 Relibility, 1 = High Relibility.
Bit  6-7:  Reserved for Future Use.
```

Service bit assignments

In 2001, RFC 3168 (`https://tools.ietf.org/html/rfc3168`) found a use for the last two bits. RFC 3168 outlined ECN, which provides a congestion notification on the network. Let's see how ECN improves 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 congestion notification. This ultimately prevents the additional traffic that occurs when there are duplicate acknowledgments and fast retransmissions.

ECN uses both the TCP and IP header, as outlined here:

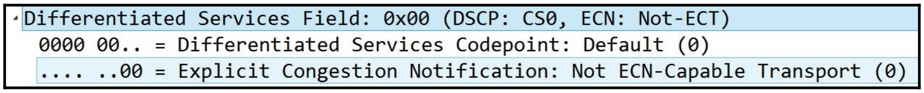
- The IP header uses the two bits at the end of the differentiated services field to indicate **ECN-Capable Transport** (**ECT**) and **Congestion Experienced** (**CE**).
- The TCP header uses two flags: CWR and ECE.

When using ECN, the two 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:

Bits	**Indication**	**Identifier**
00	Non ECN-Capable Transport)	Non-ECT
10	ECN Capable Transport)	ECT(0)
01	ECN Capable Transport	ECT(1)
11	Congestion Encountered	CE

As you can see, code points 01 and 10 are basically the same.

In `bigFlows.pcap` 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)
```

Not ECT-Capable

The devices involved in the connection will communicate with one another and when available, use ECN, which helps notify endpoints on congestion issues.

Although the 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. The next group of field values in the IP header deal with using fragmentation.

Fragmenting the data

In RFC 791, the 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, which divides a datagram into smaller pieces, so that they can be sent on the network with a restrictive MTU.

The following fields are related to fragmentation: **Identifier**, **Flags**, and **Fragment offset**. Total length is, in part, related to fragmentation. However, it has other implications as well.

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 reasons:

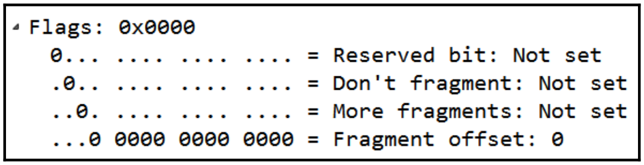
- During troubleshooting, you may need to look at the fields when determining why data may not be getting through.
- During a security assessment, since use of the fragmentation fields could be an indication of malicious activity.

Let's look at the next four fields in the IP header:

- **Total length 16-bit**: This indicates the value of the header length and any data. The field value is 16-bit, which means the entire length cannot exceed 2^{16}, or 65,535 bytes.

  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.

- **Identification 16-bit**: This 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
```

IP flags

- **Fragment Offset 13-bit**: After the three flags in the IP header, there is a **Fragment offset** field, which 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,** because, in today's networks, fragmentation is not used as most pipelines have generous bandwidth with an acceptable MTU.

Internet Control Message Protocol (**ICMP**) acts as a scout for the IP. When ICMP encounters a network with an MTU that is smaller than the size of the packet, and 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.`

To successfully send data through a network with restrictive bandwidth without using fragmentation, the sending host must retransmit the data using a smaller MSS.

The next few fields are more administrative, as they hold values related to the number of hops, the protocol that follows the IP 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. The following three fields hold a specific value:

- **Time to live 8-bit**: 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.

During regular operations, this most likely won't happen. However, in case there is a routing loop, the **Time to Live** (**TTL**) field value in an IP header is the number of routers or hops a packet can take before dropping the packet. 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. The TTL field is 8-bit, so the maximum value is 2^8, or 255 hops.

In frame 1, the TTL field is set at **Time to live: 64**, which is the default value for this field. The value varies as it is OS-dependent. To see the TTL values of various OSes, go to `https://subinsb.com/default-device-ttl-values/`.

- **Protocol 8-bit**: The protocol field identifies the higher-layer protocol that follows the IP 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 value offloading to the NIC card.

One of the more significant elements in the IP header is addressing, as we'll discuss in the following section.

Learning 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:

- **Source and destination address 32-bit**: Each field houses the source or destination IPv4 address, which is represented in an easy-to-understand dotted decimal format.

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 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

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.

IPv4 has several ranges of special and private IPv4 addresses, as outlined next.

Reviewing special and private IP addressing

IPv4 has several ranges of 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`

Special and private IPv4 addresses

The table shows a list of the predominant special and private IPv4 addresses. To see a complete list, visit `https://en.wikipedia.org/wiki/Reserved_IP_addresses`.

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 to provide source routing information, timestamps, and others. Several of the IP options have been deprecated and are no longer used. For a more compete discussion, refer to RFC 6814. A current list can be found at `https://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml#ip-parameters-1`, which was updated on May 3, 2018.

When used, the options field must be a multiple of 32-bit, or 4 bytes. Padding may be required so that the header is a multiple of 32-bit.

Now that we have reviewed IPv4, let's take a closer look at IPv6.

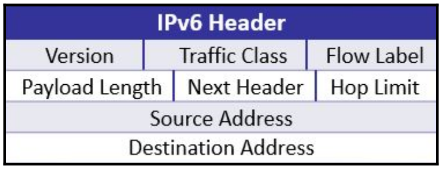
Exploring IPv6

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 IPv6 RFC 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:** Although the header is larger, due to the expanded address space, it is more streamlined.
- **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 more 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 the 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

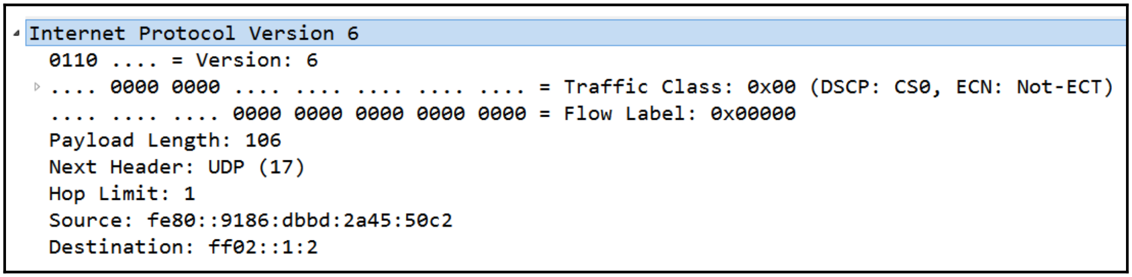
Payload Length | Next Header | Hop Limit

Source Address

Destination Address

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: 0x00000
  Payload Length: 106
  Next Header: UDP (17)
  Hop Limit: 1
  Source: fe80::9186:dbbd:2a45:50c2
  Destination: ff02::1:2
```

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, starting with the version, a field to house the TOS, 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. The traffic class field houses two (2) values, TOS and ECN:
    - **TOS:** The first 6-bit of this field is used to communicate what type of service is requested. TOS uses the same DSCP values as IPv4. Frame 347 uses the default value, **Differentiated Services Codepoint: Default (0)**.
    - **ECN:** The last 2-bit 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 is a new field that 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. However, after two decades, the flow label is considered experimental, and is generally not used.

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 as IPv4 to identify the higher-layer protocol. However, if there is an extension header, this field will indicate what extension header follows the IPv6 header. In 2017, IANA updated the list for the next header field. The list can be found at `https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml`.

- **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-bit to 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, along with an overview of IPv6 address types:

- **Source and destination address 128-bit:** 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, but not all, protocols, Wireshark provides a means to modify the way in which Wireshark presents the data. The following gives us some insight of how to adjust preferences for both IPv4 and IPv6.

Editing protocol preferences

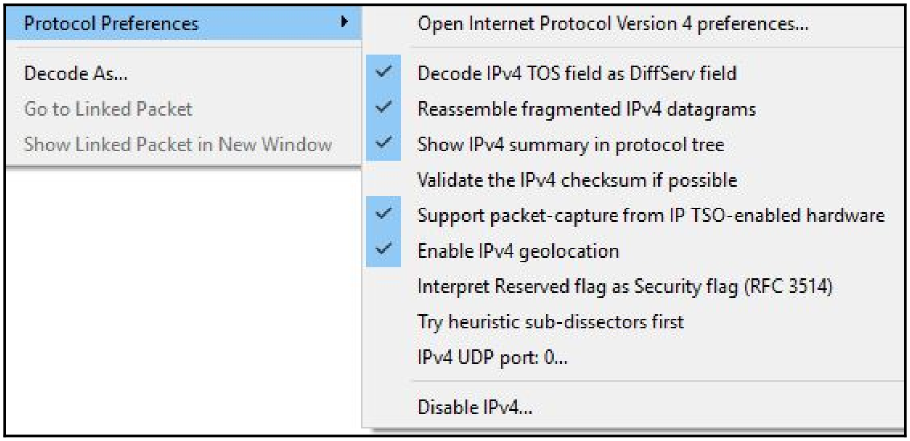
In Wireshark, you can modify most protocols by doing the following:

- Right-clicking while on the header and selecting **Protocol Preferences**, where you will see a list of preferences.
- Go to **Edit** | **Preferences** | **Protocols**, and then select 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:



Open Internet Protocol Version 4 preferences...

✓ Decode IPv4 TOS field as DiffServ field

✓ Reassemble fragmented IPv4 datagrams

✓ Show IPv4 summary in protocol tree

Validate the IPv4 checksum if possible

✓ Support packet-capture from IP TSO-enabled hardware

✓ Enable IPv4 geolocation

Interpret Reserved flag as Security flag (RFC 3514)

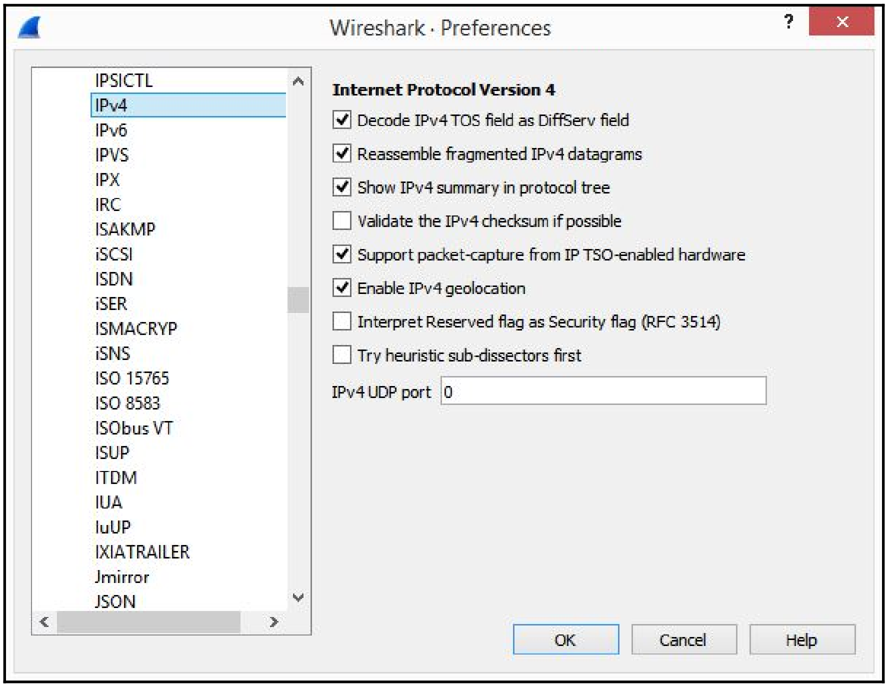
Try heuristic sub-dissectors first

IPv4 UDP port: 0...

Disable IPv4...

IPv4 preference shortcut

Once you select the shortcut, a list of preferences will be listed, as shown in the following screenshot:



Internet Protocol Version 4

☑ Decode IPv4 TOS field as DiffServ field

☑ Reassemble fragmented IPv4 datagrams

☑ Show IPv4 summary in protocol tree

☐ Validate the IPv4 checksum if possible

☑ Support packet-capture from IP TSO-enabled hardware

☑ Enable IPv4 geolocation

☐ Interpret Reserved flag as Security flag (RFC 3514)

☐ Try heuristic sub-dissectors first

IPv4 UDP port 0

IPv4 preferences

Once there, you can modify the selections as follows:

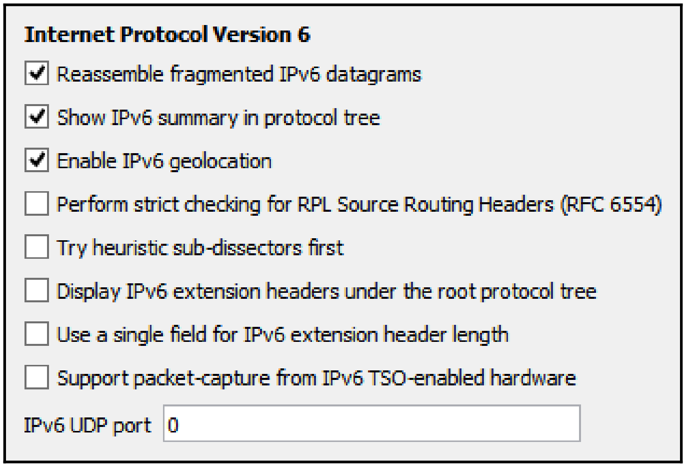
- **Decode IPv4 TOS field as DiffServ field**: RFC 791 used TOS to classify traffic. Over time, this field was modified to identify traffic using DiffServ, which allows for a wider range of classification. In most cases, this should be enabled.
- **Reassemble fragmented IP datagrams**: When necessary, 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**: Use this option if you want to change the protocol behavior to a specific port, when used on the LAN.

For any of the options that make a change to the default values, caution is advised, as what you enter may **stick** and will not allow you to undo the option without a reinstall.

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:



Internet Protocol Version 6

☑ Reassemble fragmented IPv6 datagrams

☑ Show IPv6 summary in protocol tree

☑ Enable IPv6 geolocation

□ Perform strict checking for RPL Source Routing Headers (RFC 6554)

□ Try heuristic sub-dissectors first

□ Display IPv6 extension headers under the root protocol tree

□ Use a single field for IPv6 extension header length

□ Support packet-capture from IPv6 TSO-enabled hardware

IPv6 UDP port 0

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 Header (RFC 6554)**: If enabled, this will aid in troubleshooting streams using **Routing Protocol for Low-Power and Lossy Networks** (**RPL**).
- **Try heuristic sub-dissector fist**: 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**: Use this option if you want to change the protocol behavior.

For any of the options that will change the default values, caution is advised, as what you enter may **stick** and will not allow you to undo the option without a reinstall.

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 outlines how the two protocols can coexist with one another on the same network, using various tunneling protocols.

Discovering tunneling protocols

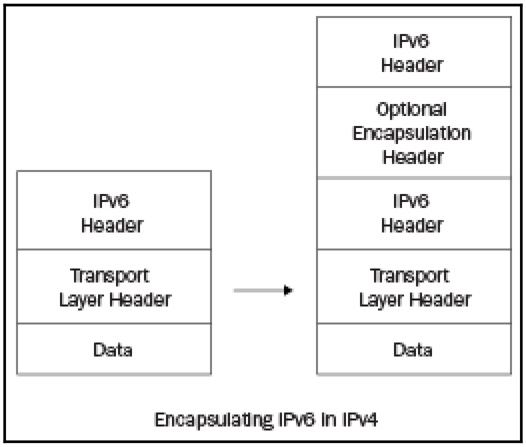
While some organizations have decided to make the switch to a dedicated IPv6 network setting, many are running a dual stack environment, where hosts use both IPv4 and IPv6 and must communicate with one another.

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 protocols is outlined in RFC 7059, found at `https://tools.ietf.org/html/rfc7059`, written in 2013. Some of the tunneling protocols include the following:

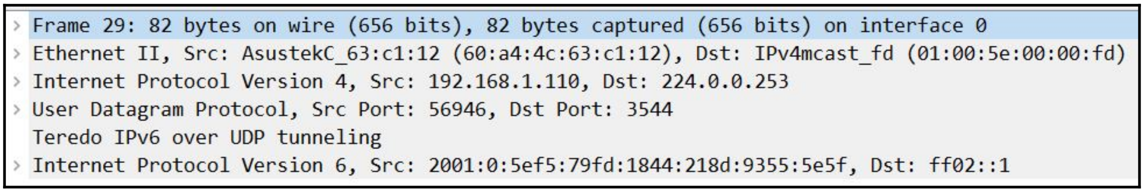
- **ISATAP** (short for **Intra-Site Automatic Tunnel Addressing Protocol**): Transmits data to and from hosts that use IPv6 through an IPv4 network
- **Teredo**: Generated in a Windows OS to allow IPv4 hosts to connect to an IPv6 network when **network address translation** (**NAT**) is in place
- **GRE** (short for **Generic Routing Encapsulation**): Creates a point-to-point IPv6 connection within an IPV4 network

The following diagram from RFC 7059 shows the proper format for encapsulation of an IPv6 packet within an IPv4 packet:



Encapsulation of an IPv6

Teredo wraps or encapsulates an IPv4 packet with an IPv6 header so that the packets can travel over an IPv6 environment. 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 IPv4 packet encapsulated in an IPv6 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
```

IPv4 packet encapsulated in an IPv6 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 own network.

Summary

By now, you should have a solid understanding of the role and purpose of the IP, and its influence in addressing and routing data. In this chapter, we covered a brief history of IP. We now know that both versions of IP can do the job of routing and addressing; however, there are several differences between the IPv4 and IPv6 headers. We then examined and explained each of the field values of both IPv4 and IPv6. To give you a better understanding of the two protocols, we compared some of the similarities along with some of 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 sister protocol to IP that 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 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 *Assessment*:

1. Class selector 6 in the DiffServ field is used with _____.
    1. Signaling
    2. Broadcast video
    3. Network control
    4. Realtime
2. The IP address 172.18.23.119 is a
    1. Class C IPv4 address
    2. Class B private IPv4 address
    3. Class E IPv4 address
    4. Class D private IPv4 address

3. An IPv6 address has _____ bit.
    1. 32
    2. 48
    3. 64
    4. 128
4. In IPv4, we use a TTL value that indicates the number of hops it can take when traveling through the network. In IPv6, this field value is called _____.
    1. Router pass
    2. Class stop
    3. TTL
    4. Hop count
5. An IPv4 header has _____ flags.
    1. 1
    2. 2
    3. 3
    4. 4

12
Discovering ICMP

Everyone is familiar with **Internet Protocol** (**IP**), which is responsible for routing and addressing traffic. However, many people are not familiar with **Internet Control Message Protocol** (**ICMP**), the unsung hero of the network layer. ICMP is a powerful protocol that helps IP do its job.

In this chapter, we'll learn about ICMP, which is the sister protocol to IP and works in the network layer of the OSI model. First, we'll go through 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 will look at common type and code values, as well as some basic firewall guidelines in terms of what types of ICMP messages to allow on your network. At the end of the chapter, you'll see how ICMP is the scout for IP and plays a major role in delivering data.

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

- Understanding the purpose of ICMP
- Dissecting ICMPv4 and ICMPv6
- Sending ICMP messages
- Evaluating type and code values
- Configuring firewall rules

Understanding the purpose of ICMP

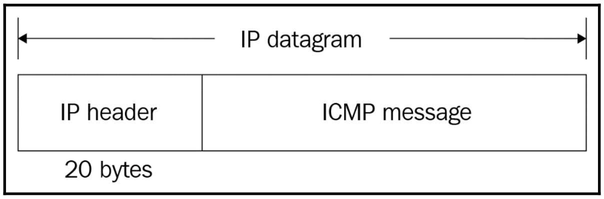
Early on, scientists developed protocols that drove internet traffic. In addition, they identified potential issues that might prevent traffic from reaching its destination, especially when using IP, as it doesn't guarantee delivery and has no way of communicating network problems with end devices. ICMP overcomes the deficiencies of IP by sending query messages and generating error reports on possible issues that may require attention.

The network layer is responsible for addressing and routing traffic. IP is a best-effort, unreliable protocol. As a result, ICMP is essential for data delivery and must be implemented by every IP module. ICMP communicates issues that prevent data delivery. Common errors include the network or port being unreachable. ICMP can also issue queries such as an echo request/reply, which is used in the ping network utility.

Because there are two IP versions, there are two versions of ICMP, which have roles that are specific to their respective IP version:

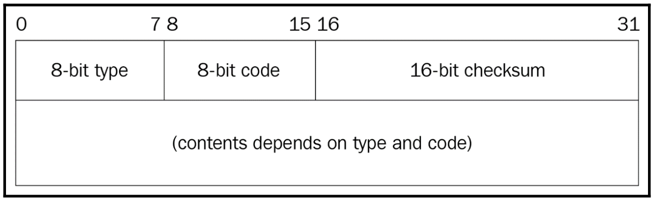
- IPv4 uses ICMPv4
- IPv6 uses ICMPv6

An ICMP packet is nested within an IP packet, as shown in the following diagram:



ICMP message within an IP packet

All ICMP messages have a common structure that begins with the type, code, and checksum, as shown along the top of the following diagram:



ICMP message

The three fields are consistent in an ICMP header. After the header, you'll find the data payload, where the contents will depend on the ICMP type and code. Let's start with the three header fields, as outlined in the next section.

Understanding the ICMP header

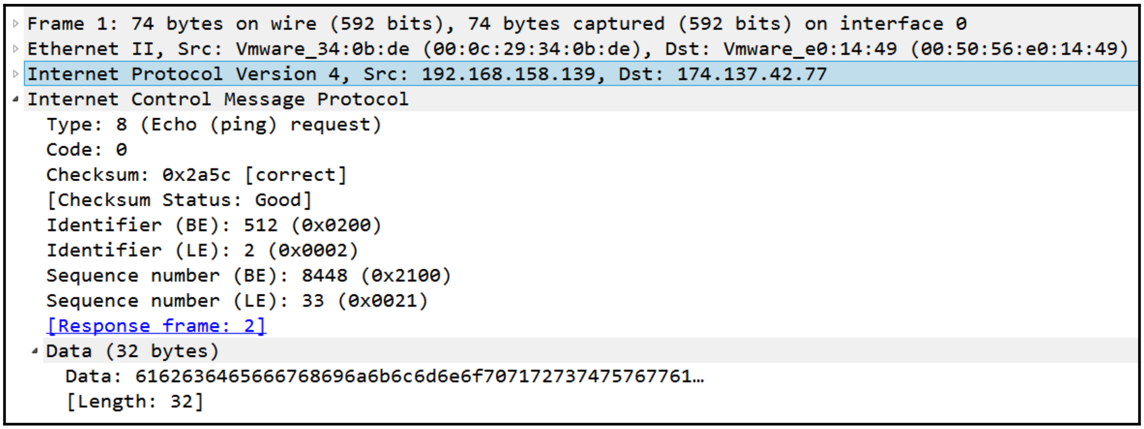
To communicate information, the ICMP message must provide information within the header, as follows:

- **8-bit type**: This field indicates the type, such as type 0—echo reply.
- **8-bit code**: The code field further defines the type field. For example, type 3—destination unreachable, might have a corresponding code 2—protocol unreachable.
- **16-bit checksum**: This field holds a numeric value used for error detection.

Following the type, code, and checksum are the contents of the ICMP message. The contents will depend on what was sent, which can either be an error report or a query message.

To see an example of an echo request/reply, go to CloudShark at `https://www.cloudshark.org/captures/fe65ed807bc3` and open `icmp.pcap` in Wireshark.

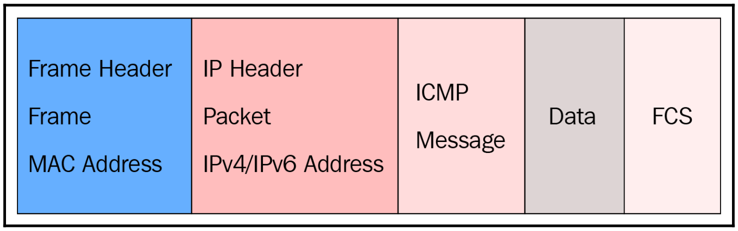
In this example, frame 1 of the echo request/reply shows a type 8, code 0 message. Expand the ICMP header, as shown in the following screenshot:



```
▷ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▷ Ethernet II, Src: Vmware_34:0b:de (00:0c:29:34:0b:de), Dst: Vmware_e0:14:49 (00:50:56:e0:14:49)
▷ Internet Protocol Version 4, Src: 192.168.158.139, Dst: 174.137.42.77
◢ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x2a5c [correct]
    [Checksum Status: Good]
    Identifier (BE): 512 (0x0200)
    Identifier (LE): 2 (0x0002)
    Sequence number (BE): 8448 (0x2100)
    Sequence number (LE): 33 (0x0021)
    [Response frame: 2]
  ◢ Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f707172737475767761…
      [Length: 32]
```

ICMP echo request details

As shown in the preceding screenshot, the details for this type of ICMP message include fields for identifiers and sequence numbers, which help to match corresponding echoes and replies. The entire payload is encapsulated in a frame, as shown in the following diagram:



ICMP message in an Ethernet II frame

Here, we see the various headers, which includes the frame header, the IP header, the ICMP message, and the data.

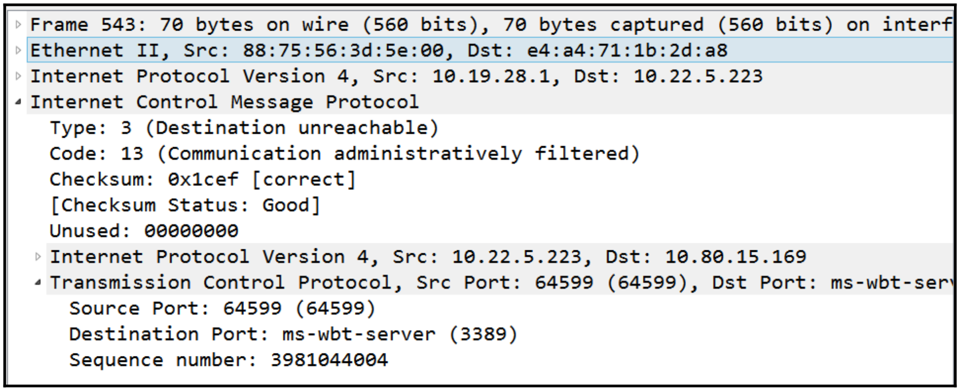
ICMP does not have a transport layer header, as it does not exchange or transport data. Its primary role is to test for reachability and report transmission errors.

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 OS. For example, in the `Cloudshark icmp.pcap` graphic echo request/reply (shown in the *Sending messages* section), the data portion is a string of characters: `6162636465666768696a6b6c6d6e6f7071727374757677 61`.

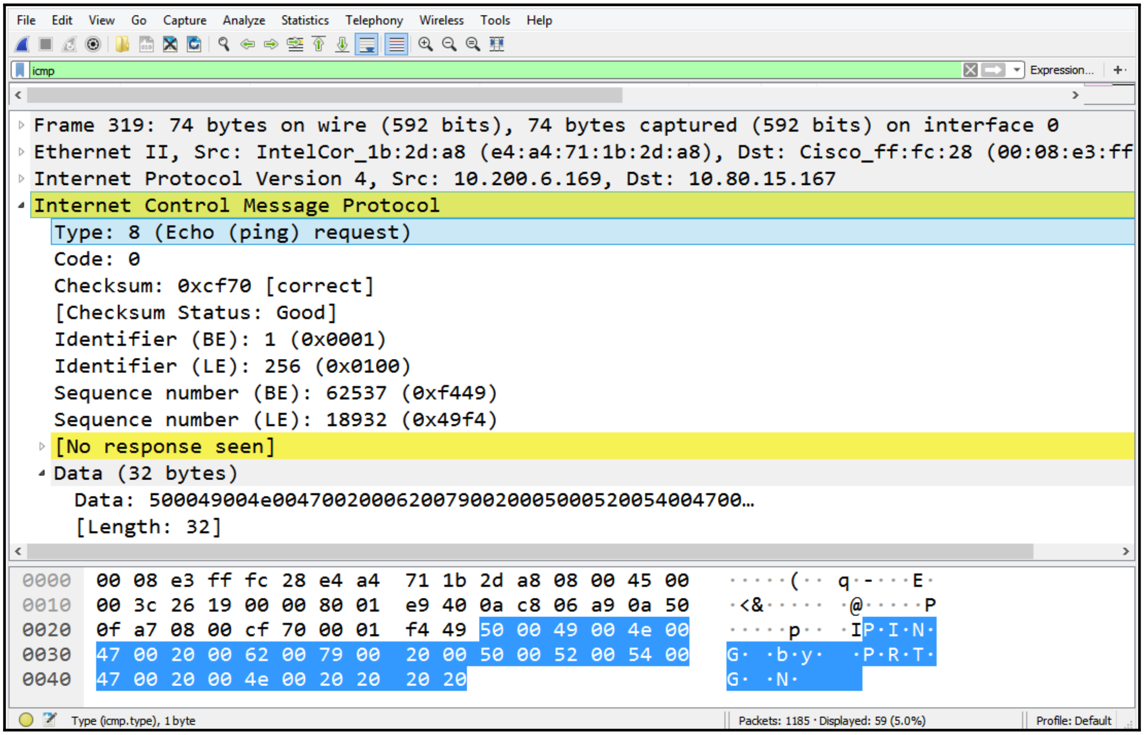
With normal ICMP behavior, when there is an error, ICMP must return the IP header, plus the first eight bytes of the original datagram, to the sender. As shown in the following screenshot, ICMP has returned an ICMP type 3 and code 13, which means a firewall is blocking the request:



```
▹ 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
```

ICMP type 3 and code 13

The data portion in an ICMP request can be modified. For example, ping monitoring, by Paessler (`https://www.paessler.com/ping-monitoring`), has a watermark, as shown in the following screenshot:



```
▹ Frame 319: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▹ Ethernet II, Src: IntelCor_1b:2d:a8 (e4:a4:71:1b:2d:a8), Dst: Cisco_ff:fc:28 (00:08:e3:ff
▹ Internet Protocol Version 4, Src: 10.200.6.169, Dst: 10.80.15.167
◢ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xcf70 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 62537 (0xf449)
    Sequence number (LE): 18932 (0x49f4)
  ▹ [No response seen]
  ◢ Data (32 bytes)
      Data: 500049004e004700200062007900200050005200540047000…
      [Length: 32]

0000  00 08 e3 ff fc 28 e4 a4  71 1b 2d a8 08 00 45 00   ·····(·· q·-···E·
0010  00 3c 26 19 00 00 80 01  e9 40 0a c8 06 a9 0a 50   ·<&····· ·@·····P
0020  0f a7 08 00 cf 70 00 01  f4 49 50 00 49 00 4e 00   ·····p·· ·IP·I·N·
0030  47 00 20 00 62 00 79 00  20 00 50 00 52 00 54 00   G· ·b·y·  ·P·R·T·
0040  47 00 20 00 4e 00 20 20  20 20                     G· ·N·

Type (icmp.type), 1 byte          Packets: 1185 · Displayed: 59 (5.0%)          Profile: Default
```

Ping request with a watermark

In this case, the watermark is not malicious. However, an ICMP packet can be modified to exfiltrate data by using the **Loki** tool to execute a covert channel attack. Data is embedded within an ICMP packet and is sent through the network, which poses a security risk. As a result, the network administrator should tune devices to enable the inspection of ICMP data, and send an alert if the payload contains a 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. Let's take a look at the two versions: ICMPv4 and ICMPv6.

Dissecting ICMPv4 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. ICMPv4 is used with IPv4, and ICMPv6 is used with IPv6.

In the next section, we'll explore ICMPv4 alongside ICMPv6 so that you understand some of the basic roles and functions in reporting network issues.

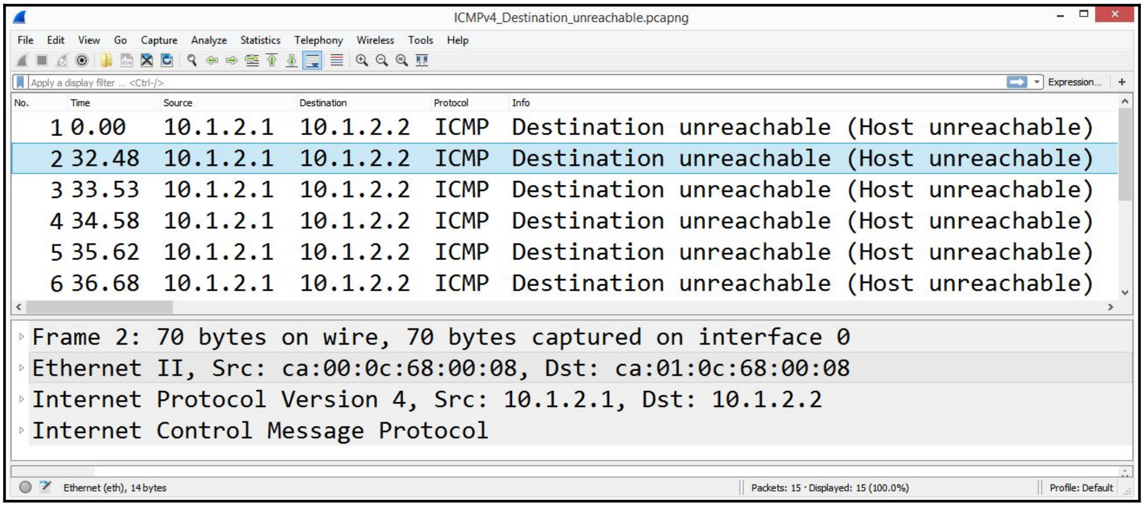
Let's start with ICMPv4, which is commonly referred to as ICMP.

Reviewing ICMPv4

ICMPv4, or simply 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 connectivity issues, 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 ICMP can send and report. In that case, it's up to TCP to notify the host of transmission errors during delivery.

To see an example of an error, we can use this example on CloudShark. Go to `https://www.cloudshark.org/captures/155db9732c91` and then open the file in Wireshark, as shown in the following screenshot:


ICMPv4_Destination_unreachable.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No.  Time   Source    Destination Protocol Info
  1  0.00   10.1.2.1  10.1.2.2    ICMP     Destination unreachable (Host unreachable)
  2  32.48  10.1.2.1  10.1.2.2    ICMP     Destination unreachable (Host unreachable)
  3  33.53  10.1.2.1  10.1.2.2    ICMP     Destination unreachable (Host unreachable)
  4  34.58  10.1.2.1  10.1.2.2    ICMP     Destination unreachable (Host unreachable)
  5  35.62  10.1.2.1  10.1.2.2    ICMP     Destination unreachable (Host unreachable)
  6  36.68  10.1.2.1  10.1.2.2    ICMP     Destination unreachable (Host unreachable)

Frame 2: 70 bytes on wire, 70 bytes captured on interface 0
Ethernet II, Src: ca:00:0c:68:00:08, Dst: ca:01:0c:68:00:08
Internet Protocol Version 4, Src: 10.1.2.1, Dst: 10.1.2.2
Internet Control Message Protocol

Ethernet (eth), 14 bytes    Packets: 15 · Displayed: 15 (100.0%)    Profile: Default
```

ICMP destination unreachable

I have removed the coloring rules to make the graphic more visible, as Wireshark views this as an error and will show up with black coloring. In this capture, ICMP is reporting an error. In the lower half of the screenshot, we see the IP datagram with an IPv4 header, followed by the ICMP header.

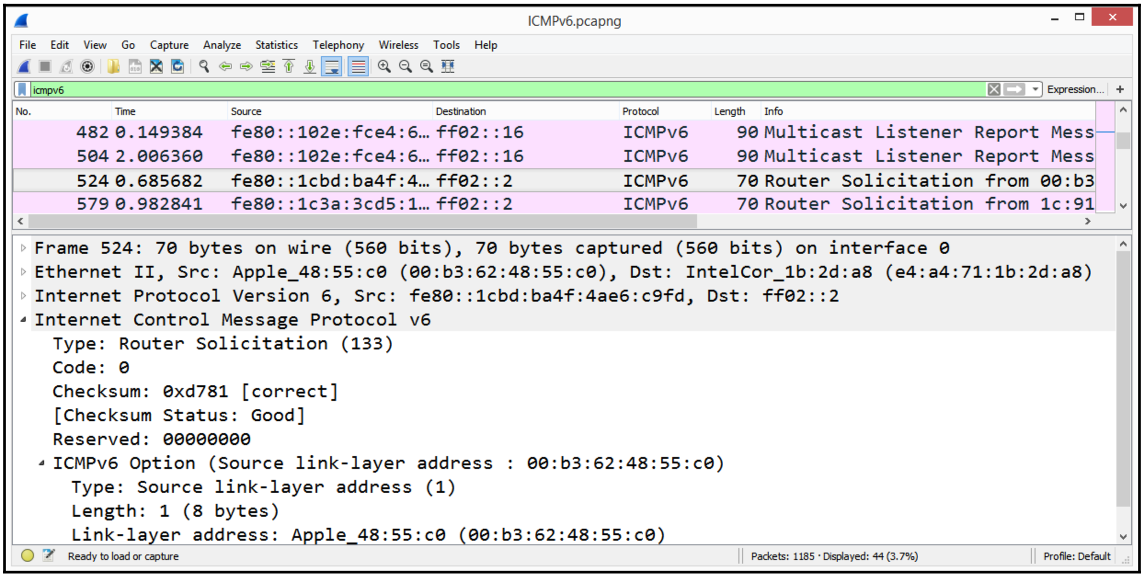
Now that we have reviewed some of the basics of ICMP, let's take a look at ICMPv6, which has many of the same functions, but which also provides additional roles to support IPv6.

Outlining ICMPv6

While IPv4 and IPv6 are similar in terms of their overall functions, IPv6 has many additional benefits. The benefits include options and extensions, improved multicast routing, and **Stateless Autoconfiguration** (**SLAAC**).

As a result, ICMPv6 was developed for IPv6 and is also an important protocol. Along with ICMPv4 for IPv4, ICMPv6 is used to communicate updates or error messages. As stated in RFC 4443, *"the base protocol must be fully implemented by every IP version six node."*

An ICMPv6 message is like an ICMPv4 message in that the header contains the type, code, and checksum, followed by the contents, which will depend on the type and the code. In the following screenshot, frame 524 has an ICMPv6 message:


ICMPv6.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmpv6
Expression...
+

```
No.   Time      Source              Destination  Protocol  Length Info
  482 0.149384  fe80::102e:fce4:6… ff02::16     ICMPv6        90 Multicast Listener Report Mess
  504 2.006360  fe80::102e:fce4:6… ff02::16     ICMPv6        90 Multicast Listener Report Mess
  524 0.685682  fe80::1cbd:ba4f:4… ff02::2      ICMPv6        70 Router Solicitation from 00:b3
  579 0.982841  fe80::1c3a:3cd5:1… ff02::2      ICMPv6        70 Router Solicitation from 1c:91
```

```
Frame 524: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
Ethernet II, Src: Apple_48:55:c0 (00:b3:62:48:55:c0), Dst: IntelCor_1b:2d:a8 (e4:a4:71:1b:2d:a8)
Internet Protocol Version 6, Src: fe80::1cbd:ba4f:4ae6:c9fd, Dst: ff02::2
Internet Control Message Protocol v6
  Type: Router Solicitation (133)
  Code: 0
  Checksum: 0xd781 [correct]
  [Checksum Status: Good]
  Reserved: 00000000
  ICMPv6 Option (Source link-layer address : 00:b3:62:48:55:c0)
    Type: Source link-layer address (1)
    Length: 1 (8 bytes)
    Link-layer address: Apple_48:55:c0 (00:b3:62:48:55:c0)
```

ICMPv6 router solicitation

In the frame details, we see the following:

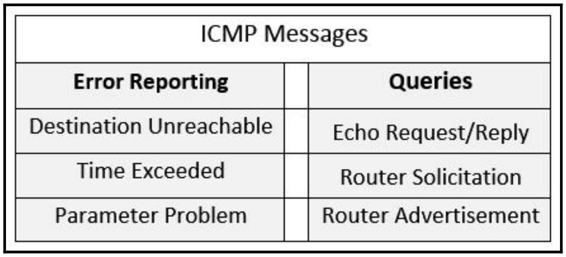
- **Ethernet II**: Frame header
- **Internet Protocol Version 6**: IP header
- **Internet Control Message Protocol v6**: Router solicitation

Although, in many ways, ICMPv4 and ICMPv6 are similar, ICMPv6 has more responsibilities. IPv6 no longer uses **Address Resolution Protocol** (**ARP**) broadcasts or IGMP. Consequently, ICMPv6 provides additional services to communicate issues on the network.

Both ICMPv4 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**
Destination Unreachable		Echo Request/Reply
Time Exceeded		Router Solicitation
Parameter Problem		Router Advertisement

ICMPv6 messages

For both categories, each ICMP packet has a **Type**, **Code**, and **Checksum** field. 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

ICMP error messages report on network issues that prevent data from being delivered. The more commonly sent error messages are grouped into categories that have a specific purpose, as discussed here:

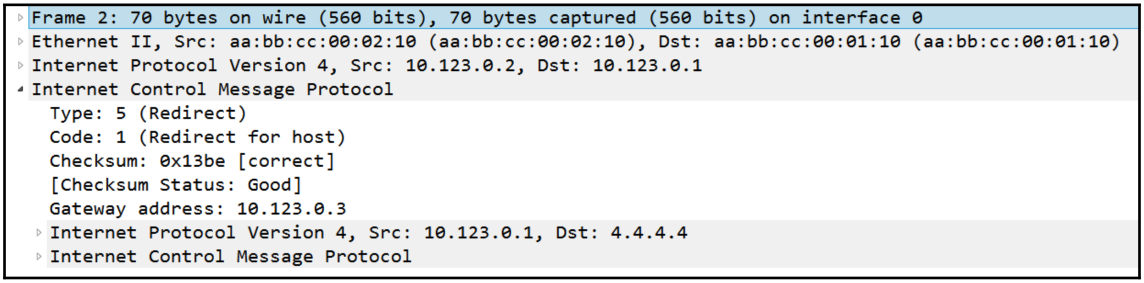
- **Destination unreachable** is where a router informs the host that the requested destination address can't be reached.
- **Time exceeded** is sent when the hop limit reaches zero.
- **Parameter problems** can be reported when there is an issue in determining a field value in the header or extension header.

As discussed, ICMPv6 has many of the same functions as ICMP, but also provides additional roles to support IPv6. As a result, when reporting errors, you may see this on an IPv6 network:

- **Packet too big** is sent when a device cannot send the data, as the packet is larger than the **Maximum Transmission Unit** (**MTU**) of the outgoing link.

In either version, when there is an error, ICMP helps reconstruct the possible reasons why the data did not get to its final destination. If appropriate, the sending host will adjust the payload so that the data can be delivered successfully.

When an error message is sent, the message includes the header, as well as the IP header, and then the first eight bytes (or 64 bits) of the original datagram that caused the error, as shown in this screenshot:



```
▷ Frame 2: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
▷ Ethernet II, Src: aa:bb:cc:00:02:10 (aa:bb:cc:00:02:10), Dst: aa:bb:cc:00:01:10 (aa:bb:cc:00:01:10)
▷ Internet Protocol Version 4, Src: 10.123.0.2, Dst: 10.123.0.1
◢ Internet Control Message Protocol
    Type: 5 (Redirect)
    Code: 1 (Redirect for host)
    Checksum: 0x13be [correct]
    [Checksum Status: Good]
    Gateway address: 10.123.0.3
  ▷ Internet Protocol Version 4, Src: 10.123.0.1, Dst: 4.4.4.4
  ▷ Internet Control Message Protocol
```

ICMP redirect message

Within the redirect message, you see an IP header, and then an ICMP message followed by the first eight bytes (or 64 bits) of the original IP and ICMP header.

Common error messages include destination unreachable and redirection. Errors are received and acted upon by TCP, IP, or user applications. In some cases, ICMP messages are ignored. However, some error messages must not be ignored, such as redirect messages, which will cause an automatic update to the host's routing table.

As we can see, ICMP error messages provide additional information so that the host can see exactly what happened. However, ICMP can also request and provide information, as discussed in the following section.

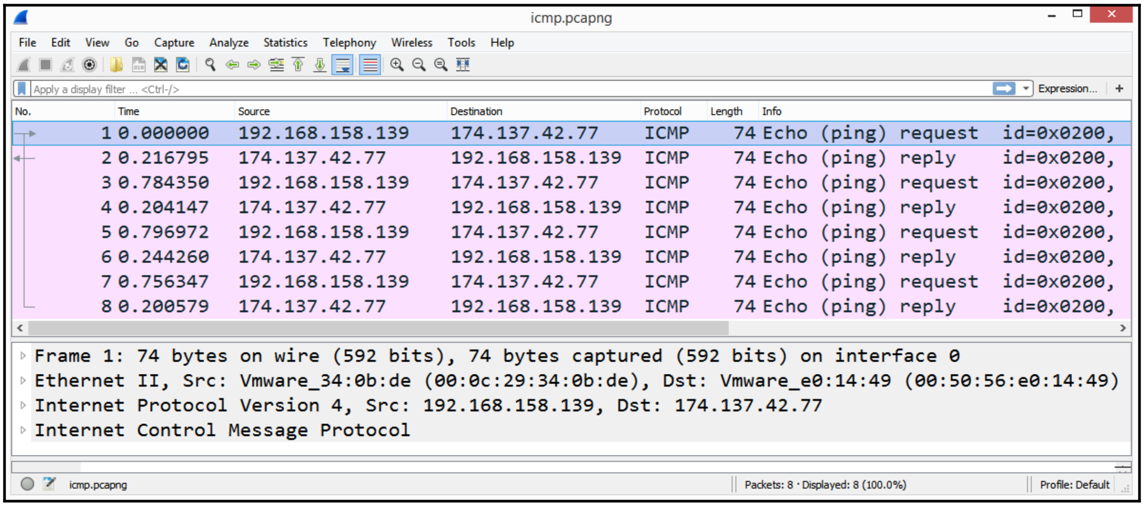
Issuing queries

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 requests and replies that are common to both ICMP and ICMPv6 are as follows:

- **Echo request/reply**: Tests for reachability
- **Router solicitation/advertisement**: Provides a way to solicit and receive router information that provides the IP addresses of that interface

One example is an echo request/reply, which is used in the ping network utility. To see an example, go to CloudShark, at `https://www.cloudshark.org/captures/fe65ed807bc3`, and open the file in Wireshark, as shown in the following screenshot:



```
No.      Time      Source            Destination       Protocol  Length Info
       1 0.000000  192.168.158.139   174.137.42.77     ICMP      74 Echo (ping) request  id=0x0200,
       2 0.216795  174.137.42.77     192.168.158.139   ICMP      74 Echo (ping) reply    id=0x0200,
       3 0.784350  192.168.158.139   174.137.42.77     ICMP      74 Echo (ping) request  id=0x0200,
       4 0.204147  174.137.42.77     192.168.158.139   ICMP      74 Echo (ping) reply    id=0x0200,
       5 0.796972  192.168.158.139   174.137.42.77     ICMP      74 Echo (ping) request  id=0x0200,
       6 0.244260  174.137.42.77     192.168.158.139   ICMP      74 Echo (ping) reply    id=0x0200,
       7 0.756347  192.168.158.139   174.137.42.77     ICMP      74 Echo (ping) request  id=0x0200,
       8 0.200579  174.137.42.77     192.168.158.139   ICMP      74 Echo (ping) reply    id=0x0200,

▷ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▷ Ethernet II, Src: Vmware_34:0b:de (00:0c:29:34:0b:de), Dst: Vmware_e0:14:49 (00:50:56:e0:14:49)
▷ Internet Protocol Version 4, Src: 192.168.158.139, Dst: 174.137.42.77
▷ Internet Control Message Protocol
```

ICMP echo request/reply

ICMPv6 needs to provide more specific information to assist IPv6 in delivering data, as we'll see next.

Providing information using ICMPv6

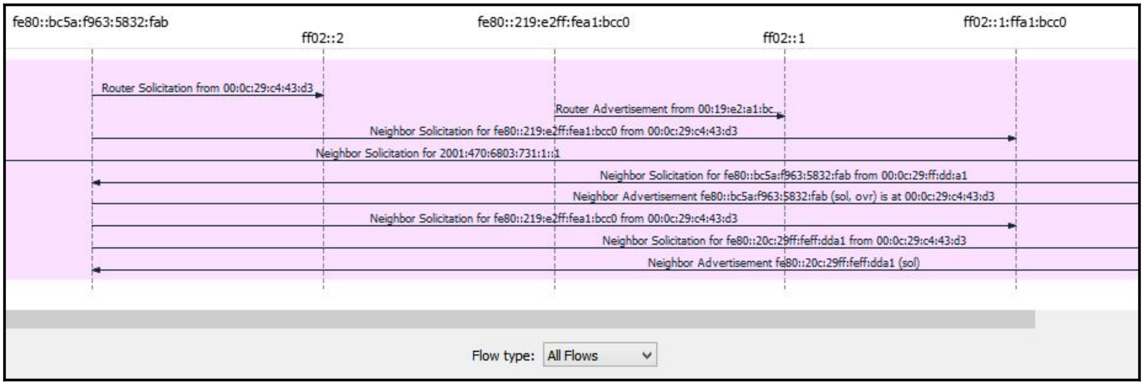
IPv6 no longer uses ARP broadcasts or IGMP. As a result, ICMPv6 provides additional services to communicate issues on the network, which include the following:

- **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.

To see an example of the many ICMPv6 messages communicating to other devices on the network, go to CloudShark (`https://www.cloudshark.org/captures/fe65ed807bc3`) and then download and open the file in Wireshark. Create a flow graph by completing the following steps:

- In the display filter, enter `icmpv6` and press *Enter* to run the filter.
- Go to **Statistics**, and then **Flow Graph**.
- Once open, go to the lower left-hand corner and select the **Limit to Display** filter.

The results are as shown in the following screenshot, where I have zoomed in to show the transactions:



Router Solicitation from 00:0c:29:c4:43:d3

Router Advertisement from 00:19:e2:a1:bc...

Neighbor Solicitation for fe80::219:e2ff:fea1:bcc0 from 00:0c:29:c4:43:d3

Neighbor Solicitation for 2001:470:6803:731:1::1

Neighbor Solicitation for fe80::bc5a:f963:5832:fab from 00:0c:29:ff:dd:a1

Neighbor Advertisement fe80::bc5a:f963:5832:fab (sol, ovr) is at 00:0c:29:c4:43:d3

Neighbor Solicitation for fe80::219:e2ff:fea1:bcc0 from 00:0c:29:c4:43:d3

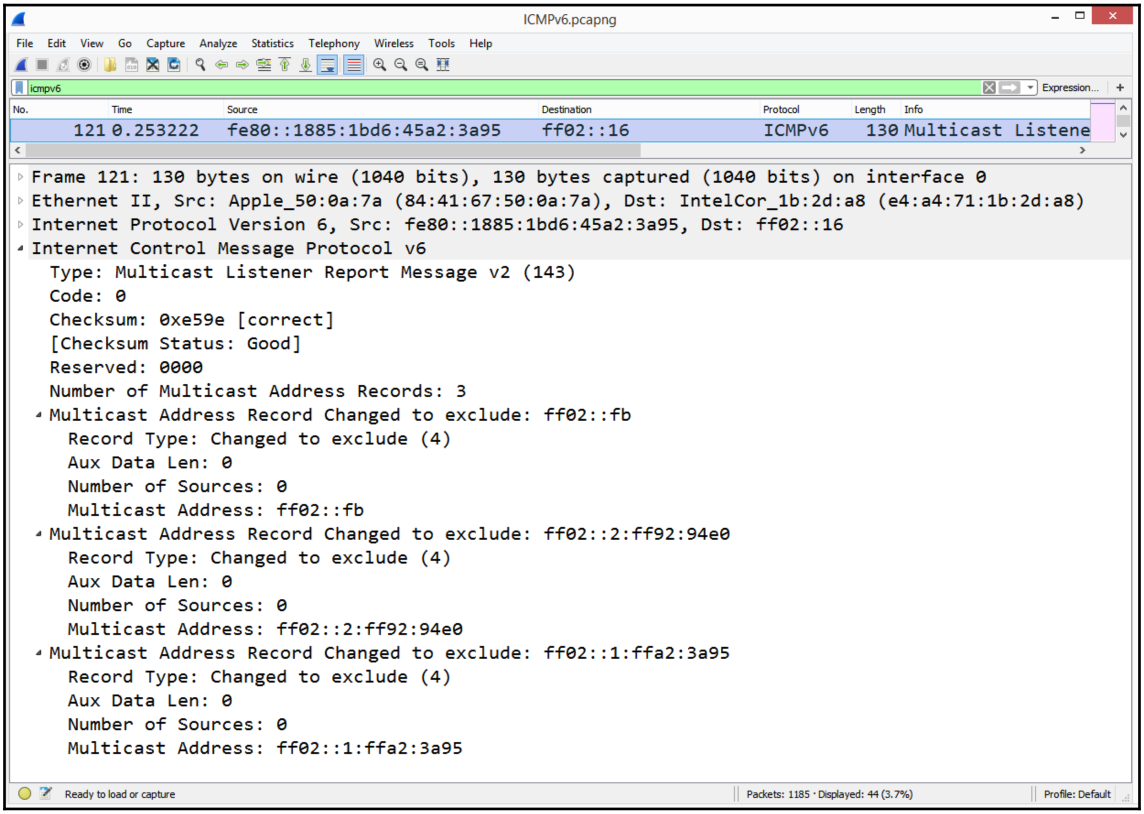
Neighbor Solicitation for fe80::20c:29ff:feff:dda1 from 00:0c:29:c4:43:d3

Neighbor Advertisement fe80::20c:29ff:feff:dda1 (sol)

Flow type: All Flows

ICMPv6 flow graph: zoomed-in

Some of the ICMPv6 reports have additional details. For example, in the following screenshot, we can see the details provided in a single report:


ICMPv6.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmpv6
Expression...
Ready to load or capture
Packets: 1185 · Displayed: 44 (3.7%)
Profile: Default

```
No.  Time      Source                     Destination  Protocol  Length  Info
121  0.253222  fe80::1885:1bd6:45a2:3a95  ff02::16     ICMPv6    130     Multicast Listene
```

```
▹ Frame 121: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
▹ Ethernet II, Src: Apple_50:0a:7a (84:41:67:50:0a:7a), Dst: IntelCor_1b:2d:a8 (e4:a4:71:1b:2d:a8)
▹ Internet Protocol Version 6, Src: fe80::1885:1bd6:45a2:3a95, Dst: ff02::16
◂ Internet Control Message Protocol v6
    Type: Multicast Listener Report Message v2 (143)
    Code: 0
    Checksum: 0xe59e [correct]
    [Checksum Status: Good]
    Reserved: 0000
    Number of Multicast Address Records: 3
  ◂ Multicast Address Record Changed to exclude: ff02::fb
      Record Type: Changed to exclude (4)
      Aux Data Len: 0
      Number of Sources: 0
      Multicast Address: ff02::fb
  ◂ Multicast Address Record Changed to exclude: ff02::2:ff92:94e0
      Record Type: Changed to exclude (4)
      Aux Data Len: 0
      Number of Sources: 0
      Multicast Address: ff02::2:ff92:94e0
  ◂ Multicast Address Record Changed to exclude: ff02::1:ffa2:3a95
      Record Type: Changed to exclude (4)
      Aux Data Len: 0
      Number of Sources: 0
      Multicast Address: ff02::1:ffa2:3a95
```

ICMPv6 multicast listener report

As you can see, ICMPv6 is a powerful protocol. In addition to error and information messages, ICMPv6 provides additional information on IPv6 router and host configuration.

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
- **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. Some of these code values are as follows:

- **Code 0**: Net unreachable
- **Code 1**: Host unreachable
- **Code 2**: Protocol unreachable
- **Code 3**: Port unreachable
- **Code 4**: Fragmentation needed and Don't fragment was Set

Type 9 router advertisement only has two code values:

- **Code 0**: Normal router advertisement
- **Code 16**: Does not route common traffic

In the next section, we'll review the type and code values for ICMPv6.

Defining ICMPv6 type and code values

Along with ICMP for IPv4, ICMPv6 is used to communicate updates or error messages and has its own set of types to identify messages. Because ICMPv6 provides additional information on IPv6 router and host configuration, you'll find specific type values that help provide this information.

A shortlist 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. If the type does not have a corresponding code value, the code value will be set to 0, as shown in the *ICMPv6 multicast listener report* screenshot.

Let's look at the following examples.

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 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 since, over time, they have been found to be ineffective and are considered deprecated, such as these examples:

- **Type 33**: IPv6 Where-are-you (deprecated)
- **Type 34**: IPv6 I-am-here (deprecated)
- **Type 35**: Mobile registration request (deprecated)

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 `https://www.iana.org/`:

- For ICMP: `https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml`
- For ICMPv6: `https://www.iana.org/assignments/icmpv6-parameters/icmpv6-parameters.xhtml`

Now that you have seen how ICMP works, you should also see that ICMP can provide a great deal of information on a network and devices. As a result, it's important to understand that this protocol can be used in malicious ways, and that the firewall rules should be tuned to prevent malicious activity, as outlined in the next section.

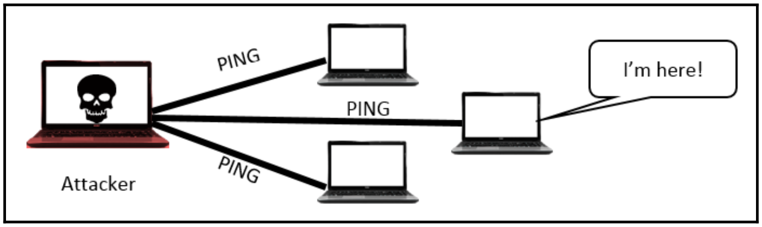
Configuring firewall rules

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. Then, we'll evaluate some of the firewall rules.

First, let's start with an overview of a ping sweep, which is used to see which network hosts might be awake.

Sending malicious ping sweeps

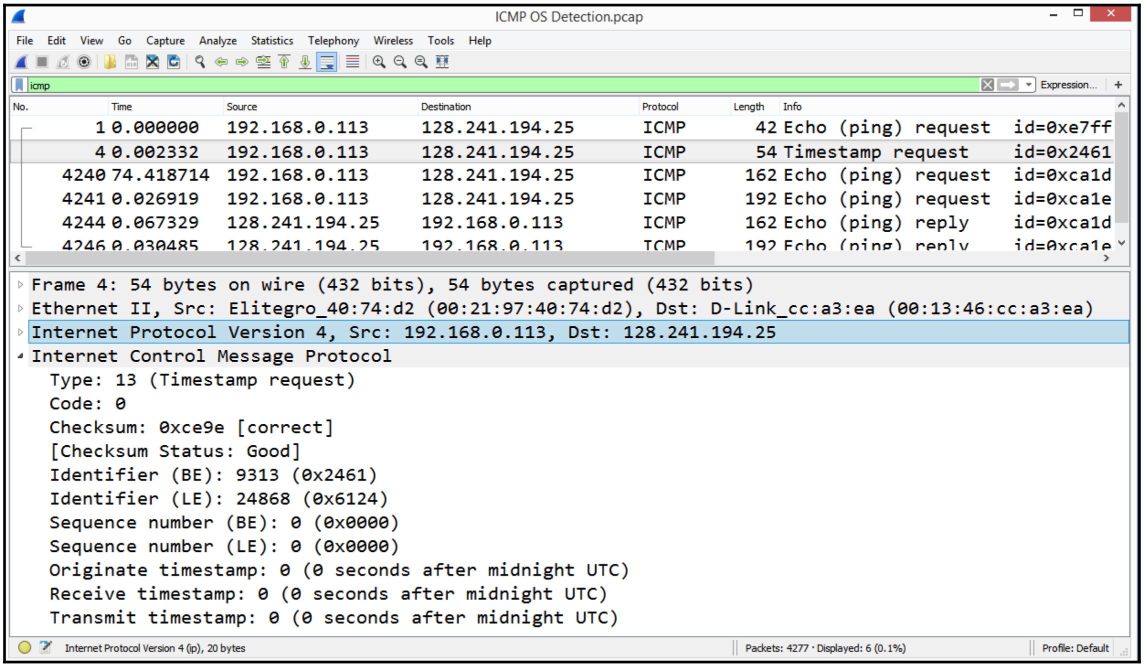
ICMP can be used as an effective scanning tool as it can determine a great deal of information about a network. Malicious actors use various techniques to scan a network for vulnerable hosts. Using ICMP can determine which hosts are alive and responding:



ICMP ping sweep

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. Once a responding host is identified, the hacker will send more advanced probes to obtain additional information.

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. For example, in the following screenshot, an ICMP timestamp request is sent in the hope of getting a reply to help the software rule out different OSes:



```
No.   Time        Source          Destination     Protocol  Length Info
    1 0.000000    192.168.0.113   128.241.194.25  ICMP          42 Echo (ping) request   id=0xe7ff
    4 0.002332    192.168.0.113   128.241.194.25  ICMP          54 Timestamp request     id=0x2461
 4240 74.418714   192.168.0.113   128.241.194.25  ICMP         162 Echo (ping) request   id=0xca1d
 4241 0.026919    192.168.0.113   128.241.194.25  ICMP         192 Echo (ping) request   id=0xca1e
 4244 0.067329    128.241.194.25  192.168.0.113   ICMP         162 Echo (ping) reply     id=0xca1d
 4246 0.030485    128.241.194.25  192.168.0.113   ICMP         192 Echo (ping) reply     id=0xca1e
```

```
Frame 4: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
Ethernet II, Src: Elitegro_40:74:d2 (00:21:97:40:74:d2), Dst: D-Link_cc:a3:ea (00:13:46:cc:a3:ea)
Internet Protocol Version 4, Src: 192.168.0.113, Dst: 128.241.194.25
Internet Control Message Protocol
  Type: 13 (Timestamp request)
  Code: 0
  Checksum: 0xce9e [correct]
  [Checksum Status: Good]
  Identifier (BE): 9313 (0x2461)
  Identifier (LE): 24868 (0x6124)
  Sequence number (BE): 0 (0x0000)
  Sequence number (LE): 0 (0x0000)
  Originate timestamp: 0 (0 seconds after midnight UTC)
  Receive timestamp: 0 (0 seconds after midnight UTC)
  Transmit timestamp: 0 (0 seconds after midnight UTC)
```

Internet Protocol Version 4 (ip), 20 bytes
Packets: 4277 · Displayed: 6 (0.1%)
Profile: Default

ICMP timestamp request

As you can see, ICMP can be used to obtain information about a network and hosts. 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 in an active attack. As a result, a best practice is to block certain ICMP messages with an **Access Control List** (**ACL**) firewall, especially at border routers.

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 destination unreachable, along with the corresponding codes, which are type 3 for ICMP and type 1 for ICMPv6.

All other ICMP types are optional, depending on whether you would like to allow them on your network. Depending on your organization, some other types that are allowed may 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

Hopefully, 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 ICMPv4 and ICMPv6, which have similar functions; however, we identified how ICMPv6 has a bigger role. So that you can use this protocol while troubleshooting, we looked at ICMP messages that communicate with hosts to report on transmission errors, along with query messages that attempt to obtain information from a host.

To better understand the ICMP type and code values, we took a look at how they work in communicating information. In addition, we saw that there are some ICMP types that you will rarely see as they are now deprecated and/or not supported. By now, you should see that ICMP is a powerful protocol that helps to move traffic on a network, but we covered how ICMP can be used in malicious ways. As a result, you now understand the need to configure firewall rules that allow or deny specific types of ICMP traffic in order to reduce the threat of malicious ICMP traffic on the local area network.

In the next chapter, we will review ARP and begin with an overview of the role and purpose of ARP. So that you understand how ARP works and what an ARP packet looks like, we will cover an ARP transaction along with a closer look at ARP headers and fields. We will see the importance of a different type of ARP, called a Gratuitous ARP. 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, and then check your answers with those in *Assessment*:

1. ICMP communicates issues that prevent data delivery. Common issues include the network or port being _____.
    1. Stateful
    2. Inspected
    3. Unreachable
    4. 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 _____.
    1. Quenched
    2. Unreachable
    3. Digital
    4. 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.
    1. Time exceeded
    2. Parameter problems
    3. Packet too big
    4. Source quench

4. When setting firewall rules, the only essential ICMP traffic is type _____. The others are optional.
    1. 3
    2. 1
    3. 8
    4. 2
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.
    1. Payload
    2. Ping
    3. Port
    4. Checksum

13 Understanding ARP

We know that on a **Local Area Network** (**LAN**), we use a physical address or MAC address. When a packet is delivered from a website back to you on the LAN, how does the device find you when the packet only has an **Internet Protocol** (**IP**) address as an address? That is the responsibility of the **Address Resolution Protocol** (**ARP**), which resolves an IP address to a MAC address so that your packet gets delivered.

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 examine the different types of ARP 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.

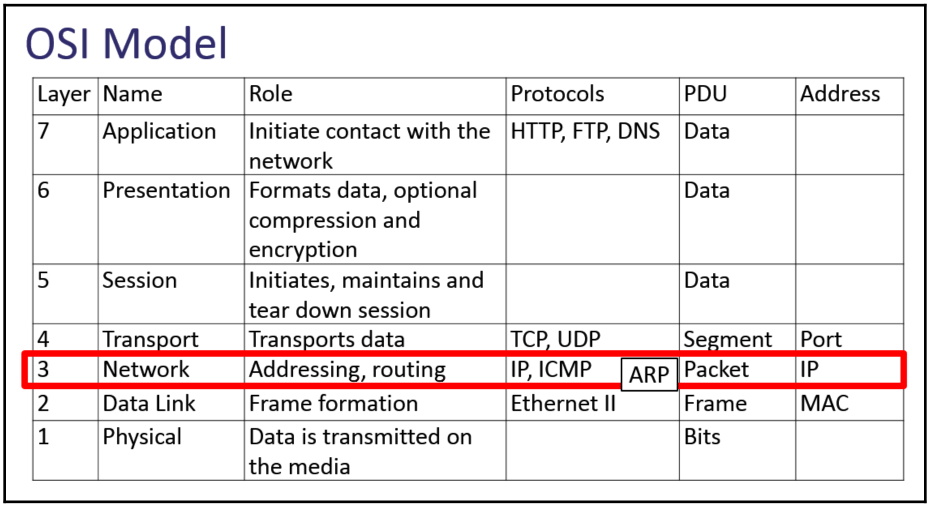
This chapter will cover the following:

- Understanding the role and purpose of ARP
- Exploring ARP headers and fields
- Examining the different types of ARP
- Analyzing ARP attacks along with some defense methods

Understanding the role and purpose of ARP

ARP resolves an IPv4 address to a MAC address on a LAN. ARP is one of the three main network layer protocols that includes ARP, IPv4, and 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, as ARP resolves an IP address (network layer) to a MAC address (data link layer). However, many consider ARP as a layer 3 protocol:



OSI Model

Layer	Name	Role	Protocols	PDU	Address
7	Application	Initiate contact with the network	HTTP, FTP, DNS	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 ARP	Packet	IP
2	Data Link	Frame formation	Ethernet II	Frame	MAC
1	Physical	Data is transmitted on the media		Bits	

The OSI model: network layer

Now that you can see where ARP resides in the OSI model, let's look at what happens within a LAN and how ARP does its job.

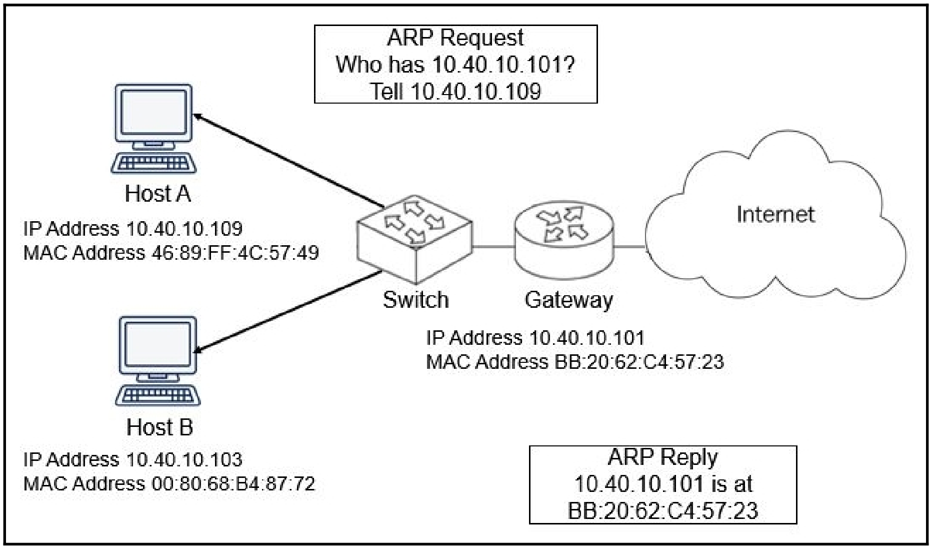
Resolving MAC addresses

ARP resolves an IP address to a MAC address on a LAN so that the frame can be delivered to the appropriate host. Now, let's step through why this is important.

When data travels through different networks, packets use a logical address or IP address along with routing to get data to its final destination. The IP provides addressing and routing to get data to its final destination. Once the data is at the desired network, the IP address is no longer needed. The reason is that on a LAN, the data link layer uses the MAC address of the destination machine, rather than the IP address.

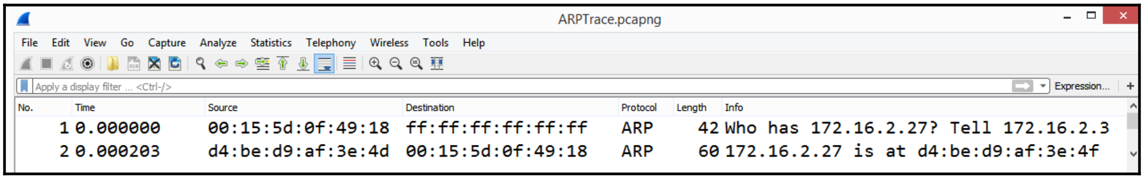
Therefore, to deliver the data to its final destination, a MAC or physical address is needed to place in the frame header. The device will first check its local cache, and if there is no entry, the device issues an ARP request (broadcast) and will wait for a reply.

As shown in the following diagram, **Host A** needs the MAC address for the gateway (which is the router interface for the LAN). **Host A** issues an ARP broadcast that asks who has the IP address `10.40.10.101`, tells `10.40.10.109`, and waits for a reply. The gateway then sends an ARP reply that lets the host know that `10.40.10.101` is at MAC address `BB:20:62:C4:57:23`.



ARP broadcast on a network

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:


ARPTrace.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

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

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 MAC address is `00:15:5d:0f:49`, 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. As a result, when this ARP broadcast was sent, every host on the network received the frame. However, only one will respond.

A broadcast message is sent from one host to all devices on a network.

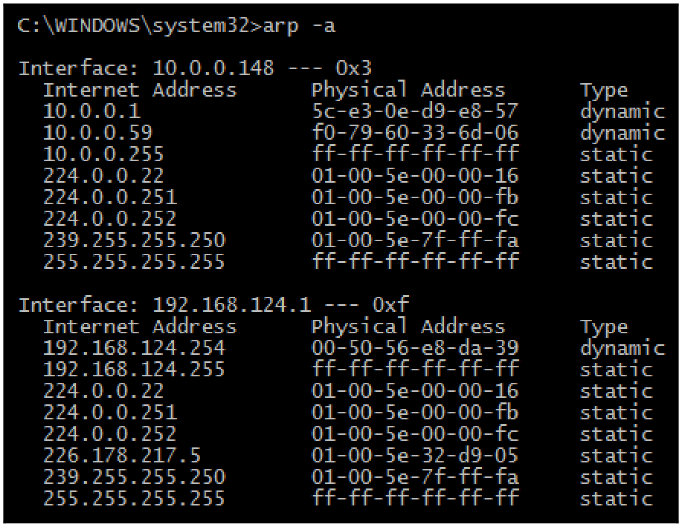
- In frame 2, the device identifies itself with an ARP reply. The source MAC address is `d4:be:d9:af:3e:4d`, which is the device with the IP address `172.16.2.27`, and the destination MAC address is `00:15:5d:0f:49`, which is the MAC address of the device requesting the resolution.

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 (or physical) address so the device has a MAC address that can be placed in the frame header in order for the data to be delivered. 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 Command Prompt. Then, enter `arp -a` to see 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
```

The arp -a command

As shown, the ARP cache table lists the following:

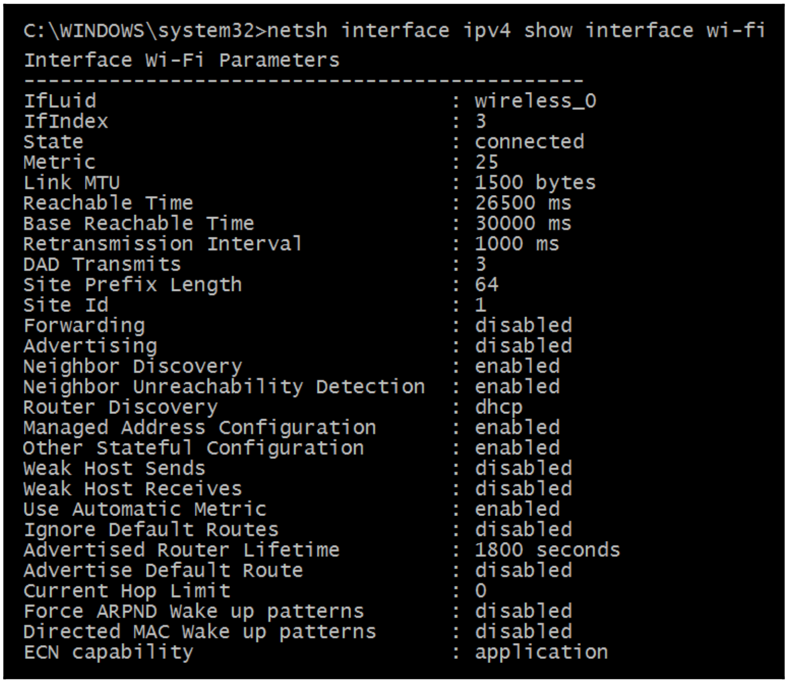
- **Internet Address**: The IP address
- **Physical Address**: The MAC address
- **Type**: Either static or dynamic

The ARP cache values will time out after a period of time. Once the timeout limit is reached, the entry will go away. If the OS needs the MAC address and there is no entry in the ARP table, it will need to issue a new ARP request.

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, where `NNN` is 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
```

netsh show interface

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.

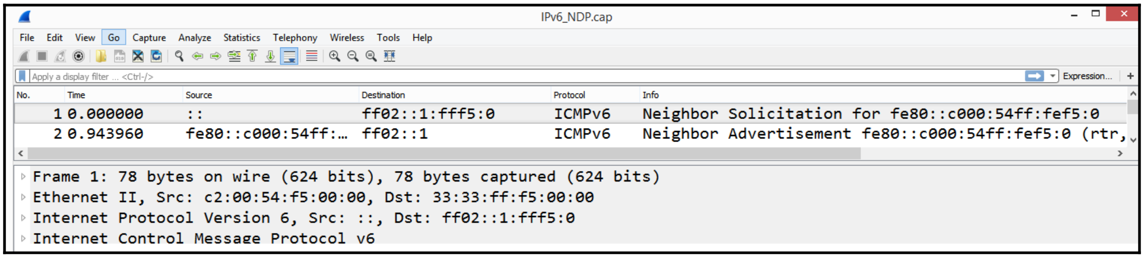
That means the ARP cache will out after 30 seconds. After that, if a MAC address is needed for a specific IP address, the system must send an ARP request out on the network.

We can see how ARP works in an IPv4 network, but what about an IPv6 network? The following section outlines how the **Neighbor Discovery Protocol** (**NDP**) takes the place of ARP in an IPv6 network.

Replacing ARP with NDP in IPv6

ARP is essential in an IPv4 network, but what happens in an IPv6 network? IPv6 doesn't use ARP. ARP is replaced with the NDP, which resolves an IP address to a MAC address.

To see an example of the NDP, go to `http://packetlife.net/captures/protocol/icmpv6/` and open the file in Wireshark. As shown in the following screenshot, the first packet in the `IPv6_NDP.pcap` trace file is a **Neighbor Solicitation** (**NS**), followed by a neighbor advertisement:


IPv6_NDP.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.  Time        Source            Destination        Protocol  Info
   1 0.000000    ::                ff02::1:fff5:0     ICMPv6    Neighbor Solicitation for fe80::c000:54ff:fef5:0
   2 0.943960    fe80::c000:54ff:… ff02::1            ICMPv6    Neighbor Advertisement fe80::c000:54ff:fef5:0 (rtr,
```

```
Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
Ethernet II, Src: c2:00:54:f5:00:00, Dst: 33:33:ff:f5:00:00
Internet Protocol Version 6, Src: ::, Dst: ff02::1:fff5:0
Internet Control Message Protocol v6
```

Example of the 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 doing an analysis on 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 the field values.

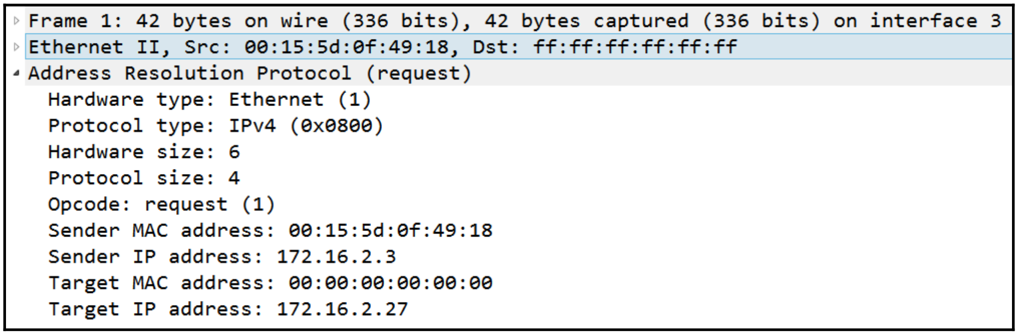
Exploring ARP headers and fields

In a trace file, if you use 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 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

In the `ARPTrace` trace file, 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 IP or transport layer headers.

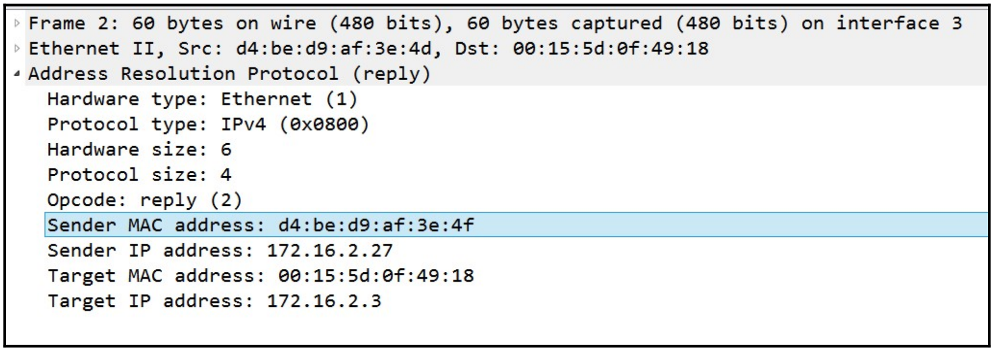
Expand frame 1 to see a standard ARP request that has several field values, which provide information about the transaction, as shown here:



```
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
```

Frame 1: ARP request

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
```

Frame 2: ARP reply

Once the ARP reply is received, the MAC address is resolved. As shown in both the *Frame 1: ARP request* and *Frame 2: ARP reply* screenshots, there are several field values, which we'll review next.

Breaking down the ARP header fields

Within an ARP header, there are several values that provide information on the ARP transaction, as outlined in the following list:

- **Hardware type**: This lists the type of connection for the session. In frame 1, the hardware type is listed as **Ethernet (1)**, which is common in today's networks. However, there are other types, such as **IPsec tunnel (31)** and **Fiber Channel (18)**, as shown in the list found at `https://www.iana.org/assignments/arp-parameters/arp-parameters.xhtml#arp-parameters-2`.
- **Protocol type**: This lists the internetworking protocol in use for the session. In frame 1, the protocol type is listed as **IPv4 0x0800**, which is standard on today's networks.
- **Hardware size**: The number of bytes of a hardware address. *Frame 1: ARP request* lists **Hardware size: 6**. A MAC address is 6 bytes or 48 bits, which is a standard MAC address length.
- **Protocol size**: This lists the bytes in the IP address. Frame 1 lists **Protocol size: 4**. An IPv4 address is 4 bytes or 32 bits, which is the length of an IPv4 address.
- **Opcode**: This lists what operation the sender is executing. Although there are many, as listed at `https://www.iana.org/assignments/arp-parameters/arp-parameters.xhtml#arp-parameters-1`, the opcode is most likely **request (1)**, as shown in the *Frame 1: ARP request* screenshot, or **reply (2)**, as shown in the *Frame 2: ARP reply* screenshot.
- **Sender MAC address**: Frame 1 lists the sender MAC address as `00:15:5d:0f:49:18`, which is the MAC address of the host sending the request.
- **Sender IP address**: This is the network address of the sender. Frame 1 lists `172.16.2.3` as the sender's IP address.
- **Target MAC address**: This is the MAC address of the target. In frame 1, which is an ARP request, the target MAC is listed as all zeros, or `00:00:00:00:00:00`. There is no MAC address listed because the target MAC address is unknown.
- **Target IP address**: This is the network address of the target. Frame 1 lists **Target IP address: 172.16.2.27.**

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 request/replies:

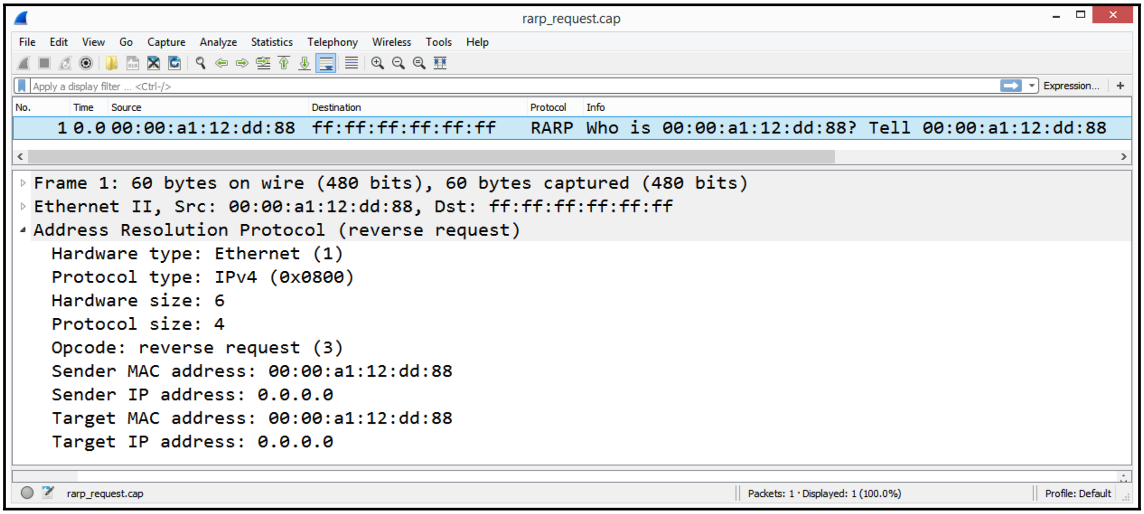
- A standard ARP request is a broadcast message that is sent out on the network requesting a resolution of an IP address to a MAC address.
- A standard ARP reply is a unicast message that is sent to the requesting host that provides the address resolution.

However, as we'll see next, 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. With RARP, a client requests its IPv4 address from a computer network by using its MAC address.

Using an example found at `https://wiki.wireshark.org/SampleCaptures?action=AttachFiledo=viewtarget=rarp_request.cap`, we can see the host requesting its IP address, as shown in the following screenshot:


rarp_request.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No.  Time Source             Destination        Protocol Info
   1 0.0  00:00:a1:12:dd:88  ff:ff:ff:ff:ff:ff  RARP     Who is 00:00:a1:12:dd:88? Tell 00:00:a1:12:dd:88
```

```
Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: 00:00:a1:12:dd:88, Dst: ff:ff:ff:ff:ff:ff
Address Resolution Protocol (reverse request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reverse request (3)
  Sender MAC address: 00:00:a1:12:dd:88
  Sender IP address: 0.0.0.0
  Target MAC address: 00:00:a1:12:dd:88
  Target IP address: 0.0.0.0
```

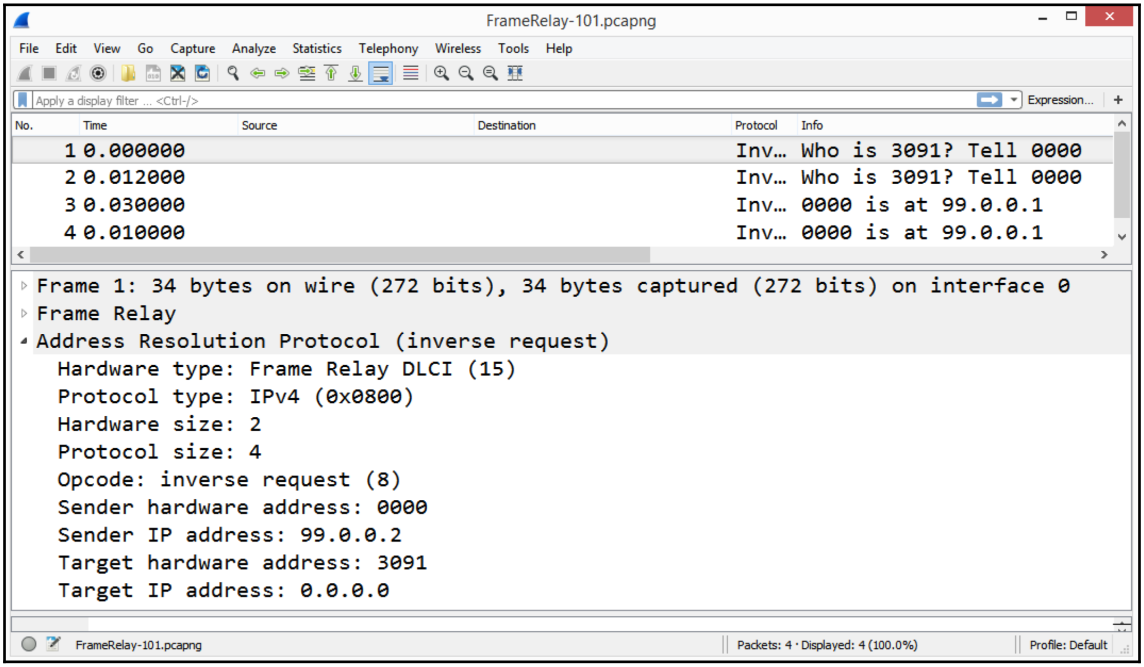
RARP

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)`. RARP is an obsolete protocol that has been replaced by more efficient protocols such as the **Dynamic Host Configuration Protocol** (**DHCP**).

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 it in Wireshark, as shown here:

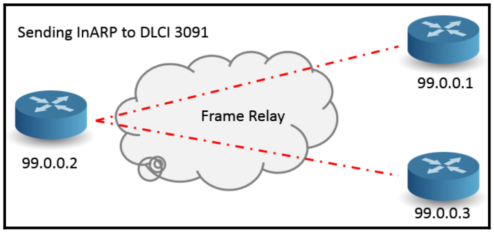

FrameRelay-101.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+
FrameRelay-101.pcapng
Packets: 4 · Displayed: 4 (100.0%)
Profile: Default

```
No.   Time       Source       Destination      Protocol  Info
   1 0.000000                                  Inv…      Who is 3091? Tell 0000
   2 0.012000                                  Inv…      Who is 3091? Tell 0000
   3 0.030000                                  Inv…      0000 is at 99.0.0.1
   4 0.010000                                  Inv…      0000 is at 99.0.0.1
```

```
▹ Frame 1: 34 bytes on wire (272 bits), 34 bytes captured (272 bits) on interface 0
▹ Frame Relay
◂ Address Resolution Protocol (inverse request)
    Hardware type: Frame Relay DLCI (15)
    Protocol type: IPv4 (0x0800)
    Hardware size: 2
    Protocol size: 4
    Opcode: inverse request (8)
    Sender hardware address: 0000
    Sender IP address: 99.0.0.2
    Target hardware address: 3091
    Target IP address: 0.0.0.0
```

InARP

InARP is used by the frame relay in the same way as a standard ARP in resolving an IP address. However, InARP does not use broadcasts as it already knows the target hardware address, which is the **Data Link Connection Identifier** (**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:



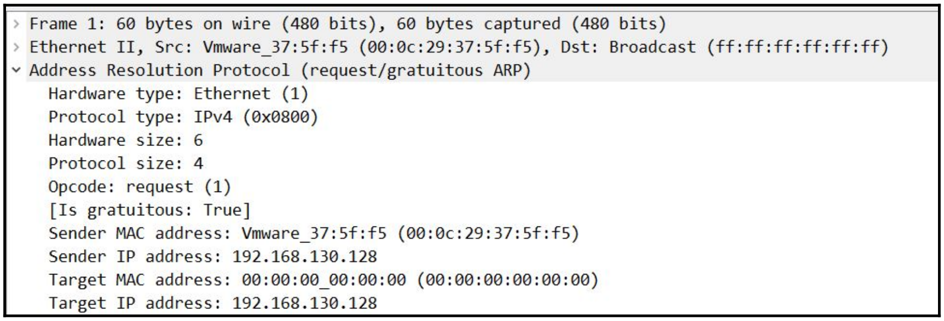
InARP over frame relay

While you may not see an InARP very often, while conducting analysis, you may see a type of ARP called a gratuitous ARP, as outlined next.

Issuing a gratuitous ARP

On a LAN, it's not uncommon to see a gratuitous ARP, which is an unsolicited ARP used to prevent duplicate IP addresses on a network, which can cause conflicts. To see an example, go to `https://www.cloudshark.org/captures/54af88021aa8`, and then download and open the file in Wireshark. Or, you can view the details in CloudShark.

In this example, we see a gratuitous ARP where the source and destination IP addresses are both set to the IP of the sending machine. As shown in the following screenshot, you can see **Sender IP address: 192.168.130.128**, which is the same as **Target IP address: 192.168.130.128**:



```
> Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
> Ethernet II, Src: Vmware_37:5f:f5 (00:0c:29:37:5f:f5), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
v Address Resolution Protocol (request/gratuitous ARP)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    [Is gratuitous: True]
    Sender MAC address: Vmware_37:5f:f5 (00:0c:29:37:5f:f5)
    Sender IP address: 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
```

Gratuitous ARP

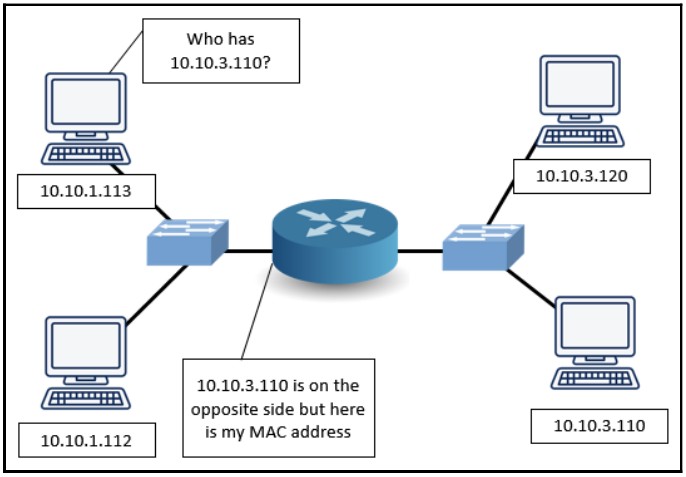
As shown, a gratuitous ARP request is sent as a broadcast; however, no reply is expected. This type of ARP is a way for a host to share IP and MAC address pairings so that all hosts on the network can update their ARP tables.

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 not actually an ARP type but a technique instead. Here are a few examples:

- We can use a proxy when a machine with a public IP address is in a private network behind a firewall. In this case, a way to resolve the MAC address is by using a proxy ARP, which conceals the existence of the hidden host behind the firewall and makes it appear as if it is in *front* of the firewall. The firewall uses a proxy ARP to and from the hidden device to maintain the illusion that the machine is on the public side.
- A proxy ARP can be used in a LAN when a host in one subnetwork is separated by a proxy router. When an ARP broadcast is sent to a host on another subnetwork, the router responds with its own MAC address and acts as a proxy to the host on the other subnetwork, as shown here:



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 and some defense methods.

Analyzing ARP attacks

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, there are 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.

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. Let's start with using ARP in a way that can trick or deceive hosts on a network.

Discovering ARP spoofing

We know that ARP resolves an IP address to a MAC address on a LAN, so the frame can be delivered to the appropriate host.

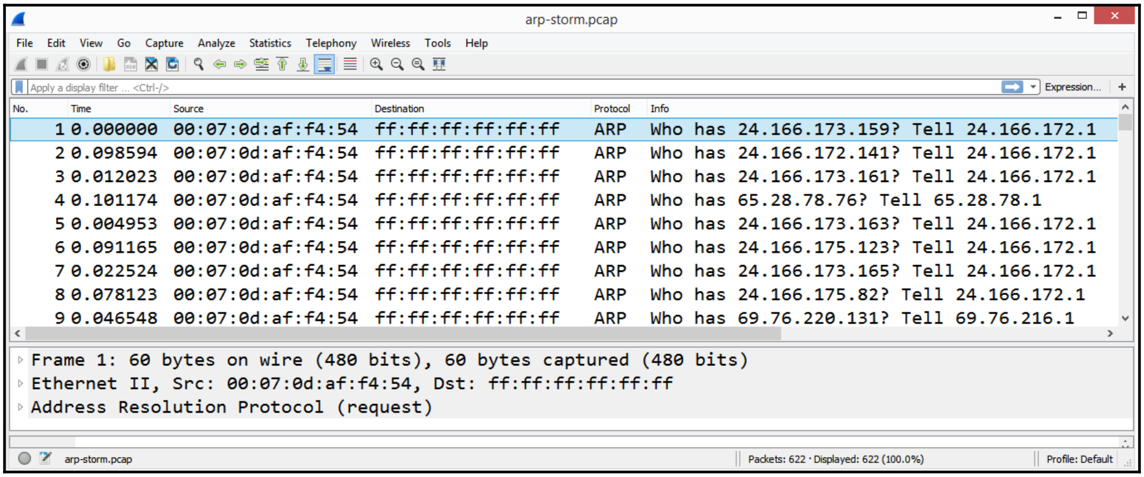
ARP spoofing is also known as an ARP cache poison and is used in man-in-the-middle attacks. The attacker will spoof its MAC address so instead of traffic going to the actual host, traffic will go to the host with the spoofed MAC address.

If a malicious actor redirects traffic, they can intercept traffic to obtain sensitive information, redirect traffic, or prepare for a more advanced attack.

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:


arp-storm.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.  Time      Source             Destination        Protocol  Info
  1 0.000000  00:07:0d:af:f4:54  ff:ff:ff:ff:ff:ff  ARP       Who has 24.166.173.159? Tell 24.166.172.1
  2 0.098594  00:07:0d:af:f4:54  ff:ff:ff:ff:ff:ff  ARP       Who has 24.166.172.141? Tell 24.166.172.1
  3 0.012023  00:07:0d:af:f4:54  ff:ff:ff:ff:ff:ff  ARP       Who has 24.166.173.161? Tell 24.166.172.1
  4 0.101174  00:07:0d:af:f4:54  ff:ff:ff:ff:ff:ff  ARP       Who has 65.28.78.76? Tell 65.28.78.1
  5 0.004953  00:07:0d:af:f4:54  ff:ff:ff:ff:ff:ff  ARP       Who has 24.166.173.163? Tell 24.166.172.1
  6 0.091165  00:07:0d:af:f4:54  ff:ff:ff:ff:ff:ff  ARP       Who has 24.166.175.123? Tell 24.166.172.1
  7 0.022524  00:07:0d:af:f4:54  ff:ff:ff:ff:ff:ff  ARP       Who has 24.166.173.165? Tell 24.166.172.1
  8 0.078123  00:07:0d:af:f4:54  ff:ff:ff:ff:ff:ff  ARP       Who has 24.166.175.82? Tell 24.166.172.1
  9 0.046548  00:07:0d:af:f4:54  ff:ff:ff:ff:ff:ff  ARP       Who has 69.76.220.131? Tell 69.76.216.1
```

```
▹ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
▹ Ethernet II, Src: 00:07:0d:af:f4:54, Dst: ff:ff:ff:ff:ff:ff
▹ Address Resolution Protocol (request)
```

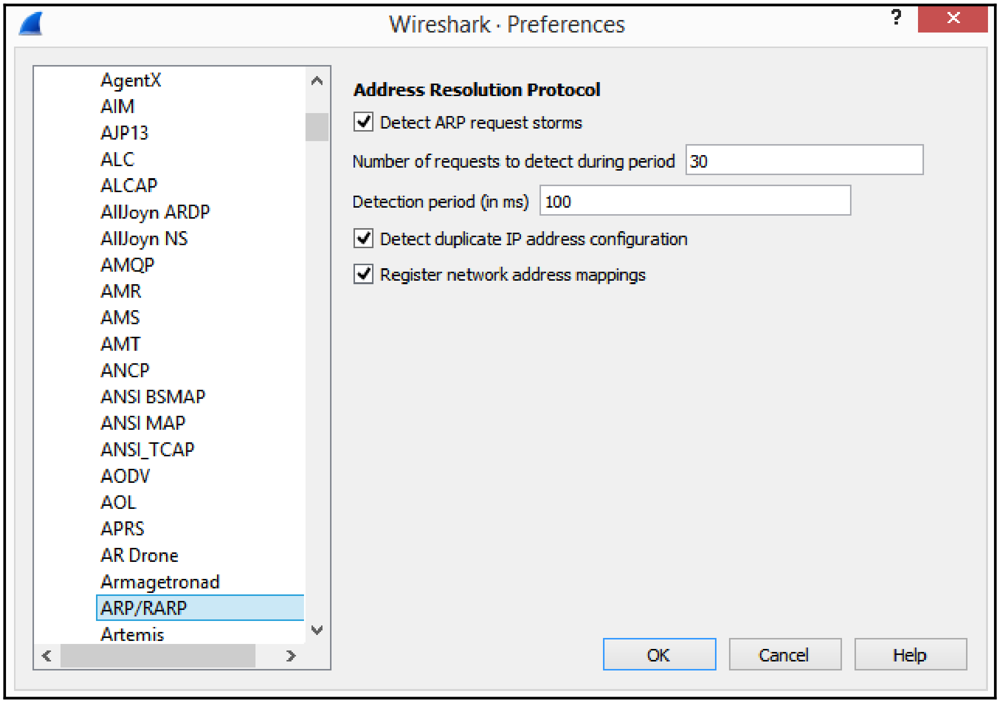
```
arp-storm.pcap    Packets: 622 · Displayed: 622 (100.0%)    Profile: Default
```

ARP storm

Let's step through how an ARP storm works:

- 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.
- An ARP storm floods the CAM table and overwhelms the switch with thousands of bogus entries.
- At that point, the switch simply acts as a hub and sends data out all of the ports, which can do the following:
    - Allow sniffing with the possible exposure of sensitive data.
    - Prevent 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, go to the ARP preferences by selecting an ARP header, right-click, and select **Protocol Preferences** | **Open Address Resolution Protocol Preferences**..., which will display the following screenshot:



Address Resolution Protocol

☑ Detect ARP request storms

Number of requests to detect during period: 30

Detection period (in ms): 100

☑ Detect duplicate IP address configuration

☑ Register network address mappings

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, as there are many available tools. Many of the tools 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`. However, Kali Linux can be used in malicious ways as well. Some of the tools 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 you can see, there can be 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 possible defensive strategies, so you do not fall victim to an ARP attack, include the following:

- **Intrusion Detection/Intrusion Prevention Systems (IDS/IPS)**: 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 an option, this isn't the best 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 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, as well as other methods, to detect malicious ARP behavior.

- **Secure Neighbor Discovery (SEND)**: On an IPv6 network, it is possible to use the SEND protocol. This is an extension of the NDP that provides authentication by using cryptographic techniques and reduces the ability to successfully launch an ARP spoofing attack on a LAN.

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. Then, we discussed the fact that, in addition to a standard ARP, you may encounter different types of ARP while doing analysis 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 this type of attack.

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 system, 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 *Assessment*:

1. On a LAN, the _____ layer uses the MAC address of the destination machine, rather than the IP address.
    1. Network
    2. Presentation
    3. Data link
    4. Transport

2. The opcode lists what operation the sender is executing. Although there are many, the most common opcodes are _____.
    1. 10 and 12
    2. 0 and 8
    3. 7 and 8
    4. 1 and 2
3. With _____, a client requests its IPv4 address from a computer network by using its MAC address.
    1. InARP
    2. RARP
    3. Gratuitous ARP
    4. Proxy ARP
4. The _____ is an unsolicited ARP used to prevent duplicate IP addresses on a network. No reply is expected.
    1. InARP
    2. RARP
    3. Gratuitous ARP
    4. Proxy ARP
5. An ARP_____ is a large number of ARP requests that creates a DoS attack and prevents the network from functioning normally.
    1. Storm
    2. Spoof
    3. Gratuitous
    4. Inverse

4

Section 4: Working with Packet Captures

This section examines ways to troubleshoot network latency issues, discover how to subset traffic, save and export captures, and how to use CloudShark.

This section is comprised of the following chapters:

- `Chapter 14`, *Troubleshooting Latency Issues*
- `Chapter 15`, *Subsetting, Saving, and Exporting Captures*
- `Chapter 16`, *Using CloudShark for Packet Analysis*

14
Troubleshooting 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 help, 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. Once you complete the chapter, you'll have gained a better understanding of 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 system, which subdivides the 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 the expert system

Analyzing latency issues

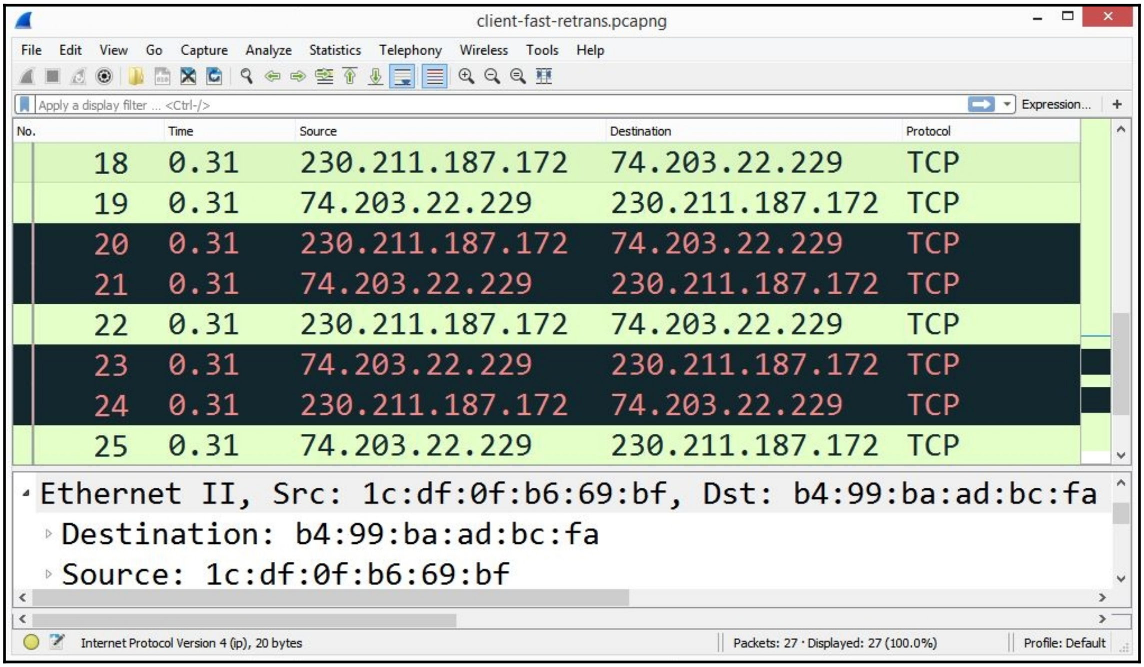
On today's networks, there are many different types of devices that communicate and exchange information. Applications and devices include **Unified Communication** (**UC**) systems, intermediary devices, the **Internet of Things** (**IoT**), mobile devices, and the many other types of traffic that are added to the network on a daily basis. As a result, there are multiple reasons that packet loss and slow response times occur. Once it's determined that there is an issue, the troubleshooting process begins.

With 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 and malware or 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. 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 capture 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
18	0.31	230.211.187.172	74.203.22.229	TCP
19	0.31	74.203.22.229	230.211.187.172	TCP
20	0.31	230.211.187.172	74.203.22.229	TCP
21	0.31	74.203.22.229	230.211.187.172	TCP
22	0.31	230.211.187.172	74.203.22.229	TCP
23	0.31	74.203.22.229	230.211.187.172	TCP
24	0.31	230.211.187.172	74.203.22.229	TCP
25	0.31	74.203.22.229	230.211.187.172	TCP

```
Ethernet II, Src: 1c:df:0f:b6:69:bf, Dst: b4:99:ba:ad:bc:fa
  Destination: b4:99:ba:ad:bc:fa
  Source: 1c:df:0f:b6:69:bf
```

Internet Protocol Version 4 (ip), 20 bytes | Packets: 27 · Displayed: 27 (100.0%) | Profile: Default

client-fast-retrans.pcap

As shown, both duplicate acknowledgments and fast retransmission are evident in packets 20-21 and 23-24. Both duplicate acknowledgments and fast retransmissions can occur on a network. However, when there is an excessive number of them, this is generally an indication of congestion.

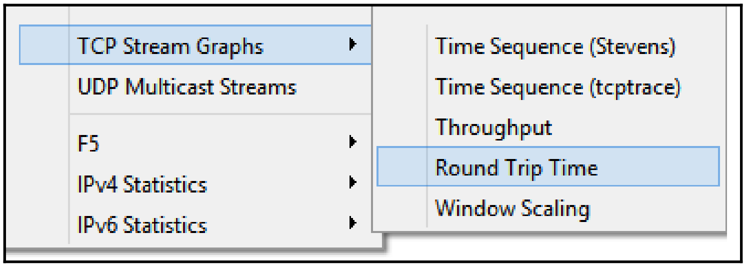
Generally, there are three indicators in measuring performance: latency, packet loss, and throughput. Let's take a look at each of these and how they can point to evidence of an unstable and sluggish network, beginning with latency.

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 the loading of web pages to slow down 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.

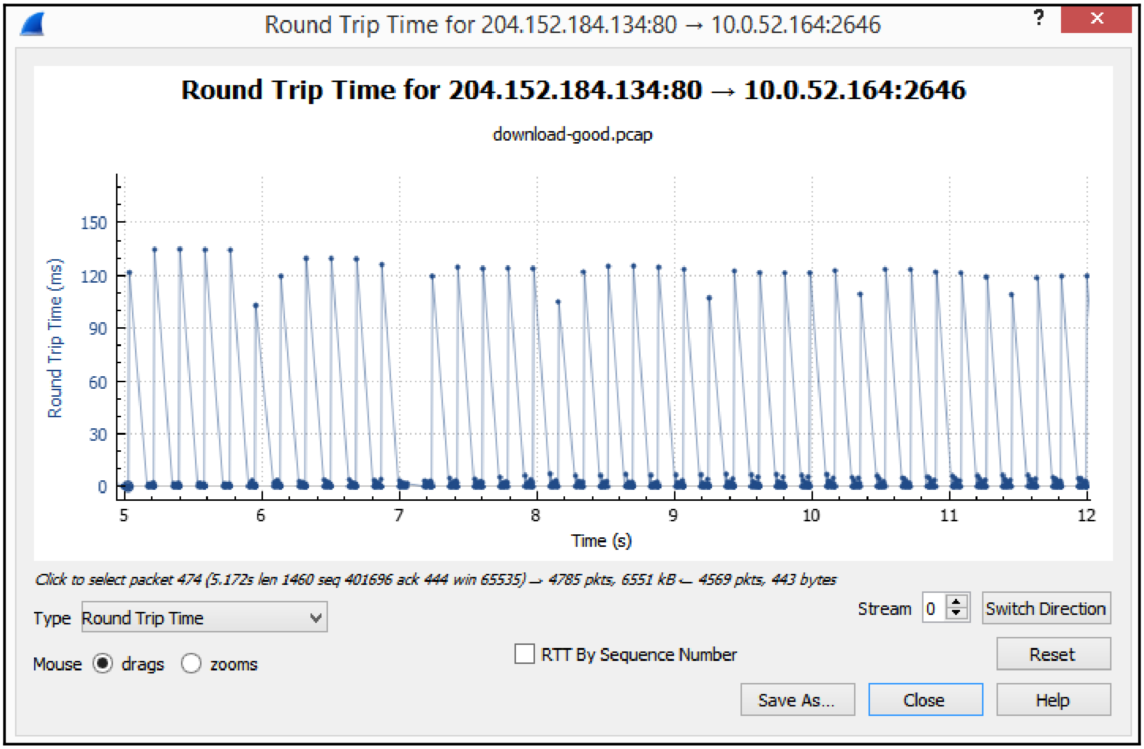
RTT can increase and vary during the course of transmission. To see the RTT for a particular stream in Wireshark, go to the menu and choose **Statistics** | **TCP Stream Graphs** | **Round Trip Time**, as shown here:



Stream Graphs menu

Once 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.

An optimal RTT remains steady, as shown in the following screenshot. However, you won't always see a steady RTT:


Round Trip Time for 204.152.184.134:80 → 10.0.52.164:2646
?
✕
Round Trip Time for 204.152.184.134:80 → 10.0.52.164:2646
download-good.pcap
Round Trip Time (ms)
150
120
90
60
30
0
5
6
7
8
9
10
11
12
Time (s)

Click to select packet 474 (5.172s len 1460 seq 401696 ack 444 win 65535) → 4785 pkts, 6551 kB ← 4569 pkts, 443 bytes

Type: Round Trip Time

Stream 0

Mouse ● drags ○ zooms

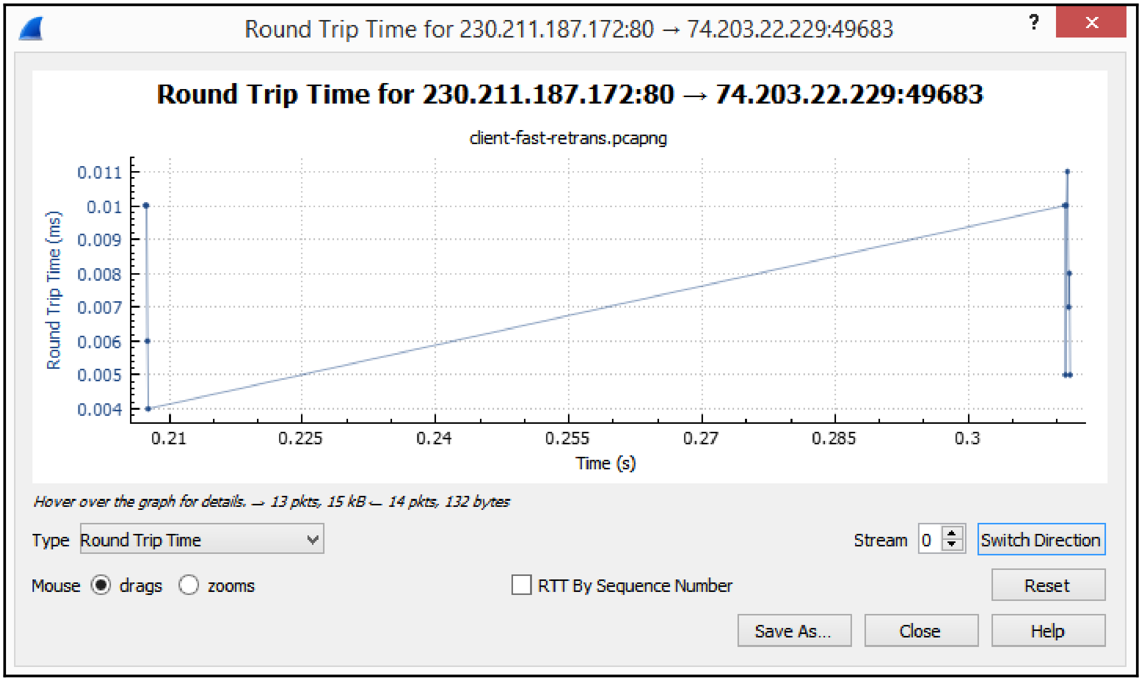
□ RTT By Sequence Number

Switch Direction
Reset
Save As...
Close
Help

Steady RTT

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:


Round Trip Time for 230.211.187.172:80 → 74.203.22.229:49683
?
✕
Round Trip Time for 230.211.187.172:80 → 74.203.22.229:49683
client-fast-retrans.pcapng
Round Trip Time (ms)
0.011
0.01
0.009
0.008
0.007
0.006
0.005
0.004
0.21
0.225
0.24
0.255
0.27
0.285
0.3
Time (s)
Switch Direction
Reset
Save As...
Close
Help

Hover over the graph for details. → 13 pkts, 15 kB ← 14 pkts, 132 bytes

Type Round Trip Time

Stream 0

Mouse ● drags ○ zooms

□ RTT By Sequence Number

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.

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. However, 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-alive, duplicate acknowledgments, and retransmissions.

Wireshark is capable of identifying many common transmission errors and can calculate delays in transmission and interruptions in the data flow by using time values. The following section provides an insight into the significance of time while doing analysis.

Learning the importance of time values

When doing analysis, time values can provide an insight into the delays in transmission. In Wireshark, there are choices as to how you want your time value displayed, which include the following:

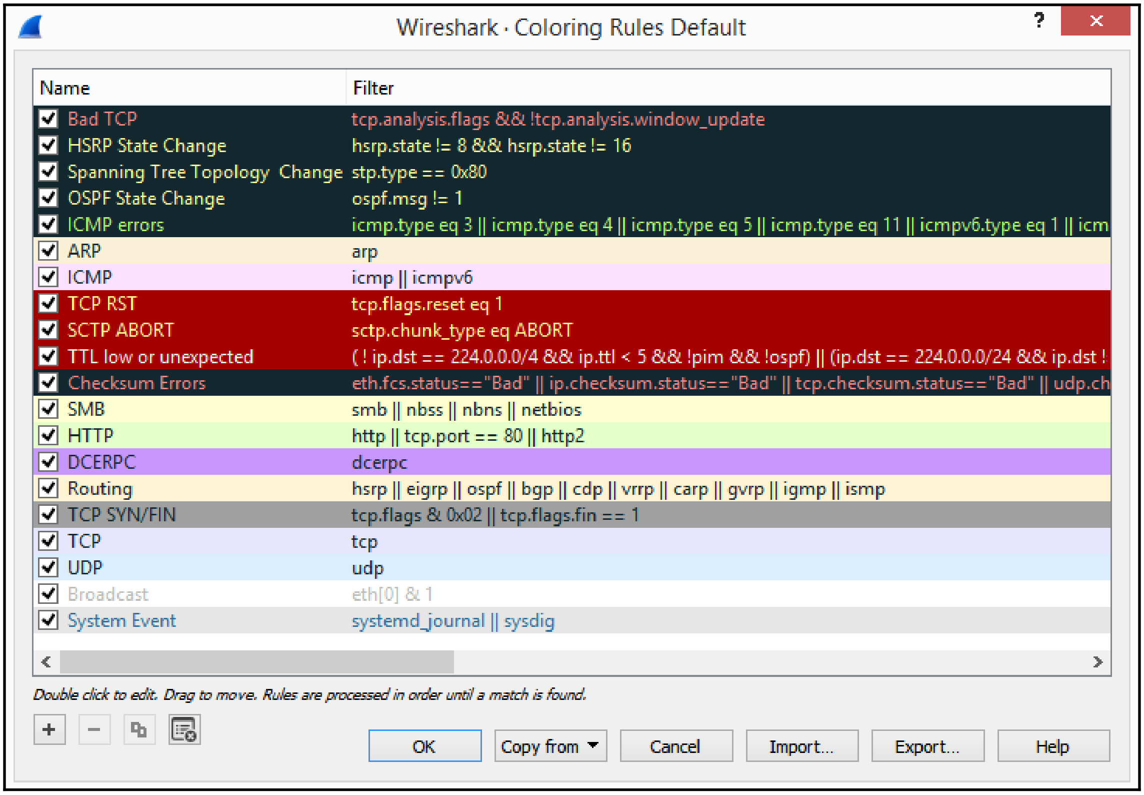
- **Seconds Since Beginning of Capture**
- **Seconds Since Previously Captured Packet**
- **Seconds Since Previously Displayed Packet**

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 the 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:



Name	Filter
☑ Bad TCP	tcp.analysis.flags && !tcp.analysis.window_update
☑ HSRP State Change	hsrp.state != 8 && hsrp.state != 16
☑ Spanning Tree Topology Change	stp.type == 0x80
☑ OSPF State Change	ospf.msg != 1
☑ ICMP errors	icmp.type eq 3 || icmp.type eq 4 || icmp.type eq 5 || icmp.type eq 11 || icmpv6.type eq 1 || icm
☑ ARP	arp
☑ ICMP	icmp || icmpv6
☑ TCP RST	tcp.flags.reset eq 1
☑ SCTP ABORT	sctp.chunk_type eq ABORT
☑ TTL low or unexpected	(! ip.dst == 224.0.0.0/4 && ip.ttl < 5 && !pim && !ospf) || (ip.dst == 224.0.0.0/24 && ip.dst !
☑ Checksum Errors	eth.fcs.status=="Bad" || ip.checksum.status=="Bad" || tcp.checksum.status=="Bad" || udp.ch
☑ SMB	smb || nbss || nbns || netbios
☑ HTTP	http || tcp.port == 80 || http2
☑ DCERPC	dcerpc
☑ Routing	hsrp || eigrp || ospf || bgp || cdp || vrrp || carp || gvrp || igmp || ismp
☑ TCP SYN/FIN	tcp.flags & 0x02 || tcp.flags.fin == 1
☑ TCP	tcp
☑ UDP	udp
☑ Broadcast	eth[0] & 1
☑ System Event	systemd_journal || sysdig

Double click to edit. Drag to move. Rules are processed in order until a match is found.

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`.

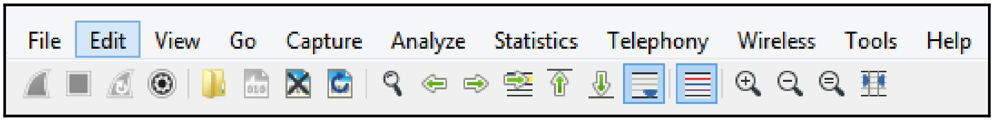
Each rule is processed until Wireshark finds a match, according to the order shown in the console. To modify the order of a particular rule, select the rule and then drag it to the desired position.

A check mark on the left-hand side indicates an active rule. To deactivate, deselect the rule you do not want Wireshark to consider.

To edit a rule, complete the following:

- Select and double-click the coloring rule you want to modify.
- You can then edit the name or the filter used, along with the background and foreground colors.

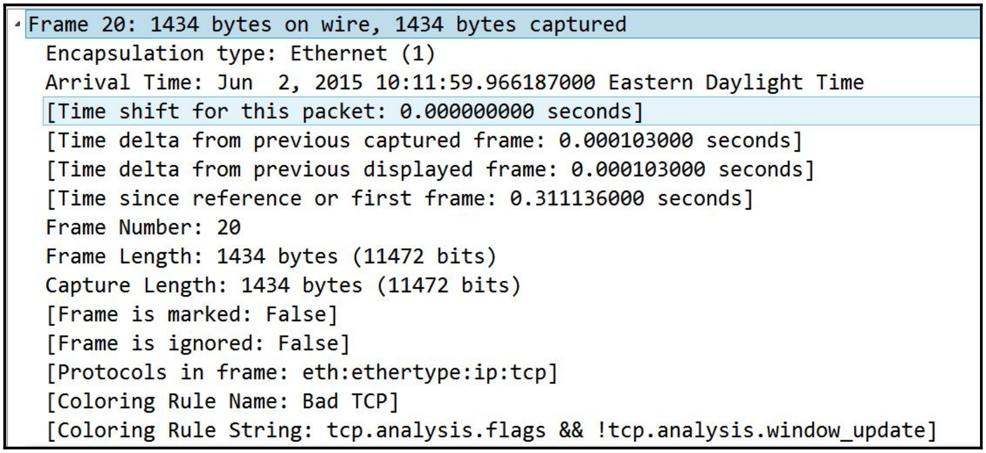
Although Wireshark can colorize packets, in some cases, the coloring can be distracting. You can disable the coloring rules by selecting the icon. The coloring rules icon is generally underneath the **Telephony** menu, as shown in the following screenshot. However, the position can vary in different versions, platforms, or layouts:



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 label for frame 20.
3. At the end of the metadata list, you will see the following:



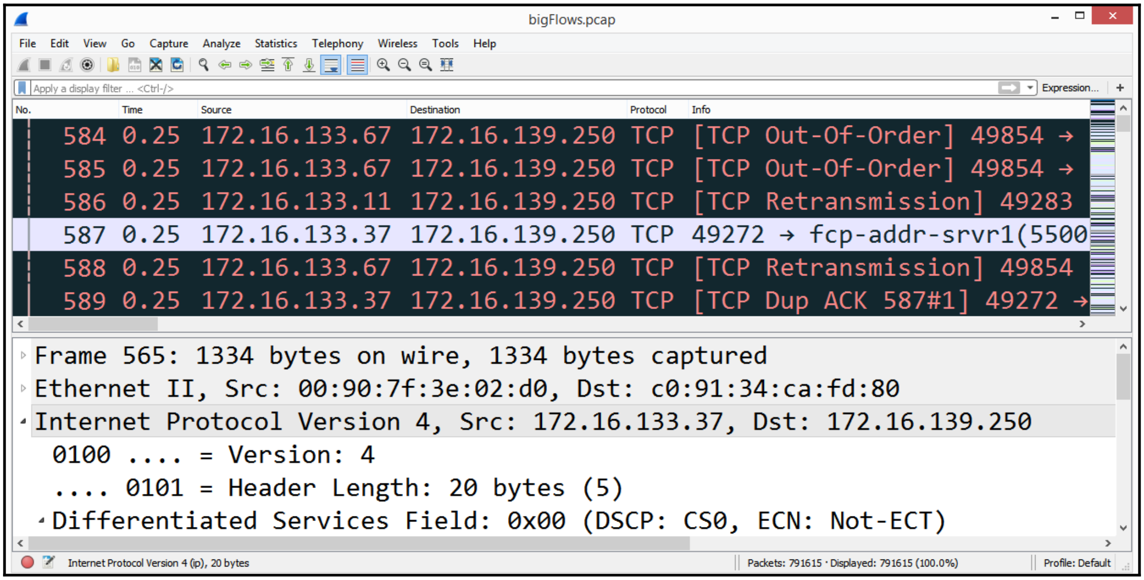
```
Frame 20: 1434 bytes on wire, 1434 bytes captured
  Encapsulation type: Ethernet (1)
  Arrival Time: Jun  2, 2015 10:11:59.966187000 Eastern Daylight Time
  [Time shift for this packet: 0.000000000 seconds]
  [Time delta from previous captured frame: 0.000103000 seconds]
  [Time delta from previous 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]
```

Coloring rules in the frame metadata

As you can see, the coloring rules provide guidelines on what traffic to home in on during analysis. For the coloring rules to work, they must be enabled; however, they are active in most cases. Next, let's take a look at how Wireshark incorporates the use of the coloring rules within the Intelligent Scrollbar.

Exploring the Intelligent Scrollbar

In addition to seeing indications of problems within a capture, you can also easily spot issues using the Intelligent Scrollbar, which is on the right-hand side of the packet list panel. In the following screenshot, we see indications of network congestion within the packet list:


bigFlows.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No.  Time  Source          Destination      Protocol  Info
584  0.25  172.16.133.67   172.16.139.250   TCP       [TCP Out-Of-Order] 49854 →
585  0.25  172.16.133.67   172.16.139.250   TCP       [TCP Out-Of-Order] 49854 →
586  0.25  172.16.133.11   172.16.139.250   TCP       [TCP Retransmission] 49283
587  0.25  172.16.133.37   172.16.139.250   TCP       49272 → fcp-addr-srvr1(5500
588  0.25  172.16.133.67   172.16.139.250   TCP       [TCP Retransmission] 49854
589  0.25  172.16.133.37   172.16.139.250   TCP       [TCP Dup ACK 587#1] 49272 →
```

```
▹ Frame 565: 1334 bytes on wire, 1334 bytes captured
▹ Ethernet II, Src: 00:90:7f:3e:02:d0, Dst: c0:91:34:ca:fd:80
◂ Internet Protocol Version 4, Src: 172.16.133.37, Dst: 172.16.139.250
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ◂ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
```

```
Internet Protocol Version 4 (ip), 20 bytes    Packets: 791615 · Displayed: 791615 (100.0%)    Profile: Default
```

bigFlows using coloring rules

The **Info** column header on the right-hand side lists several indications of trouble that warrants further investigation. These include the following:

- **[TCP Out-Of-Order]**
- **[TCP Retransmission]**
- **[TCP Dup ACK 587#1]**

In addition to the coloring in the packet list, on the right-hand side, there is a distinct coloring pattern based on the coloring rules set in the application, which is the Intelligent Scrollbar. 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 the Intelligent Scrollbar 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 the intermediary devices cause latency, delayed, and/or dropped packets and other negative effects. When troubleshooting errors in Wireshark, you will see evidence of transmission errors.

Some common indications include duplicate acknowledgments, keep-alive segments, and fast retransmissions.

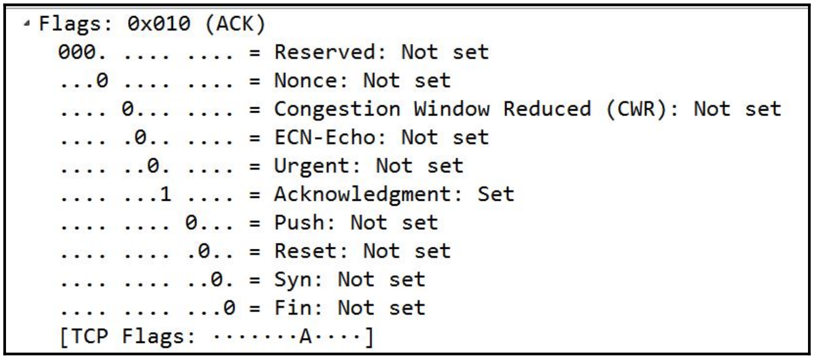
As this information may indicate latency and gaps in the delivery of data, it's important to understand the meaning of what the packets are trying to tell you. Let's start with an overview of duplicate acknowledgments.

Seeing duplicate acknowledgments

In a normal TCP conversation, the client acknowledges every byte received by transmitting an acknowledgment, with the ACK field value set as the next expected byte. When more than one acknowledgment is sent by the client (with the same ACK field value), this is said to be a duplicate acknowledgment.

To understand what a duplicate acknowledgment is, let's step through a standard TCP transaction:

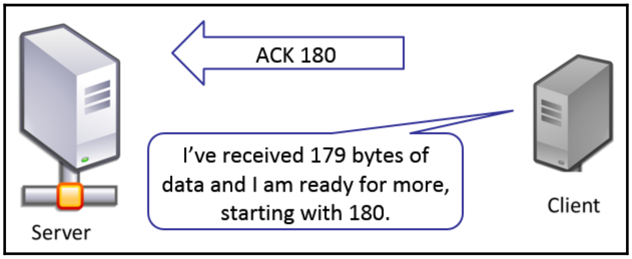
1. In the course of a normal TCP data transaction, TCP sequences and acknowledges every byte of data.
2. The client acknowledges the data received by setting the ACK flag in the TCP header, as shown here:



```
◢ 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····]
```

The TCP-ACK flag set

3. The client places a value of the next expected byte in the **Acknowledgment** field.
4. When the client sends an ACK 180 (acknowledgment number: 180) flag, 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:


ACK 180
I've received 179 bytes of data and I am ready for more, starting with 180.
Server
Client

Normal TCP acknowledgment

5. The server doesn't wait for confirmation of delivery to send more data. Instead, the data is sent concurrently with the acknowledgments.

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 acknowledgment, which means the client did not receive the next expected byte and is politely asking the server to send the data.

Take a look at the *bigFlows using coloring rules* screenshot, as shown in the *Exploring the Intelligent Scrollbar* section, and you will see a duplicate acknowledgment in frame 589. This indicates that the client is patiently re-requesting the missing data. In the **Info** column header, you will see **[TCP Dup ACK 587#1]**, which means this is the second (or duplicate) ACK flag sent after the original ACK sent in frame 587.

In the expert system, duplicate acknowledgments are under the category note, as shown in the screenshot, *Expert information grouped by severity*, in the *Discovering the expert system* section.

Latency and delays in transmission can be caused by any number of things, such as processing and queuing delays and general network congestion. As a result, duplicate acknowledgments may be sent over and over again by the client until it receives the expected data.

Another indication of transmission errors and congestion are keep-alive packets, which we will explore next.

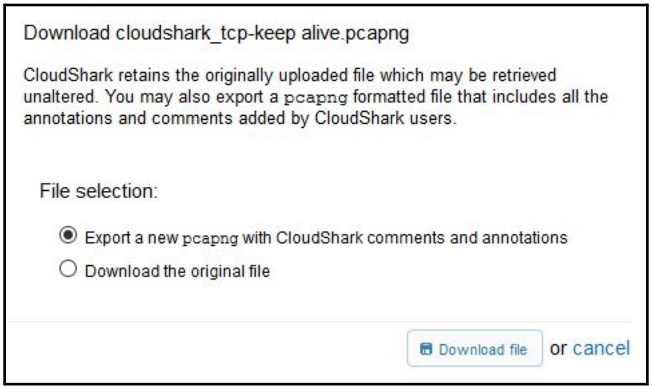
Observing keep-alive segments

Network congestion is part of today's landscape. Latency and delayed packets have many negative effects, such as slow web page retrieval. 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 that keeps a session alive instead of dropping the connection and having to go through the expensive negotiation of reestablishing the connection.

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 keep the session active and to verify that both sides are still responding.

If you would like to see an example of a keep-alive packet, go to `https://www.cloudshark.org/captures/5618ff446df8`. Once the page is open, select **Export**, which is found on the right-hand side of the interface, and then select **Export a new pcapng with CloudShark comments and annotations**, 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.

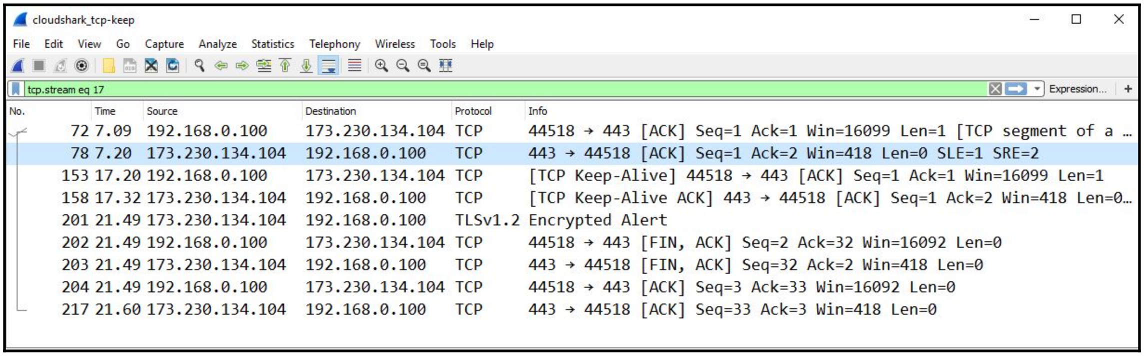
File selection:

● Export a new `pcapng` with CloudShark comments and annotations

○ Download the original file

Export file from CloudShark

Open the `cloushark_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:



cloudshark_tcp-keep
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression…

tcp.stream eq 17

```
No.   Time  Source          Destination     Protocol Info
   72 7.09  192.168.0.100   173.230.134.104 TCP      44518 → 443 [ACK] Seq=1 Ack=1 Win=16099 Len=1 [TCP segment of a …
   78 7.20  173.230.134.104 192.168.0.100   TCP      443 → 44518 [ACK] Seq=1 Ack=2 Win=418 Len=0 SLE=1 SRE=2
  153 17.20 192.168.0.100   173.230.134.104 TCP      [TCP Keep-Alive] 44518 → 443 [ACK] Seq=1 Ack=1 Win=16099 Len=1
  158 17.32 173.230.134.104 192.168.0.100   TCP      [TCP Keep-Alive ACK] 443 → 44518 [ACK] Seq=1 Ack=2 Win=418 Len=0…
  201 21.49 173.230.134.104 192.168.0.100   TLSv1.2  Encrypted Alert
  202 21.49 192.168.0.100   173.230.134.104 TCP      44518 → 443 [FIN, ACK] Seq=2 Ack=32 Win=16092 Len=0
  203 21.49 173.230.134.104 192.168.0.100   TCP      443 → 44518 [FIN, ACK] Seq=32 Ack=2 Win=418 Len=0
  204 21.49 192.168.0.100   173.230.134.104 TCP      44518 → 443 [ACK] Seq=3 Ack=33 Win=16092 Len=0
  217 21.60 173.230.134.104 192.168.0.100   TCP      443 → 44518 [ACK] Seq=33 Ack=3 Win=418 Len=0
```

HTTP keep-alive packets

I have removed the coloring so you can see the exchange of keep-alive in packets 153 and 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, which are messages between both endpoints to keep the session alive.

Therefore, in addition to seeing duplicate acknowledgments when there is network congestion, you may see also 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

Retransmissions, fast retransmissions, and spurious retransmissions are all related. However, each has subtle differences. The following bullets explain them:

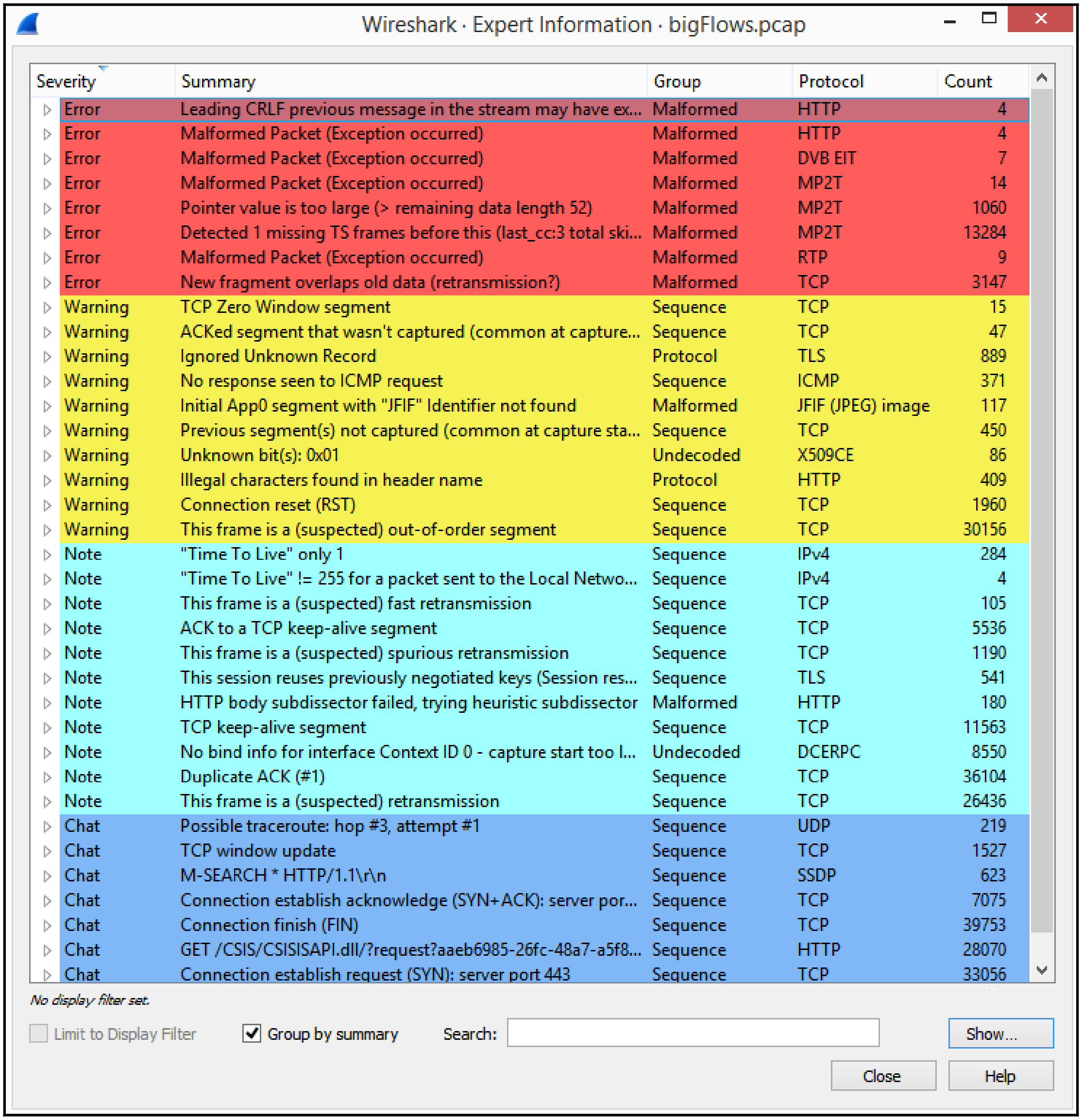
- **Retransmissions or fast retransmissions**: In a TCP connection, each side of a conversation actively monitors the data transaction. When congestion is evident and the data is not getting through, recovery efforts are triggered when certain conditions are met. Depending on the algorithm, you will see retransmissions or fast retransmissions that resend the missing data.
- **Spurious retransmissions**: 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 acknowledgment. This is called spurious retransmission. Although the data is not needed, this can still be 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 the expert system, which groups common issues together so you can quickly investigate network delays, which we'll explore next.

Discovering the expert system

While analyzing a packet capture, you may observe a colored circle in the lower left-hand corner of the interface. That is the expert system, 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 the *bigFlows using coloring rules* screenshot (shown in the *Exploring the Intelligent Scrollbar* section), the expert system shows a red circle, which indicates an error; this is the highest expert information level. If you double-click on the circle, it will open a console, as shown in the following screenshot:



Wireshark · Expert Information · bigFlows.pcap

Severity	Summary	Group	Protocol	Count
Error	Leading CRLF previous message in the stream may have ex...	Malformed	HTTP	4
Error	Malformed Packet (Exception occurred)	Malformed	HTTP	4
Error	Malformed Packet (Exception occurred)	Malformed	DVB EIT	7
Error	Malformed Packet (Exception occurred)	Malformed	MP2T	14
Error	Pointer value is too large (> remaining data length 52)	Malformed	MP2T	1060
Error	Detected 1 missing TS frames before this (last_cc:3 total ski...	Malformed	MP2T	13284
Error	Malformed Packet (Exception occurred)	Malformed	RTP	9
Error	New fragment overlaps old data (retransmission?)	Malformed	TCP	3147
Warning	TCP Zero Window segment	Sequence	TCP	15
Warning	ACKed segment that wasn't captured (common at capture...	Sequence	TCP	47
Warning	Ignored Unknown Record	Protocol	TLS	889
Warning	No response seen to ICMP request	Sequence	ICMP	371
Warning	Initial App0 segment with "JFIF" Identifier not found	Malformed	JFIF (JPEG) image	117
Warning	Previous segment(s) not captured (common at capture sta...	Sequence	TCP	450
Warning	Unknown bit(s): 0x01	Undecoded	X509CE	86
Warning	Illegal characters found in header name	Protocol	HTTP	409
Warning	Connection reset (RST)	Sequence	TCP	1960
Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	30156
Note	"Time To Live" only 1	Sequence	IPv4	284
Note	"Time To Live" != 255 for a packet sent to the Local Netwo...	Sequence	IPv4	4
Note	This frame is a (suspected) fast retransmission	Sequence	TCP	105
Note	ACK to a TCP keep-alive segment	Sequence	TCP	5536
Note	This frame is a (suspected) spurious retransmission	Sequence	TCP	1190
Note	This session reuses previously negotiated keys (Session res...	Sequence	TLS	541
Note	HTTP body subdissector failed, trying heuristic subdissector	Malformed	HTTP	180
Note	TCP keep-alive segment	Sequence	TCP	11563
Note	No bind info for interface Context ID 0 - capture start too l...	Undecoded	DCERPC	8550
Note	Duplicate ACK (#1)	Sequence	TCP	36104
Note	This frame is a (suspected) retransmission	Sequence	TCP	26436
Chat	Possible traceroute: hop #3, attempt #1	Sequence	UDP	219
Chat	TCP window update	Sequence	TCP	1527
Chat	M-SEARCH * HTTP/1.1\r\n	Sequence	SSDP	623
Chat	Connection establish acknowledge (SYN+ACK): server por...	Sequence	TCP	7075
Chat	Connection finish (FIN)	Sequence	TCP	39753
Chat	GET /CSIS/CSISISAPI.dll/?request?aaeb6985-26fc-48a7-a5f8...	Sequence	HTTP	28070
Chat	Connection establish request (SYN): server port 443	Sequence	TCP	33056

No display filter set.

☐ Limit to Display Filter ☑ Group by summary Search:

Expert information grouped by severity

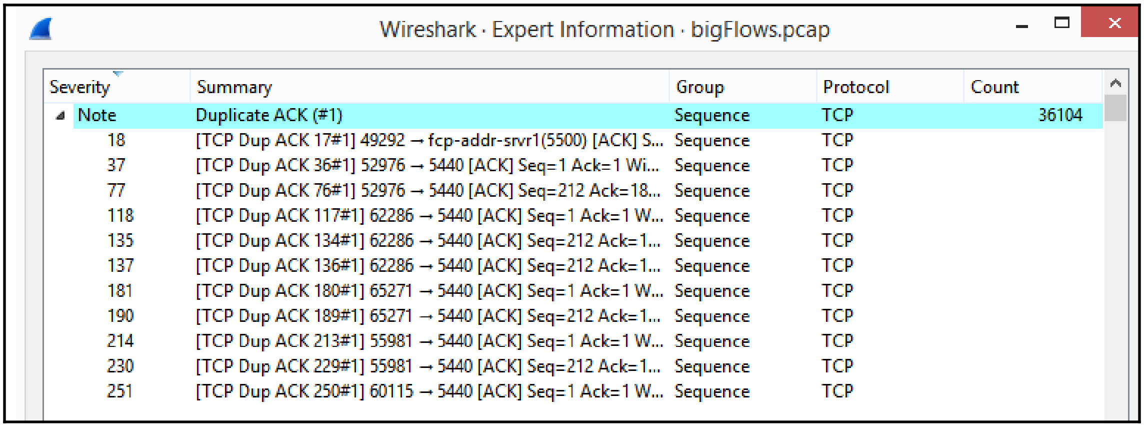
This may take a few minutes to load, depending on the size of the capture. In addition, there may be a lot of information.

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

Across the top of the **Expert Information** interface, you will see the following column headers:



Wireshark · Expert Information · bigFlows.pcap

Severity	Summary	Group	Protocol	Count
Note	Duplicate ACK (#1)	Sequence	TCP	36104
18	[TCP Dup ACK 17#1] 49292 → fcp-addr-srvr1(5500) [ACK] S...	Sequence	TCP	
37	[TCP Dup ACK 36#1] 52976 → 5440 [ACK] Seq=1 Ack=1 Wi...	Sequence	TCP	
77	[TCP Dup ACK 76#1] 52976 → 5440 [ACK] Seq=212 Ack=18...	Sequence	TCP	
118	[TCP Dup ACK 117#1] 62286 → 5440 [ACK] Seq=1 Ack=1 W...	Sequence	TCP	
135	[TCP Dup ACK 134#1] 62286 → 5440 [ACK] Seq=212 Ack=1...	Sequence	TCP	
137	[TCP Dup ACK 136#1] 62286 → 5440 [ACK] Seq=212 Ack=1...	Sequence	TCP	
181	[TCP Dup ACK 180#1] 65271 → 5440 [ACK] Seq=1 Ack=1 W...	Sequence	TCP	
190	[TCP Dup ACK 189#1] 65271 → 5440 [ACK] Seq=212 Ack=1...	Sequence	TCP	
214	[TCP Dup ACK 213#1] 55981 → 5440 [ACK] Seq=1 Ack=1 W...	Sequence	TCP	
230	[TCP Dup ACK 229#1] 55981 → 5440 [ACK] Seq=212 Ack=1...	Sequence	TCP	
251	[TCP Dup ACK 250#1] 60115 → 5440 [ACK] Seq=1 Ack=1 W...	Sequence	TCP	

Note: Duplicate acknowledgment

The following bullets outline what each column 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, including these:
    - **Checksum**: 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 the *Expert information grouped by severity* screenshot, we see there is a count of **36104 Duplicate ACK**.

As shown, the column headers highlights detail of what the packet contains. Within the expert system, Wireshark outlines the level of severity by using color, as we'll see in the following section.

Assessing the severity

When looking at the **Expert Information** console, there are five possible categories that indicate the severity of the issue, as shown here:

Category	**Color**	**Meaning**
Error	Red	Possible serious issue—the highest warning, such as a malformed packet, or 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, that is, 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 **Transport Layer Security** (**TLS**) conversation.
Chat	Gray	Specifies typical workflow and state change such as a connection finish or a Windows update.
Comment	Green	Indicates that there is a comment found in at least one of the packets.

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 **Expert Information**, you'll need to make sense of the data. The expert system 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

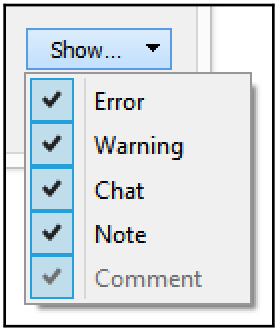
After you launch the **Expert Information** console, all the information may not be sorted. As you'll find, you can easily sort any of the column headers. I typically sort the results in order of severity.

To view all the packets for a specific summary, select the caret on the left-hand side of the summary, as shown in the *Note: Duplicate acknowledgment* screenshot.

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.

You can see an example of this by going to the *Expert information search results* screenshot, as shown in the *Searching for values* section. In the lower left-hand corner, directly above **Limit to Display Filter**, you'll see **Display filter: "http"**.

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 and selecting or deselecting what you would like to display, as shown here:



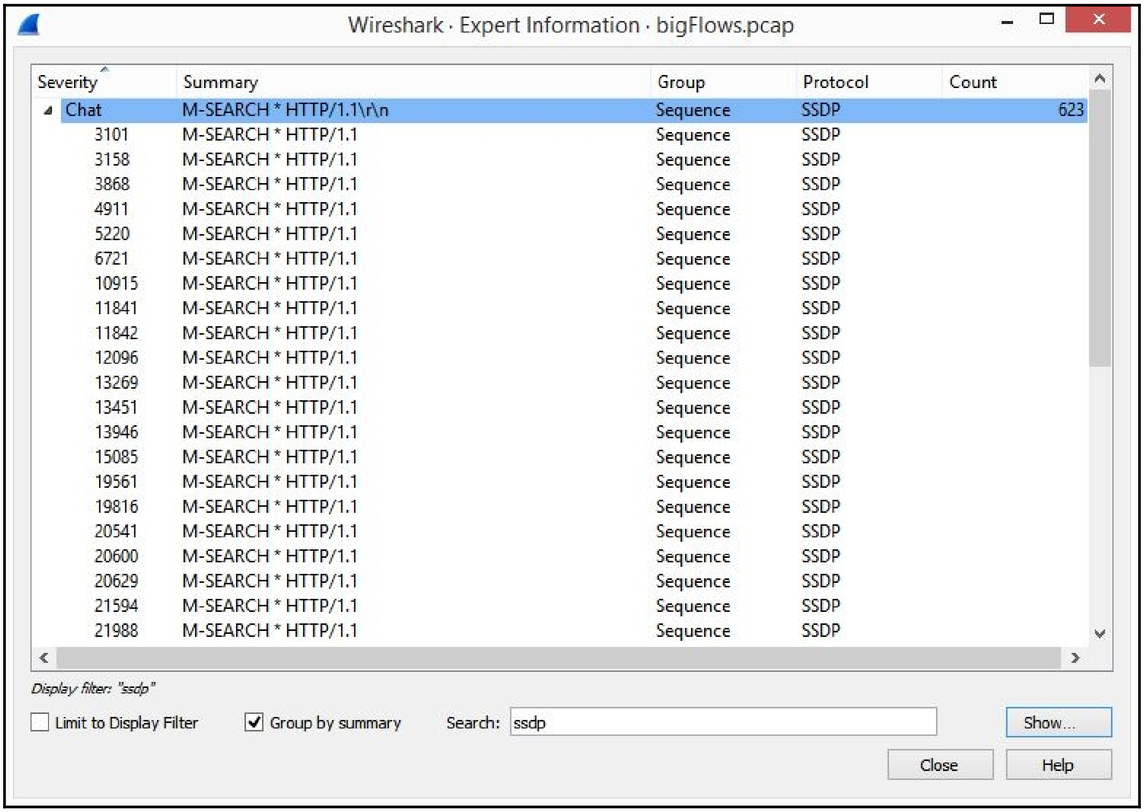
Expert information show categories

In addition, if there are any comments, you can display them as well.

As you can see, you can easily sort data within the expert system. 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 system, enter the value in the search box and press *Enter*. The following screenshot shows the results for the `ssdp` search:



Wireshark · Expert Information · bigFlows.pcap

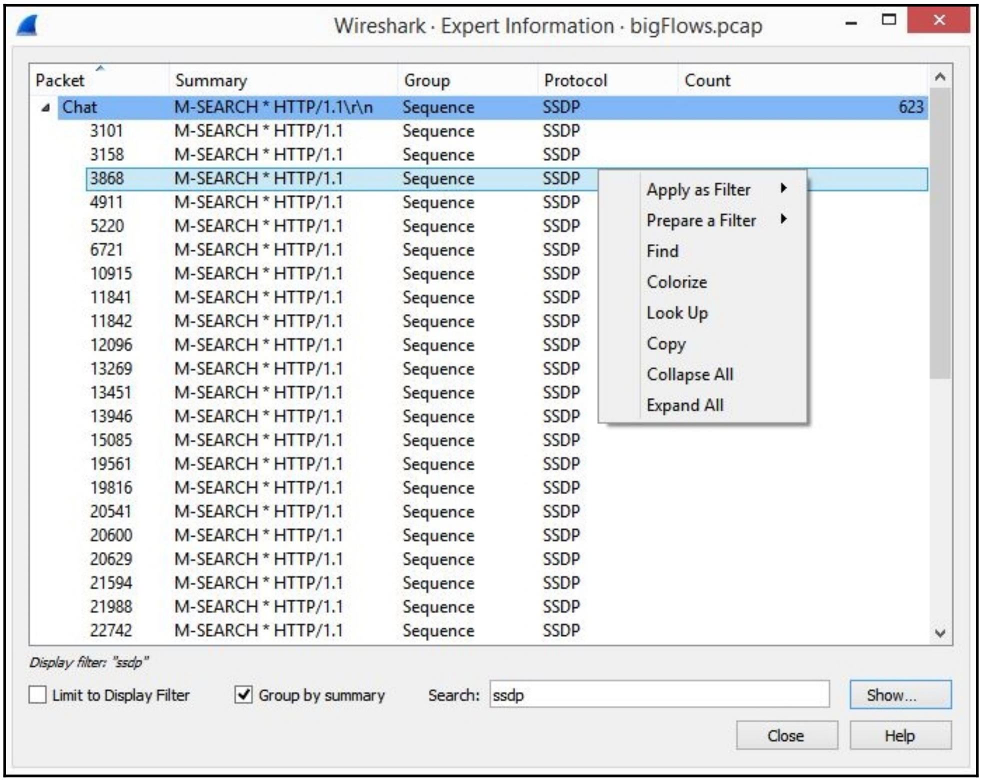
Severity	Summary	Group	Protocol	Count
Chat	M-SEARCH * HTTP/1.1\r\n	Sequence	SSDP	623
3101	M-SEARCH * HTTP/1.1	Sequence	SSDP	
3158	M-SEARCH * HTTP/1.1	Sequence	SSDP	
3868	M-SEARCH * HTTP/1.1	Sequence	SSDP	
4911	M-SEARCH * HTTP/1.1	Sequence	SSDP	
5220	M-SEARCH * HTTP/1.1	Sequence	SSDP	
6721	M-SEARCH * HTTP/1.1	Sequence	SSDP	
10915	M-SEARCH * HTTP/1.1	Sequence	SSDP	
11841	M-SEARCH * HTTP/1.1	Sequence	SSDP	
11842	M-SEARCH * HTTP/1.1	Sequence	SSDP	
12096	M-SEARCH * HTTP/1.1	Sequence	SSDP	
13269	M-SEARCH * HTTP/1.1	Sequence	SSDP	
13451	M-SEARCH * HTTP/1.1	Sequence	SSDP	
13946	M-SEARCH * HTTP/1.1	Sequence	SSDP	
15085	M-SEARCH * HTTP/1.1	Sequence	SSDP	
19561	M-SEARCH * HTTP/1.1	Sequence	SSDP	
19816	M-SEARCH * HTTP/1.1	Sequence	SSDP	
20541	M-SEARCH * HTTP/1.1	Sequence	SSDP	
20600	M-SEARCH * HTTP/1.1	Sequence	SSDP	
20629	M-SEARCH * HTTP/1.1	Sequence	SSDP	
21594	M-SEARCH * HTTP/1.1	Sequence	SSDP	
21988	M-SEARCH * HTTP/1.1	Sequence	SSDP	

Display filter: "ssdp"

□ Limit to Display Filter ☑ Group by summary Search: ssdp

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:



Packet	Summary	Group	Protocol	Count
Chat	M-SEARCH * HTTP/1.1\r\n	Sequence	SSDP	623
3101	M-SEARCH * HTTP/1.1	Sequence	SSDP	
3158	M-SEARCH * HTTP/1.1	Sequence	SSDP	
3868	M-SEARCH * HTTP/1.1	Sequence	SSDP	
4911	M-SEARCH * HTTP/1.1	Sequence	SSDP	
5220	M-SEARCH * HTTP/1.1	Sequence	SSDP	
6721	M-SEARCH * HTTP/1.1	Sequence	SSDP	
10915	M-SEARCH * HTTP/1.1	Sequence	SSDP	
11841	M-SEARCH * HTTP/1.1	Sequence	SSDP	
11842	M-SEARCH * HTTP/1.1	Sequence	SSDP	
12096	M-SEARCH * HTTP/1.1	Sequence	SSDP	
13269	M-SEARCH * HTTP/1.1	Sequence	SSDP	
13451	M-SEARCH * HTTP/1.1	Sequence	SSDP	
13946	M-SEARCH * HTTP/1.1	Sequence	SSDP	
15085	M-SEARCH * HTTP/1.1	Sequence	SSDP	
19561	M-SEARCH * HTTP/1.1	Sequence	SSDP	
19816	M-SEARCH * HTTP/1.1	Sequence	SSDP	
20541	M-SEARCH * HTTP/1.1	Sequence	SSDP	
20600	M-SEARCH * HTTP/1.1	Sequence	SSDP	
20629	M-SEARCH * HTTP/1.1	Sequence	SSDP	
21594	M-SEARCH * HTTP/1.1	Sequence	SSDP	
21988	M-SEARCH * HTTP/1.1	Sequence	SSDP	
22742	M-SEARCH * HTTP/1.1	Sequence	SSDP	

Display filter: "ssdp"

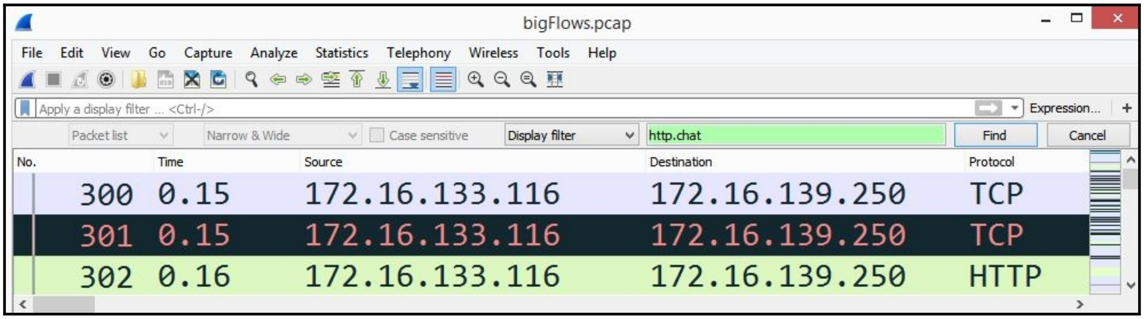
☐ Limit to Display Filter ☑ Group by summary Search: ssdp

Expert information menu choices

Similar to the menu choices offered when you right-click on 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 search toolbar in the main interface, as shown in the screenshot:


bigFlows.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
Find
Cancel

Apply a display filter ... <Ctrl-/>

Packet list | Narrow & Wide | □ Case sensitive | Display filter | http.chat

No.	Time	Source	Destination	Protocol
300	0.15	172.16.133.116	172.16.139.250	TCP
301	0.15	172.16.133.116	172.16.139.250	TCP
302	0.16	172.16.133.116	172.16.139.250	HTTP

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 **Expert Information** 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 Wireshark provides us with in order to identify trouble on the network.

We learned how coloring rules can highlight specific types of traffic. In addition, we discovered how any of the rules can be edited, deleted, or moved up or down in priority. We also looked at the Intelligent Scrollbar, which provides a visual so that we can easily spot and further investigate trouble in the capture.

We learned about the importance of time values and how they factor in latency issues. The expert system helps to alert the network administrator on possible issues once a capture has been made. The **Expert Information** console is an easy-to-use GUI that can be used to drill down on specific issues as it can subset the errors, warnings, notes, and chats.

In the next chapter, we will cover ways to work with large packet captures and break them 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*:

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.
    1. Latency
    2. Packet loss
    3. Goodput
    4. Throughput

2. ____ is the amount of data that is sent and received (typically in bits per second) at any given time.
    1. Latency
    2. Packet loss
    3. Goodput
    4. Throughput

3. 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.
    1. Group metrics
    2. Time values
    3. Intelligent Scrollbar
    4. 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.
    1. Duplicate acknowledgment
    2. Keep-alive
    3. Retransmission
    4. Fast retransmission

5. In the expert system, a Cyan circle indicates _____, which is general information, unusual errors, or a nonstandard use of a protocol.
    1. An error
    2. A warning
    3. A note
    4. A chat

15
Subsetting, Saving, and Exporting Captures

Not every packet capture is the perfect size or representative of the data you need to analyze. Whether you captured the traffic yourself or had someone send a capture to you for analysis, the reality is, some files have to be subdivided down into smaller files for analysis. In addition, when done, you will most likely need to save the file, or in some cases, export the files 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. We can reduce the capture before we begin our analysis by using a command-line tool such as TShark or Dumpcap, as covered in `Chapter 2`, *Using Wireshark NG*, or by using a capture filter, which is covered in `Chapter 7`, *Using Display and Capture Filters*. 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. It might not be your intention to get a large packet capture. Nevertheless, you may find you have to work with one, for a variety of reasons that include the following:

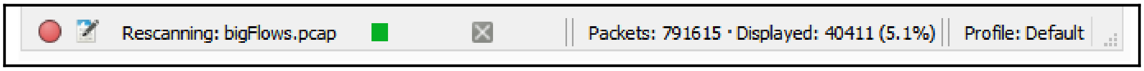
- You may have obtained the capture from a network device with a large amount of traffic. Tapping into the network, even for a short time, can generate a huge number of packets. Even if you used a capture filter while obtaining the file, you may still end up with a large amount of data.
- You may have received a file from someone with good intentions, who felt a large capture would help your analysis. For example, you may 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, Wireshark can load a large file, but it can be very resource-intensive and slow in responding, as Wireshark attempts 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. We can see that there are many ways to break down or subset traffic. Some of the ways include subsetting by IP address, by port number, by protocol, or even by TCP/UDP stream.

Together, we can examine ways of breaking apart a large file. One such capture that works exceptionally well is `bigFlows.cap`. You can download the file, open it in Wireshark, and follow along by going to `http://tcpreplay.appneta.com/wiki/captures.html#bigflows-pcap`.

Once you open `bigFlows.cap`, you can easily see how cumbersome it is to work with a large file. As shown in the following screenshot, this capture has 791,615 packets. Even when entering a simple display filter such as TCP, it will take time for Wireshark to rescan the capture and present the data. As shown in the lower left-hand corner of the following screenshot, Wireshark has a status bar that indicates the process:


Rescanning: bigFlows.pcap Packets: 791615 · Displayed: 40411 (5.1%) Profile: Default

Rescanning the capture

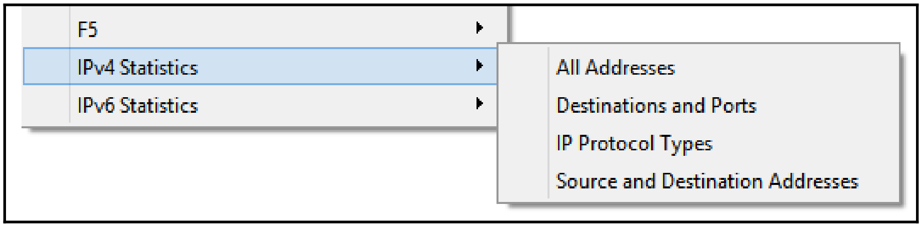
Depending on the system used to analyze the capture, it may run very slowly, freeze up, or even shut down Wireshark.

Once you have opened the file, you'll need to plan what data you want to subset. There are many ways to subset data, and it really depends on what you want to analyze. Let's look at a few ways to break down a large capture. First, we'll examine using an IP address to subset traffic.

Dissecting the capture by IP address

One way to break down a large capture is by filtering a specific IPv4 or IPv6 address, and then use that for your analysis. For example, you know that a specific host is having a problem, and you want to home in on that IP address to troubleshoot the issue.

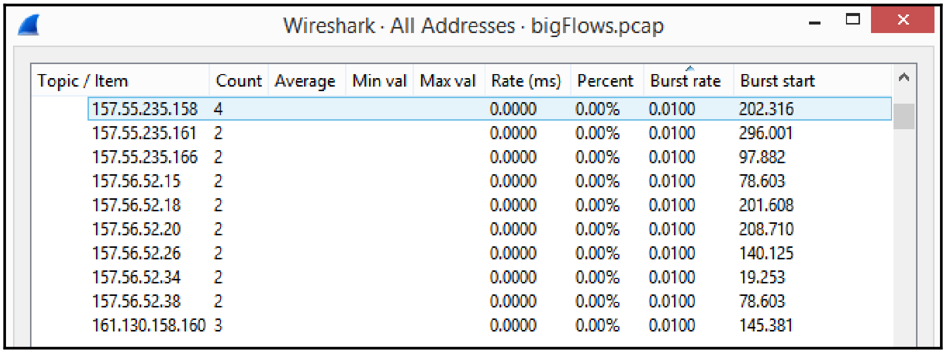
Let's review how to narrow your search. In any large capture, you will most likely have captured many IP addresses. Go to the bottom of the **Statistics** menu, where you will see menu choices for both **IPv4 Statistics** and **IPv6 Statistics**, as shown in the following screenshot:



IPv4 and IPv6 statistics

The IP statistics have four choices for either IPv4 or IPv6, which include **All Addresses**, **Destinations and Ports**, **IP Protocol Types**, and **Source and Destination Addresses**, as follows:

- **All Addresses**: Provides a sortable list of IP addresses with additional information such as **Count** and **Burst rate**:

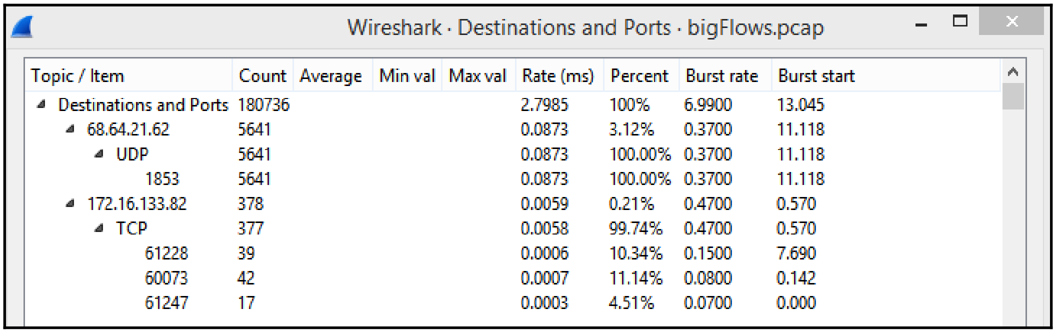


Wireshark · All Addresses · bigFlows.pcap

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
157.55.235.158	4				0.0000	0.00%	0.0100	202.316
157.55.235.161	2				0.0000	0.00%	0.0100	296.001
157.55.235.166	2				0.0000	0.00%	0.0100	97.882
157.56.52.15	2				0.0000	0.00%	0.0100	78.603
157.56.52.18	2				0.0000	0.00%	0.0100	201.608
157.56.52.20	2				0.0000	0.00%	0.0100	208.710
157.56.52.26	2				0.0000	0.00%	0.0100	140.125
157.56.52.34	2				0.0000	0.00%	0.0100	19.253
157.56.52.38	2				0.0000	0.00%	0.0100	78.603
161.130.158.160	3				0.0000	0.00%	0.0100	145.381

Statistics: All Addresses

- **Destination and Ports**: Provides similar information to **All Addresses** such as **Count** and **Burst rate**. However, this report shows a more advanced list that breaks down each IP address with additional statistics on TCP and UDP, as displayed in this screenshot:

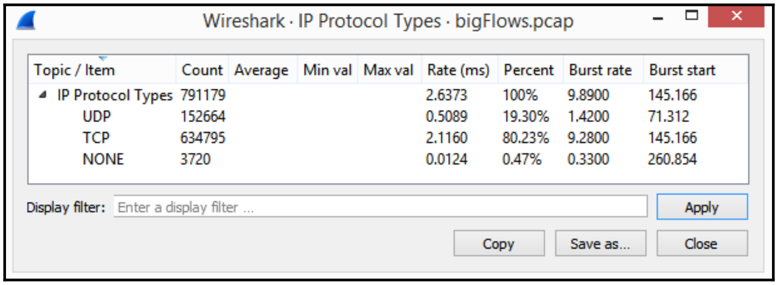


Wireshark · Destinations and Ports · bigFlows.pcap

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Destinations and Ports	180736				2.7985	100%	6.9900	13.045
68.64.21.62	5641				0.0873	3.12%	0.3700	11.118
UDP	5641				0.0873	100.00%	0.3700	11.118
1853	5641				0.0873	100.00%	0.3700	11.118
172.16.133.82	378				0.0059	0.21%	0.4700	0.570
TCP	377				0.0058	99.74%	0.4700	0.570
61228	39				0.0006	10.34%	0.1500	7.690
60073	42				0.0007	11.14%	0.0800	0.142
61247	17				0.0003	4.51%	0.0700	0.000

Statistics: Destinations and Ports

- **IP Protocol Types**: Provides a basic list of transport layer protocols:



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: Enter a display filter ...

Statistics: IP Protocol Types

- **Source and Destination Addresses**: This currently looks like **All Addresses**, as shown in the screenshot captioned *Statistics: All Addresses*, as it may be still in development.

After you run the report, you can search for an IP address with specific characteristics, and then run a filter.

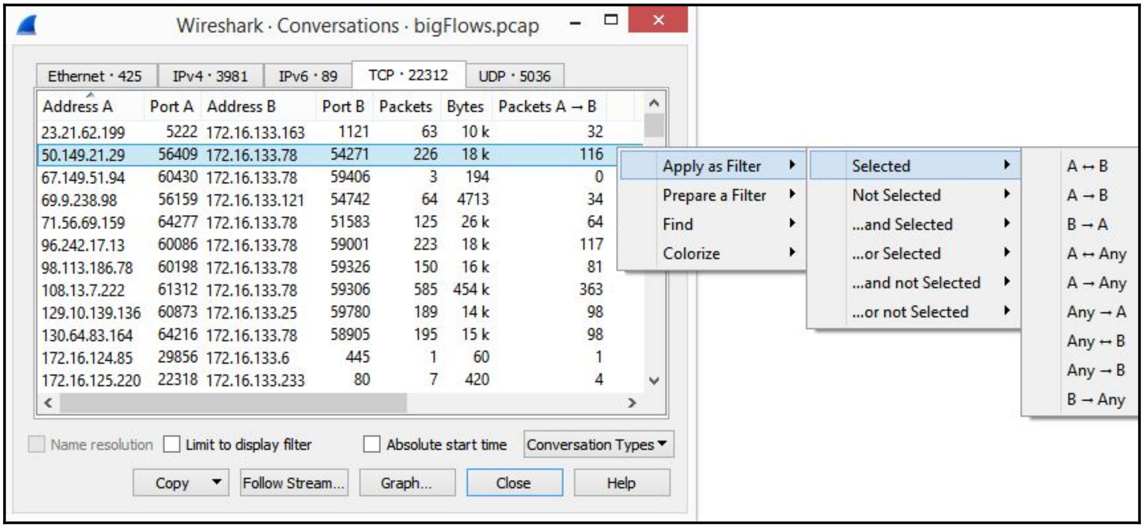
While subsetting traffic by IP addresses may be helpful to home in on troublesome hosts, another way to break down a large capture is by using conversations, which represent two endpoints that are communicating with each other.

Narrowing down by conversations

A conversation is two endpoints communicating with one another. In a large capture, you will most likely have many conversations. We can sort within the conversation dialog box 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 VoIP client and server, and create graphs and flow charts to analyze the data.

Once you have selected the appropriate conversation, you'll need to filter the conversation by going to the **Statistics** menu and selecting **Conversations**.

After the window opens, 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 the type of traffic, such as **IPv4**, and then apply a filter to narrow the results, so you only see the traffic you want to use as your subset, as shown in the following screenshot:



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B
23.21.62.199	5222	172.16.133.163	1121	63	10 k	32
50.149.21.29	56409	172.16.133.78	54271	226	18 k	116
67.149.51.94	60430	172.16.133.78	59406	3	194	0
69.9.238.98	56159	172.16.133.121	54742	64	4713	34
71.56.69.159	64277	172.16.133.78	51583	125	26 k	64
96.242.17.13	60086	172.16.133.78	59001	223	18 k	117
98.113.186.78	60198	172.16.133.78	59326	150	16 k	81
108.13.7.222	61312	172.16.133.78	59306	585	454 k	363
129.10.139.136	60873	172.16.133.25	59780	189	14 k	98
130.64.83.164	64216	172.16.133.78	58905	195	15 k	98
172.16.124.85	29856	172.16.133.6	445	1	60	1
172.16.125.220	22318	172.16.133.233	80	7	420	4

□ Name resolution □ Limit to display filter □ Absolute start time

Conversations: Filter options

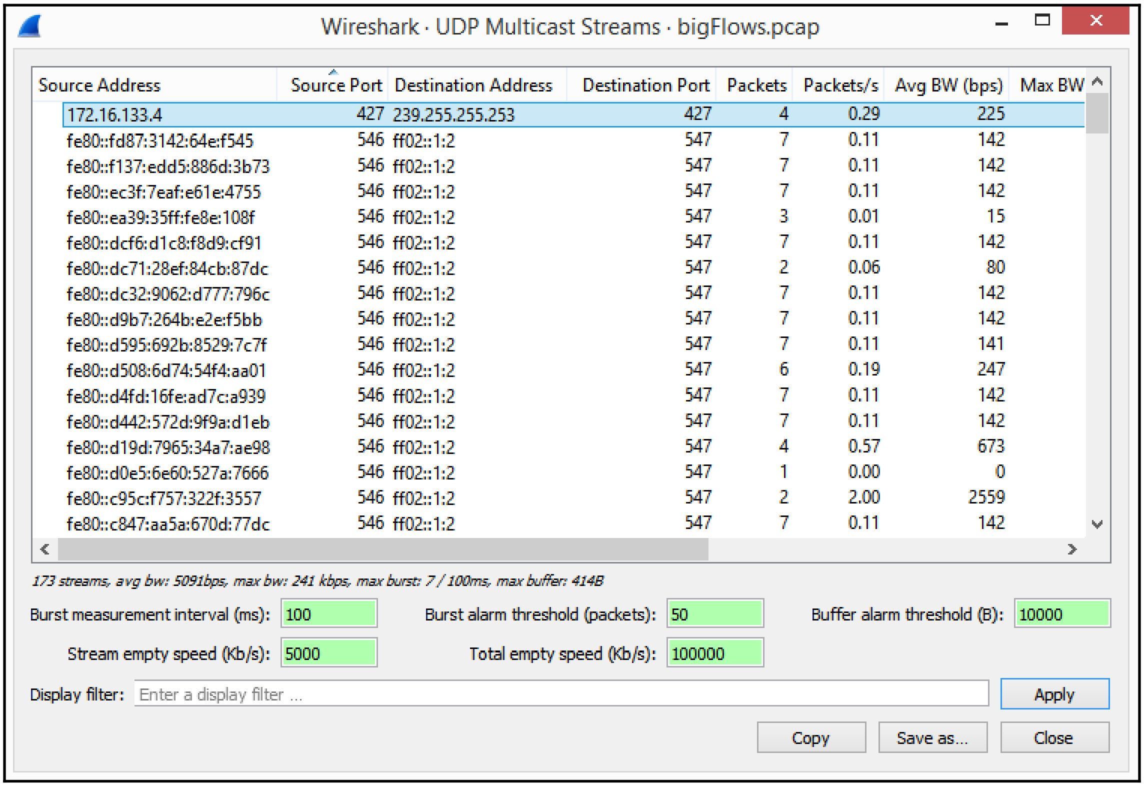
Within one capture, there may be many conversations. Although you may find that subsetting a large capture by conversations is helpful, you'll find that, sometimes, you will want to zero in on a specific port and use that as your subset. The following section illustrates how you can filter by port numbers, so you can work with the resulting smaller file.

Minimizing by port number

While subsetting by IP address or by 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 identifying 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 **Conversations**, **Endpoints**, **IPv4** or **IPv6 Destinations and Ports**, and **UDP Multicast Streams**.

For example, go to **Statistics**, and then **UDP Multicast Streams**, as shown in the following screenshot:



Source Address	Source Port	Destination Address	Destination Port	Packets	Packets/s	Avg BW (bps)	Max BW
172.16.133.4	427	239.255.255.253	427	4	0.29	225	
fe80::fd87:3142:64e:f545	546	ff02::1:2	547	7	0.11	142	
fe80::f137:edd5:886d:3b73	546	ff02::1:2	547	7	0.11	142	
fe80::ec3f:7eaf:e61e:4755	546	ff02::1:2	547	7	0.11	142	
fe80::ea39:35ff:fe8e:108f	546	ff02::1:2	547	3	0.01	15	
fe80::dcf6:d1c8:f8d9:cf91	546	ff02::1:2	547	7	0.11	142	
fe80::dc71:28ef:84cb:87dc	546	ff02::1:2	547	2	0.06	80	
fe80::dc32:9062:d777:796c	546	ff02::1:2	547	7	0.11	142	
fe80::d9b7:264b:e2e:f5bb	546	ff02::1:2	547	7	0.11	142	
fe80::d595:692b:8529:7c7f	546	ff02::1:2	547	7	0.11	141	
fe80::d508:6d74:54f4:aa01	546	ff02::1:2	547	6	0.19	247	
fe80::d4fd:16fe:ad7c:a939	546	ff02::1:2	547	7	0.11	142	
fe80::d442:572d:9f9a:d1eb	546	ff02::1:2	547	7	0.11	142	
fe80::d19d:7965:34a7:ae98	546	ff02::1:2	547	4	0.57	673	
fe80::d0e5:6e60:527a:7666	546	ff02::1:2	547	1	0.00	0	
fe80::c95c:f757:322f:3557	546	ff02::1:2	547	2	2.00	2559	
fe80::c847:aa5a:670d:77dc	546	ff02::1:2	547	7	0.11	142	

173 streams, avg bw: 5091bps, max bw: 241 kbps, max burst: 7 / 100ms, max buffer: 414B

Burst measurement interval (ms): 100 Burst alarm threshold (packets): 50 Buffer alarm threshold (B): 10000

Stream empty speed (Kb/s): 5000 Total empty speed (Kb/s): 100000

Display filter: Enter a display filter ...

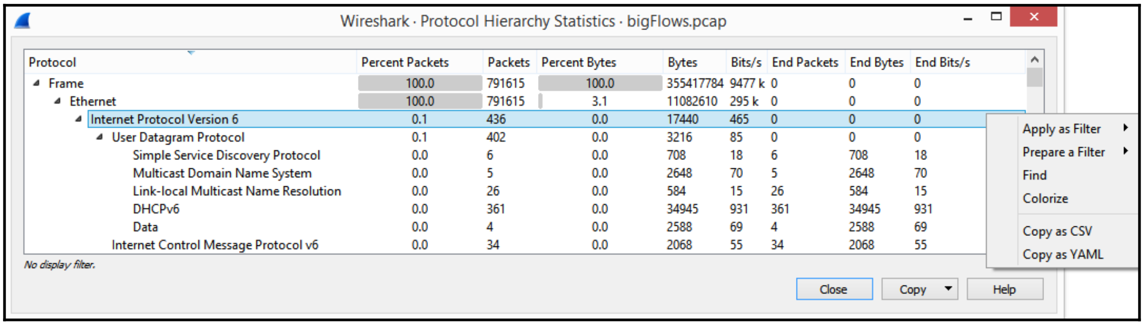
UDP Multicast Streams

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 by a specific protocol. Let's take a look.

Breaking down by protocol

Wireshark is capable of dissecting over 700 protocols. To see a list of protocols in the capture, 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**, you can apply a filter and create your subset, as shown in the following screenshot:



Wireshark · Protocol Hierarchy Statistics · bigFlows.pcap

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	791615	100.0	355417784	9477 k	0	0	0
Ethernet	100.0	791615	3.1	11082610	295 k	0	0	0
Internet Protocol Version 6	0.1	436	0.0	17440	465	0	0	0
User Datagram Protocol	0.1	402	0.0	3216	85	0	0	0
Simple Service Discovery Protocol	0.0	6	0.0	708	18	6	708	18
Multicast Domain Name System	0.0	5	0.0	2648	70	5	2648	70
Link-local Multicast Name Resolution	0.0	26	0.0	584	15	26	584	15
DHCPv6	0.0	361	0.0	34945	931	361	34945	931
Data	0.0	4	0.0	2588	69	4	2588	69
Internet Control Message Protocol v6	0.0	34	0.0	2068	55	34	2068	55

No display filter.

Protocol hierarchy—Apply as Filter

In addition, if you know the protocol you want to review, you can use a display filter and enter a specific protocol and use that as your subset.

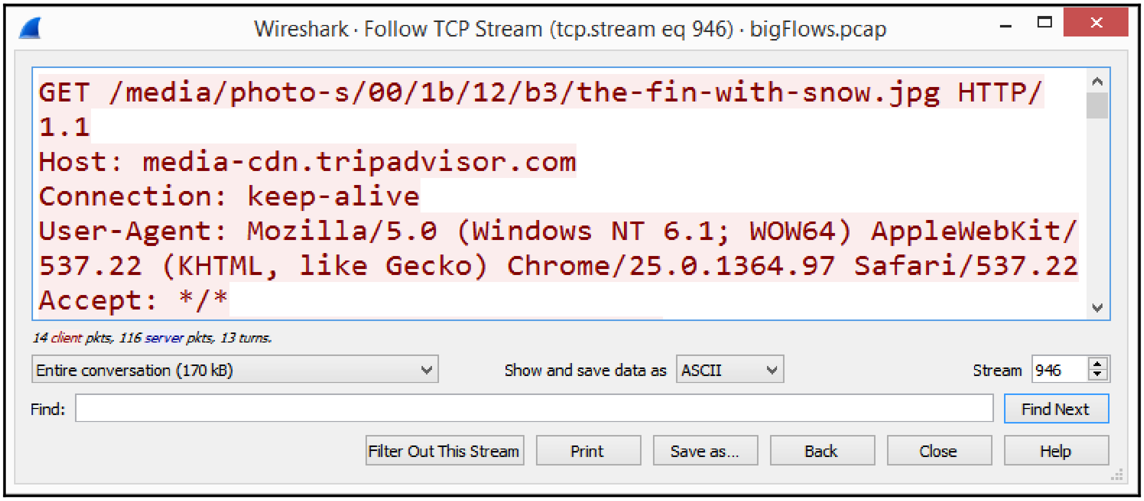
One of the common ways of examining traffic is by examining a particular traffic stream. In the final segment, we will see what elements of a capture we can view by using the follow the stream feature.

Subsetting by stream

There are times 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, either TCP, UDP, TLS, or HTTP.

For our example, 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:



Wireshark · Follow TCP Stream (tcp.stream eq 946) · bigFlows.pcap

```
GET /media/photo-s/00/1b/12/b3/the-fin-with-snow.jpg HTTP/
1.1
Host: media-cdn.tripadvisor.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/
537.22 (KHTML, like Gecko) Chrome/25.0.1364.97 Safari/537.22
Accept: */*
```

14 client pkts, 116 server pkts, 13 turns.

Entire conversation (170 kB) Show and save data as ASCII Stream 946

Find:

Find Next
Filter Out This Stream
Print
Save as...
Back
Close
Help

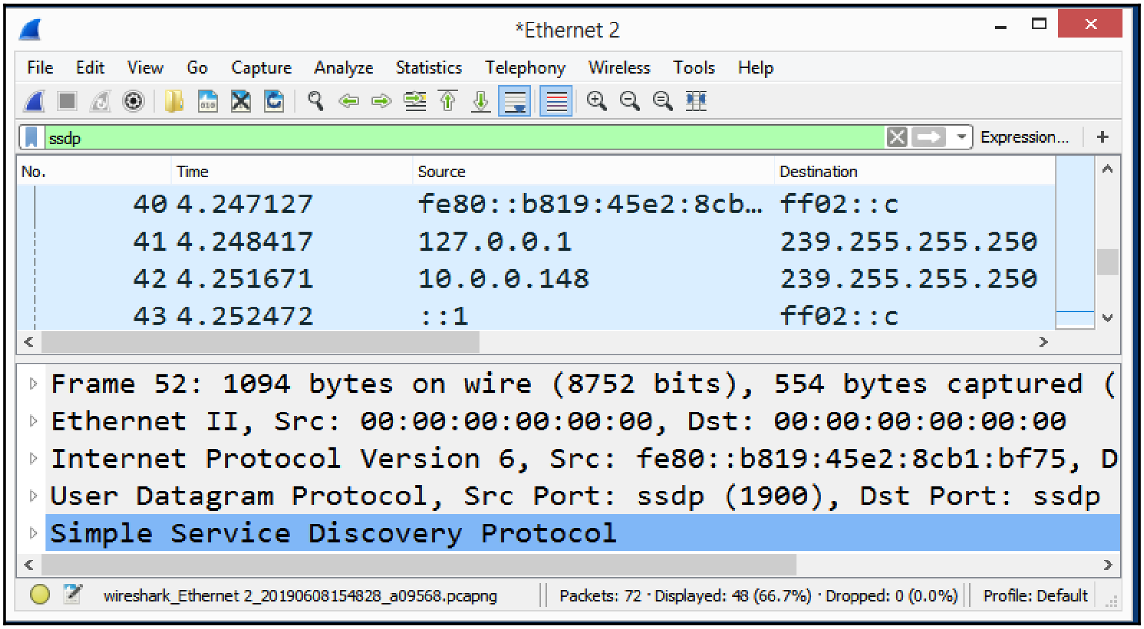
Follow the TCP stream 946

Now that we have reduced the file to a more manageable size by using any of the above methods to subset traffic, the next step is to preserve the file in some way. You can simply save the file in the default `.pcapng` format, or in any of the many other formats that have been added and enhanced over the years.

As you can see, there are many ways to subset a file to a more practical size. After you have created a smaller file, 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 and 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:



ssdp

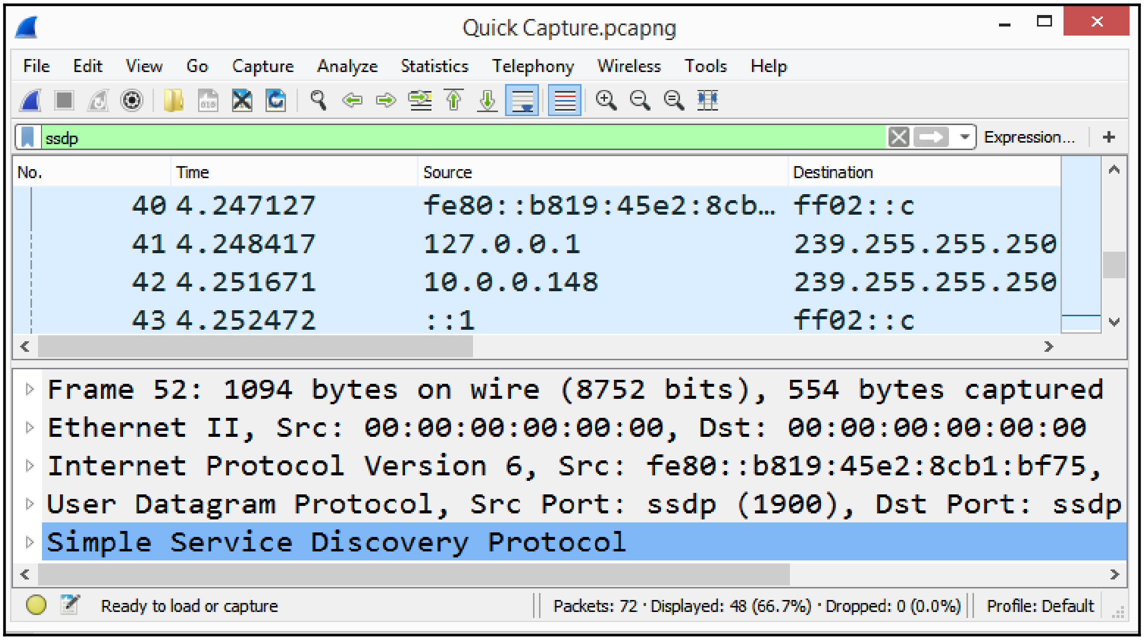
No.	Time	Source	Destination
40	4.247127	fe80::b819:45e2:8cb…	ff02::c
41	4.248417	127.0.0.1	239.255.255.250
42	4.251671	10.0.0.148	239.255.255.250
43	4.252472	::1	ff02::c

```
Frame 52: 1094 bytes on wire (8752 bits), 554 bytes captured (
Ethernet II, Src: 00:00:00:00:00:00, Dst: 00:00:00:00:00:00
Internet Protocol Version 6, Src: fe80::b819:45e2:8cb1:bf75, D
User Datagram Protocol, Src Port: ssdp (1900), Dst Port: ssdp
Simple Service Discovery Protocol
```

wireshark_Ethernet 2_20190608154828_a09568.pcapng | Packets: 72 · Displayed: 48 (66.7%) · Dropped: 0 (0.0%) | Profile: Default

Temporary File in Wireshark

At some point, you will most likely want to save the file in some format. 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:



ssdp

No.	Time	Source	Destination
40	4.247127	fe80::b819:45e2:8cb…	ff02::c
41	4.248417	127.0.0.1	239.255.255.250
42	4.251671	10.0.0.148	239.255.255.250
43	4.252472	::1	ff02::c

```
Frame 52: 1094 bytes on wire (8752 bits), 554 bytes captured
Ethernet II, Src: 00:00:00:00:00:00, Dst: 00:00:00:00:00:00
Internet Protocol Version 6, Src: fe80::b819:45e2:8cb1:bf75,
User Datagram Protocol, Src Port: ssdp (1900), Dst Port: ssdp
Simple Service Discovery Protocol
```

Ready to load or capture | Packets: 72 · Displayed: 48 (66.7%) · Dropped: 0 (0.0%) | Profile: Default

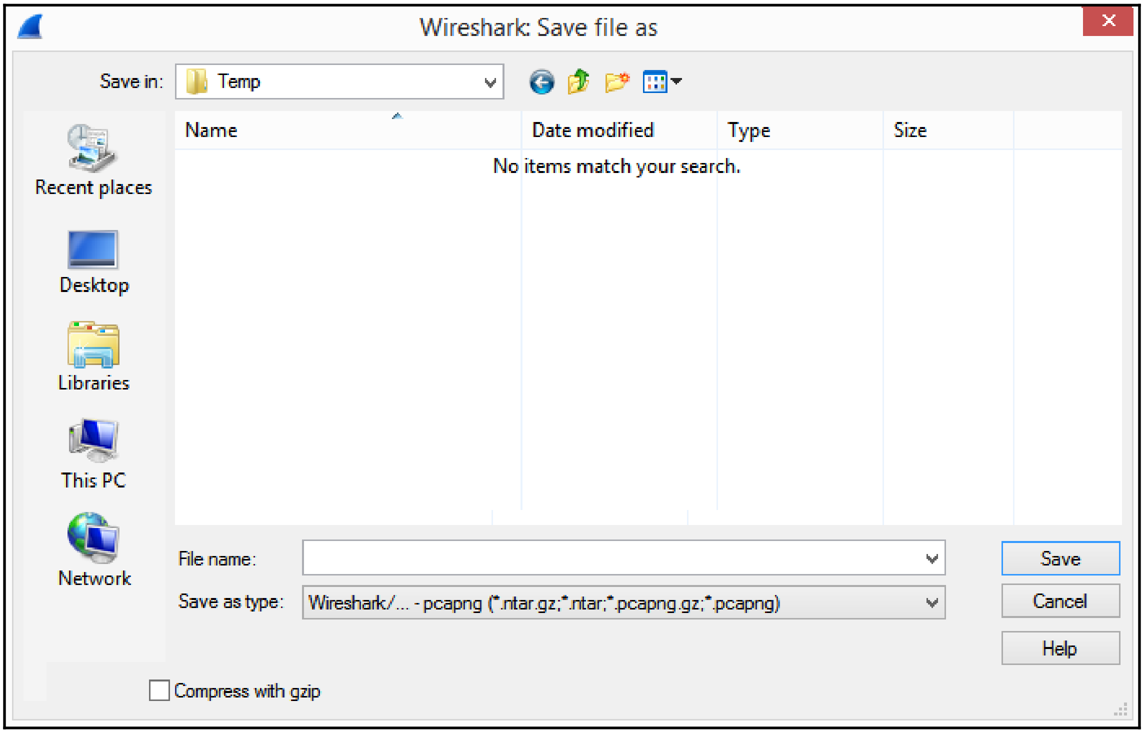
File saved in Wireshark

Once you have gone 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:



Wireshark: Save file as

Save in: Temp

No items match your search.

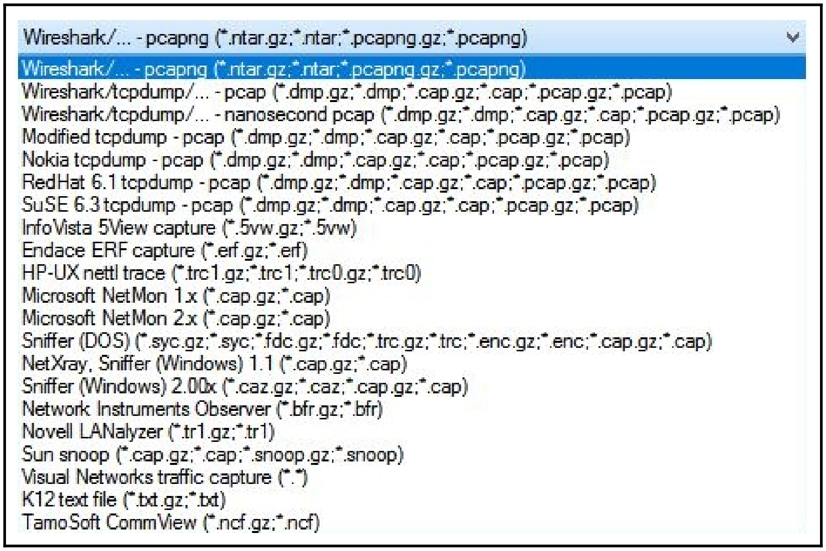
File name:

Save as type: Wireshark/... - pcapng (*.ntar.gz;*.ntar;*.pcapng.gz;*.pcapng)

☐ Compress with gzip

The Save file as dialog box

Over the years, developers have added many different file formats to Wireshark. As a result, when you drop down the **Save as type,** you will see a list of all the supported file formats, as shown here:



Wireshark/... - pcapng (*.ntar.gz;*.ntar;*.pcapng.gz;*.pcapng)

Wireshark/... - pcapng (*.ntar.gz;*.ntar;*.pcapng.gz;*.pcapng)
Wireshark/tcpdump/... - pcap (*.dmp.gz;*.dmp;*.cap.gz;*.cap;*.pcap.gz;*.pcap)
Wireshark/tcpdump/... - nanosecond pcap (*.dmp.gz;*.dmp;*.cap.gz;*.cap;*.pcap.gz;*.pcap)
Modified tcpdump - pcap (*.dmp.gz;*.dmp;*.cap.gz;*.cap;*.pcap.gz;*.pcap)
Nokia tcpdump - pcap (*.dmp.gz;*.dmp;*.cap.gz;*.cap;*.pcap.gz;*.pcap)
RedHat 6.1 tcpdump - pcap (*.dmp.gz;*.dmp;*.cap.gz;*.cap;*.pcap.gz;*.pcap)
SuSE 6.3 tcpdump - pcap (*.dmp.gz;*.dmp;*.cap.gz;*.cap;*.pcap.gz;*.pcap)
InfoVista 5View capture (*.5vw.gz;*.5vw)
Endace ERF capture (*.erf.gz;*.erf)
HP-UX nettl trace (*.trc1.gz;*.trc1;*.trc0.gz;*.trc0)
Microsoft NetMon 1.x (*.cap.gz;*.cap)
Microsoft NetMon 2.x (*.cap.gz;*.cap)
Sniffer (DOS) (*.syc.gz;*.syc;*.fdc.gz;*.fdc;*.trc.gz;*.trc;*.enc.gz;*.enc;*.cap.gz;*.cap)
NetXray, Sniffer (Windows) 1.1 (*.cap.gz;*.cap)
Sniffer (Windows) 2.00x (*.caz.gz;*.caz;*.cap.gz;*.cap)
Network Instruments Observer (*.bfr.gz;*.bfr)
Novell LANalyzer (*.tr1.gz;*.tr1)
Sun snoop (*.cap.gz;*.cap;*.snoop.gz;*.snoop)
Visual Networks traffic capture (*.*)
K12 text file (*.txt.gz;*.txt)
TamoSoft CommView (*.ncf.gz;*.ncf)

Wireshark Save as selections

Here, there are many formats that include Microsoft NetMon (`.cap`), Novell LANalyzer (`.tr1`), Sniffer (Windows: `.caz`), 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 the option of saving the file in other formats.

One common use of the **Save as** menu choice is to open a file in one format, and then save in another format. An example is obtaining a file with a `.pcap` extension. Although you can do many things with the `.pcap` file, the `.pcapng` format is a better option. PCAPNG files have several enhancements and are able to dissect and display payloads better. In addition, 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.

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

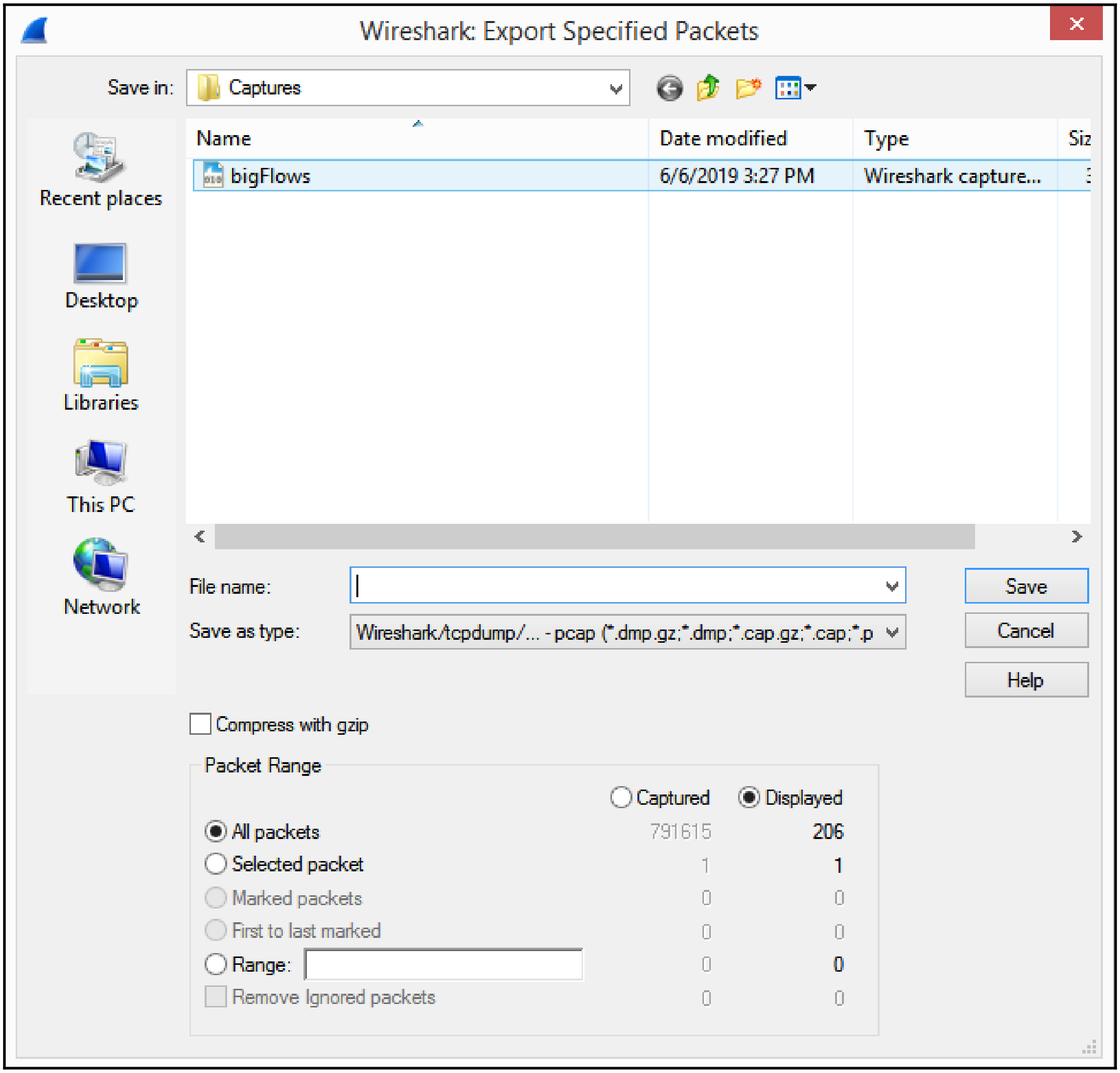
We discussed the many ways you can subset a capture to reduce the file to a more practical size, such as by IP address, by port number, or by a stream. Another option is to export the subset as specified packets, packet dissections, or even export various options that exist in the capture.

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 **Export Specified Packets**. Once open, you will see that you have several ways to export file components, as shown in the following screenshot:



Wireshark: Export Specified Packets

Save in: Captures

Name	Date modified	Type	Siz
bigFlows	6/6/2019 3:27 PM	Wireshark capture...	

File name:

Save as type: Wireshark/tcpdump/... - pcap (*.dmp.gz;*.dmp;*.cap.gz;*.cap;*.p

☐ Compress with gzip

Packet Range

	○ Captured	● Displayed
● All packets	791615	206
○ Selected packet	1	1
○ Marked packets	0	0
○ First to last marked	0	0
○ Range:	0	0
☐ Remove Ignored packets	0	0

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 all the packets, select **Captured**.

Below that, you will see other choices for the packet range you would like:

- **All packets**: This will export all packets. Wireshark will display how many are either **Captured** or **Displayed**.
- **Selected packet**: 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 Wireshark shows one (**1**) packet in the **Selected packet** option.
- **Marked packets**: This allows you to right-click and mark a specified packet or packets of interest, causing the packet(s) to turn black. This option 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, such as 233-799, and 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 **Remove Ignored packets**, 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`. In the case of this export, you will need to force the file format to `.pcapng`, as Wireshark will default to the original file format, which is `.pcap`, for `bigFlows.pcap`. To export 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 the *Export Specified Packets* screenshot.
3. Select a location in which to save the file.
4. In **File name**, enter the `Web Page` filename.
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 has many available options to export, including specific packets or bytes, TLS session keys, and objects, as outlined next.

Exporting various objects

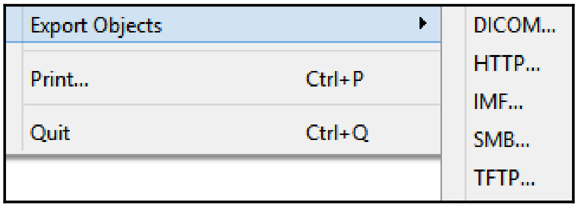
When working with a capture, there may be a variety of objects such as files, images and applications within the file. Wireshark reassembles the objects, which can be collected and analyzed, as long as the object is unencrypted.

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 type 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 what type of traffic is being sent across the network.

Some of the possible objects that can be exported include those from the following protocols:

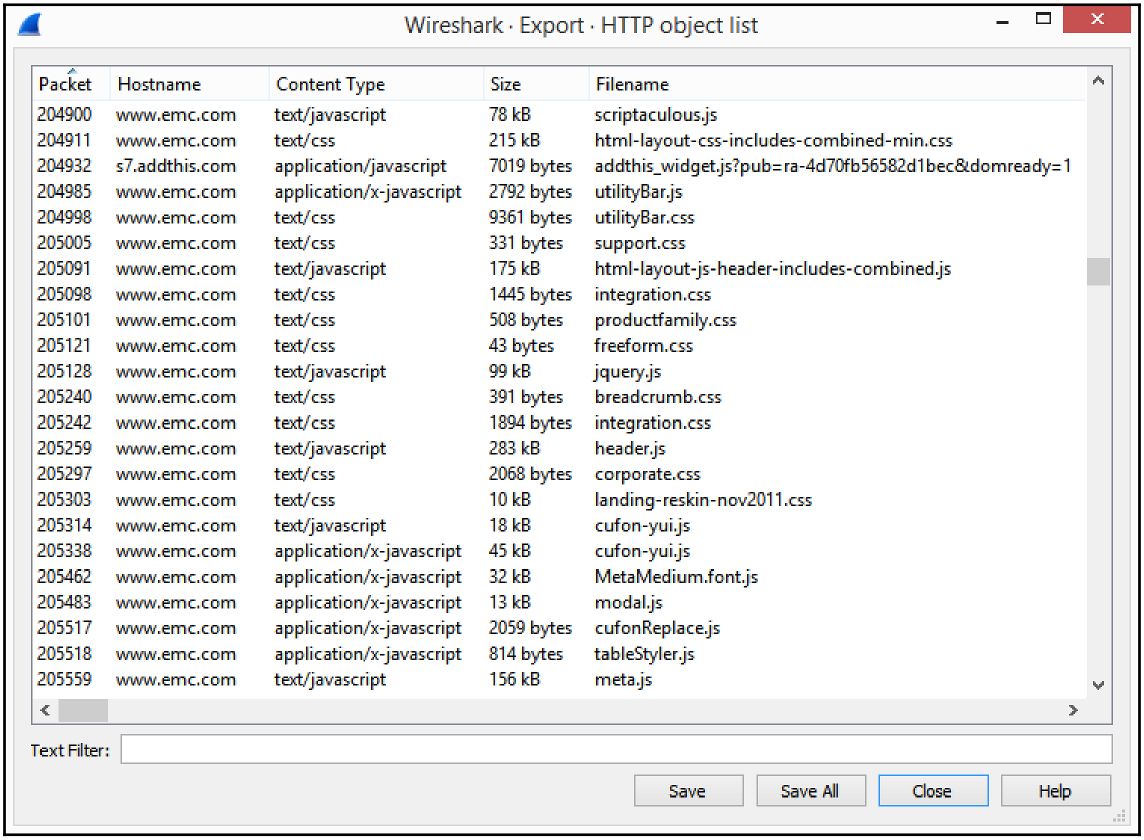
- **Digital Imaging and Communications in Medicine** (**DICOM**)
- **HyperText Transfer Protocol** (**HTTP**)
- **Internet Message Format** (**IMF**)
- **Server Message Block** (**SMB**)
- **Trivial File Transfer Protocol** (**TFTP**)

If you suspect that any of the preceding protocols contains objects, you can export them for examination by going to the **File** menu choice, and then **Export Objects**, as shown in the following screenshot:



Export Objects

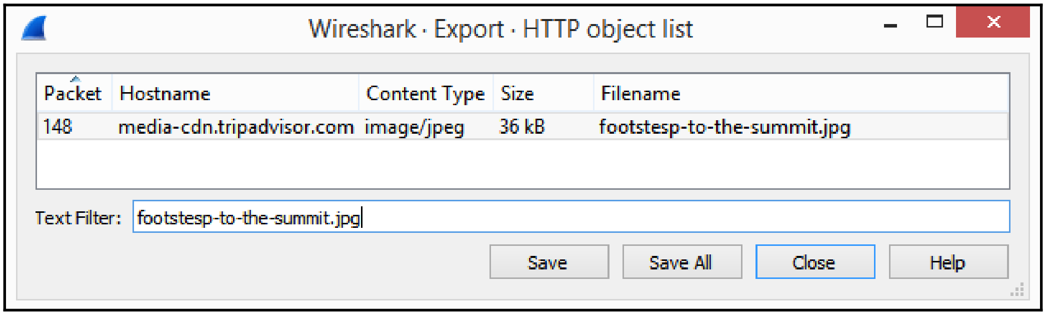
For example, open the `Web Page.pcapng` file. Once open, select **Export Objects**, and then **HTTP...**. Wireshark will locate all objects such as **text/plain**, **applications/javascript** images and **text/html**. This will take a few seconds, depending on the size of the file. Wireshark will then present a list, as shown in the following screenshot:



Packet	Hostname	Content Type	Size	Filename
204900	www.emc.com	text/javascript	78 kB	scriptaculous.js
204911	www.emc.com	text/css	215 kB	html-layout-css-includes-combined-min.css
204932	s7.addthis.com	application/javascript	7019 bytes	addthis_widget.js?pub=ra-4d70fb56582d1bec&domready=1
204985	www.emc.com	application/x-javascript	2792 bytes	utilityBar.js
204998	www.emc.com	text/css	9361 bytes	utilityBar.css
205005	www.emc.com	text/css	331 bytes	support.css
205091	www.emc.com	text/javascript	175 kB	html-layout-js-header-includes-combined.js
205098	www.emc.com	text/css	1445 bytes	integration.css
205101	www.emc.com	text/css	508 bytes	productfamily.css
205121	www.emc.com	text/css	43 bytes	freeform.css
205128	www.emc.com	text/javascript	99 kB	jquery.js
205240	www.emc.com	text/css	391 bytes	breadcrumb.css
205242	www.emc.com	text/css	1894 bytes	integration.css
205259	www.emc.com	text/javascript	283 kB	header.js
205297	www.emc.com	text/css	2068 bytes	corporate.css
205303	www.emc.com	text/css	10 kB	landing-reskin-nov2011.css
205314	www.emc.com	text/javascript	18 kB	cufon-yui.js
205338	www.emc.com	application/x-javascript	45 kB	cufon-yui.js
205462	www.emc.com	application/x-javascript	32 kB	MetaMedium.font.js
205483	www.emc.com	application/x-javascript	13 kB	modal.js
205517	www.emc.com	application/x-javascript	2059 bytes	cufonReplace.js
205518	www.emc.com	application/x-javascript	814 bytes	tableStyler.js
205559	www.emc.com	text/javascript	156 kB	meta.js

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`, as shown in the following screenshot:



Wireshark · Export · HTTP object list

Packet	Hostname	Content Type	Size	Filename
148	media-cdn.tripadvisor.com	image/jpeg	36 kB	footstesp-to-the-summit.jpg

Text Filter: footstesp-to-the-summit.jpg

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 `footsteps-to-the-summit.jpg`. After you save the object, locate, open, and view the image, as shown in the following screenshot:

Exported HTTP Object

If there are other objects, you can save them as well; alternatively, you can select **Save All**, and Wireshark will save all objects found in the file.

As you can see, 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 the significance. In either case, it's best to identify key elements and concerns by adding comments. In order to preserve the reasons why the file was important, Wireshark provides ways to add comments to a single packet or an entire capture, 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 to the entire capture, for future reference.

Wireshark has options when working with comments. You can add file comments to preserve the details of the single capture or even add comments to a single packet. Let's start with how to add comments to the entire file.

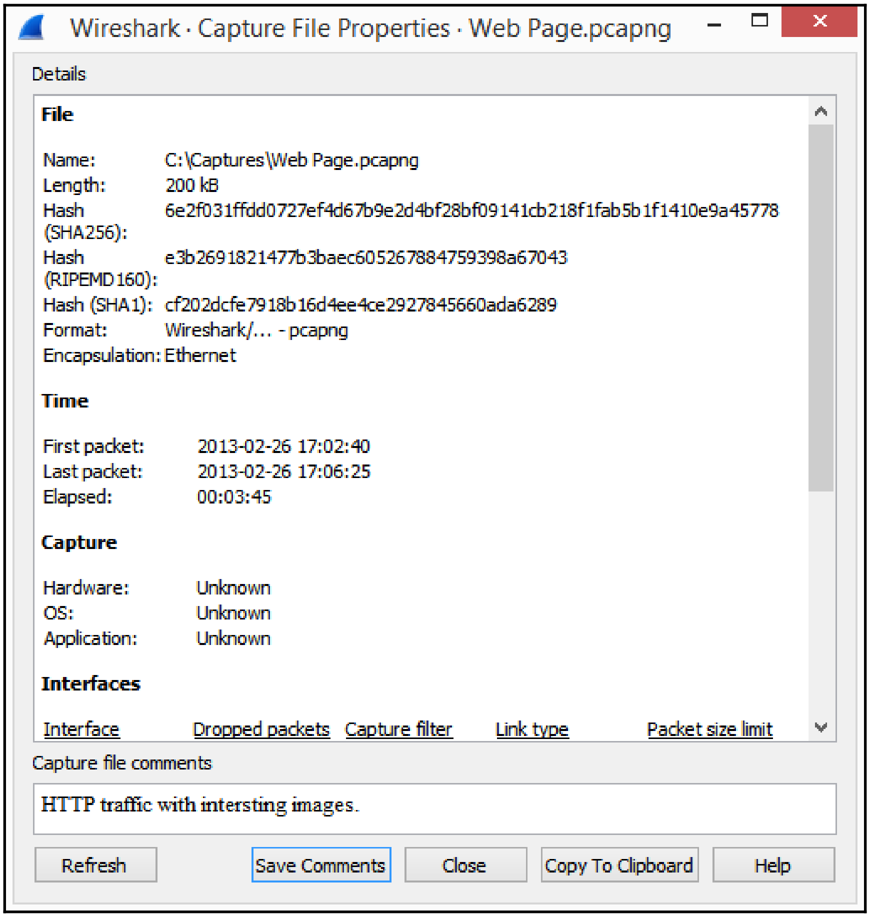
Providing file and packet comments

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 the **Capture file comments**.

In my `Web Page.pcapng` file, I entered the comment, `HTTP traffic with interesting images`, and then clicked **Save Comments**, as shown in the following screenshot:



Wireshark · Capture File Properties · Web Page.pcapng

Details

File

Name: C:\Captures\Web Page.pcapng
Length: 200 kB
Hash (SHA256): 6e2f031ffdd0727ef4d67b9e2d4bf28bf09141cb218f1fab5b1f1410e9a45778
Hash (RIPEMD160): e3b2691821477b3baec605267884759398a67043
Hash (SHA1): cf202dcfe7918b16d4ee4ce2927845660ada6289
Format: Wireshark/... - pcapng
Encapsulation: Ethernet

Time

First packet: 2013-02-26 17:02:40
Last packet: 2013-02-26 17:06:25
Elapsed: 00:03:45

Capture

Hardware: Unknown
OS: Unknown
Application: Unknown

Interfaces

Interface Dropped packets Capture filter Link type Packet size limit

Capture file comments

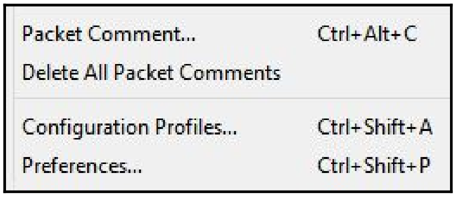
HTTP traffic with intersting images.

Capture File Properties-Web Page.pcapng

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 do a spellcheck.

While adding a comment to an entire file is handy, sometimes, you may want to preserve the details of one, or possibly a few, packets that you want to identify 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, in this case, while in a single packet, go to the **Edit** menu choice and select **Packet Comment**…, as shown in the following screenshot:


Packet Comment... Ctrl+Alt+C
Delete All Packet Comments
Configuration Profiles... Ctrl+Shift+A
Preferences... Ctrl+Shift+P

Packet Comment...

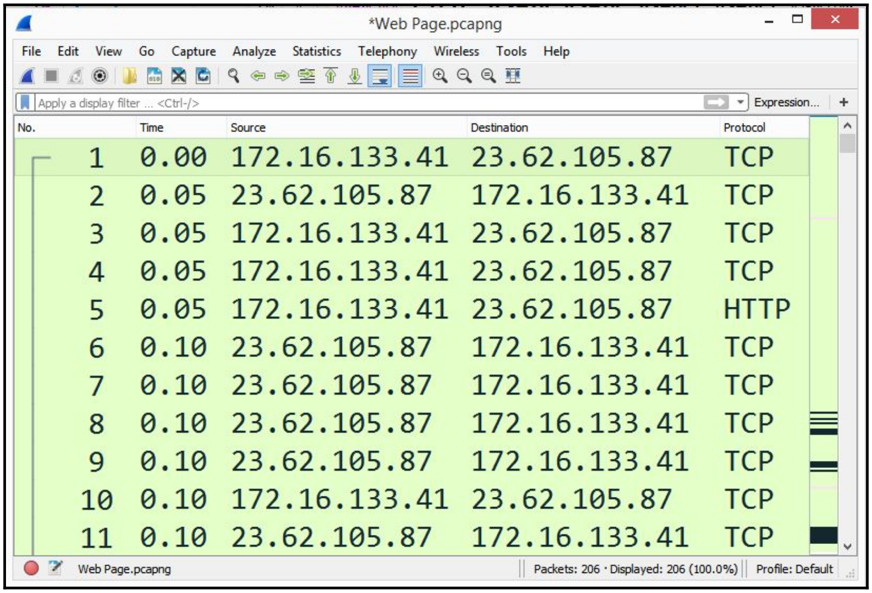
Wireshark will open a form where you can add your comment. If you would like to add more comments later, simply select the same packet and repeat the steps you took to add the original comment.

In addition, you can delete all packet comments by going to the **Edit** menu choice and selecting **Delete all Packet Comments**, as shown in the *Packet Comment...* screenshot.

Once you are done with the 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 a little asterisk in front of the name, as shown here:



*Web Page.pcapng

No.	Time	Source	Destination	Protocol
1	0.00	172.16.133.41	23.62.105.87	TCP
2	0.05	23.62.105.87	172.16.133.41	TCP
3	0.05	172.16.133.41	23.62.105.87	TCP
4	0.05	172.16.133.41	23.62.105.87	TCP
5	0.05	172.16.133.41	23.62.105.87	HTTP
6	0.10	23.62.105.87	172.16.133.41	TCP
7	0.10	23.62.105.87	172.16.133.41	TCP
8	0.10	23.62.105.87	172.16.133.41	TCP
9	0.10	23.62.105.87	172.16.133.41	TCP
10	0.10	172.16.133.41	23.62.105.87	TCP
11	0.10	23.62.105.87	172.16.133.41	TCP

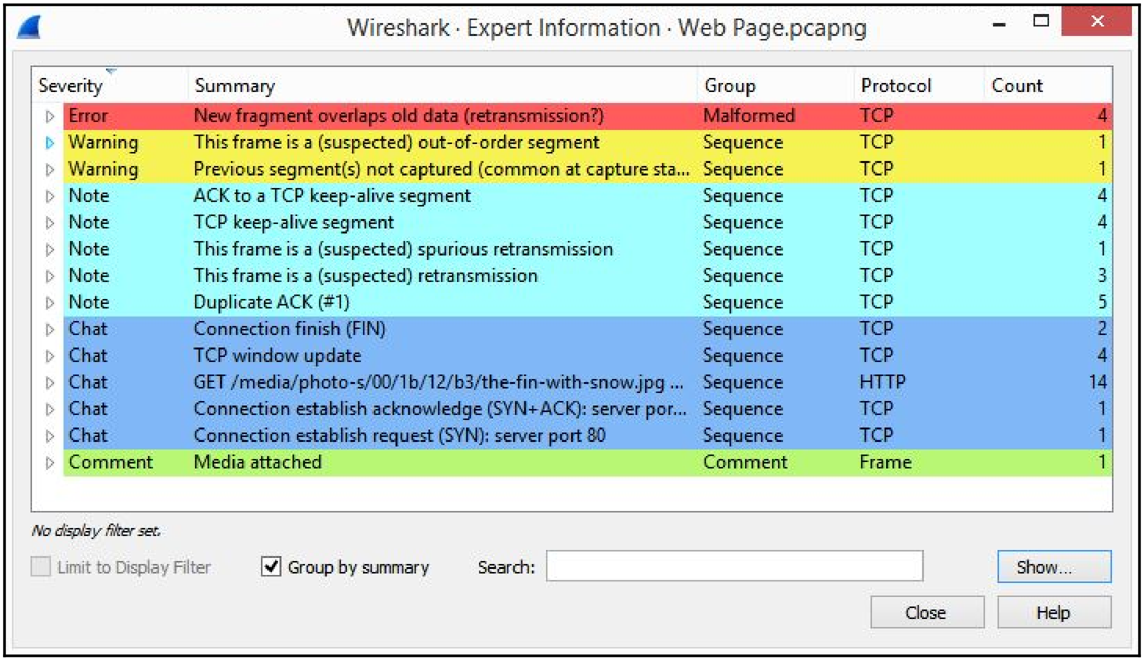
Web Page.pcapng | Packets: 206 · Displayed: 206 (100.0%) | Profile: Default

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 you have preserved the comments, there are several ways to view the comments:

- To see comments on a file, go to **Statistics** | **Capture File Properties**, as shown in the *Capture File Properties-Web Page.pcapng* screenshot.
- To see comments on packets, go to **Expert System** and select **Show Comments**, which is on the lower right-hand side of **Expert Information Console**. You will then see the comments listed, as shown in the following screenshot:



Wireshark · Expert Information · Web Page.pcapng

Severity	Summary	Group	Protocol	Count
Error	New fragment overlaps old data (retransmission?)	Malformed	TCP	4
Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	1
Warning	Previous segment(s) not captured (common at capture sta...	Sequence	TCP	1
Note	ACK to a TCP keep-alive segment	Sequence	TCP	4
Note	TCP keep-alive segment	Sequence	TCP	4
Note	This frame is a (suspected) spurious retransmission	Sequence	TCP	1
Note	This frame is a (suspected) retransmission	Sequence	TCP	3
Note	Duplicate ACK (#1)	Sequence	TCP	5
Chat	Connection finish (FIN)	Sequence	TCP	2
Chat	TCP window update	Sequence	TCP	4
Chat	GET /media/photo-s/00/1b/12/b3/the-fin-with-snow.jpg ...	Sequence	HTTP	14
Chat	Connection establish acknowledge (SYN+ACK): server por...	Sequence	TCP	1
Chat	Connection establish request (SYN): server port 80	Sequence	TCP	1
Comment	Media attached	Comment	Frame	1

No display filter set.

□ Limit to Display Filter ☑ Group by summary Search:

Expert Information—show comments

Now, you can see how easy it is to add comments to an entire capture or a single packet. Once done, it's important to save the capture in `.pcapng` format, so you can view the comments at a later date.

Summary

In this chapter, we discovered that you may want to take a large unmanageable file and turn it into a smaller, more manageable file, so that you can share it with co-workers or preserve the capture for future reference. You learned about the many ways to subset traffic, which includes filtering traffic 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. 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, you will discover CloudShark and learn how you can view captures in your browser from anywhere with internet access. We'll cover the benefits of using CloudShark to share and analyze packet captures with your team. You'll get a good understanding of the filters, graphs, and analysis tools that CloudShark has. In addition, we'll take a look at the many online repositories in order to locate sample captures and enhance our packet analysis skills.

Questions

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

1. A _____ in Wireshark represents two endpoints that are communicating with each other.
    1. Match point
    2. Tuple
    3. Conversation
    4. Filter
2. Wireshark is capable of dissecting over 700 protocols. To see a list of protocols in the capture, go to Statistics, and then _____.
    1. Protocol Hierarchy
    2. Conversations
    3. IPv4 Statistics
    4. Match point

3. Currently, when you save a file, the default file format in Wireshark is _____.
    1. snoop.gz
    2. .pcapng
    3. .pcap
    4. 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.
    1. Ignore
    2. Snoop
    3. Spatter
    4. Mark
5. When you select _____ 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.
    1. DNS
    2. DICOM
    3. HTTP
    4. SMB

16

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**). While CS does not have as many features as Wireshark, you can still execute a number of different packet analysis tasks with it.

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 do with Wireshark, you can create an account and perform more advanced tasks 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 and creating graphs to provide a visual representation of the data. We'll look at various analysis tools, such as VoIP calls, RTP streams, and HTTP analysis. Finally, so that you continue to improve your packet analysis skills, we will take a look at the many online repositories for sample captures.

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

- Diving into an overview of CS
- Sharing captures in CS
- Outlining the various filters and graphs
- Evaluating the different analysis tools
- Discovering where to find sample captures

Diving into an overview of CS

Most of us would agree that Wireshark is a great tool for packet analysis, troubleshooting, and identifying malware and other anomalies on a network. However, Wireshark has some limitations, in that it must be installed on a PC or macOS, and the user may not have the skills to zero in on the actual problems. In addition, Wireshark is not designed to be used concurrently by multiple people, such as in a team.

CS is a browser-based solution that provides a way to upload packet captures and share them with your co-workers. You can also do an analysis on the fly, or simply use it as a browser-based solution to learn about packet analysis.

Now that you know about some of the benefits, let's take a look at how you find CS, as discussed next.

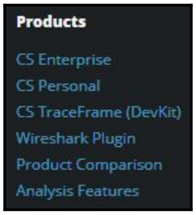
Finding CS

CS is a browser-based solution that you can find at `https://cloudshark.io/`. Once on the website, you will see that there are many industries that use CS.

These industries include the following:

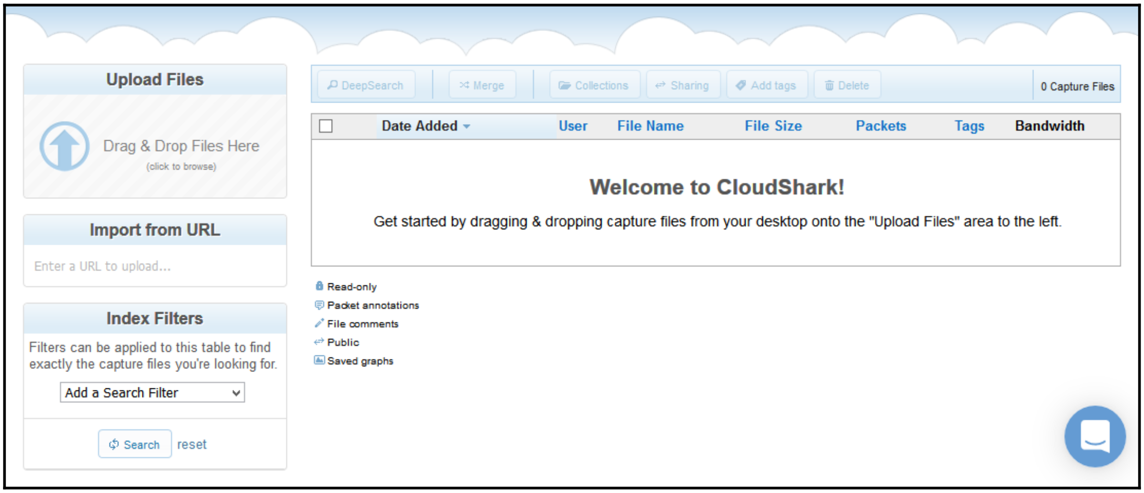
- Government
- VoIP service providers
- Cyber defense
- Security
- Educational institutions

Once on the site, you will see that CS has several products. Depending on what solution you need, you'll find CS Personal and Enterprise, along with TraceFrame (DevKit), and a Wireshark plugin. To see an example of how CS can be a collaboration tool, find the list of products, as shown in the following screenshot, and select **CS Personal**:



CS products

From there, you can set up a free trial. You can create a trial account and then CS will present you with the welcome page, as shown in the following screenshot:



Filters can be applied to this table to find exactly the capture files you're looking for.

□	Date Added	User	File Name	File Size	Packets	Tags	Bandwidth

Welcome to CloudShark!

Get started by dragging & dropping capture files from your desktop onto the "Upload Files" area to the left.

CS welcome page

Once on the welcome page, CS provides a way to upload files from your PC or laptop or import them from a URL, as shown on the left-hand side of the screenshot of the CS welcome page.

After you have uploaded the files, they will be listed on the right-hand side of the screen. If there are several packet captures, you can use a filter search for a specific capture.

Now that you can see how easy it is to find CS, let's take a look at how you can share captures with your team.

Sharing captures in CS

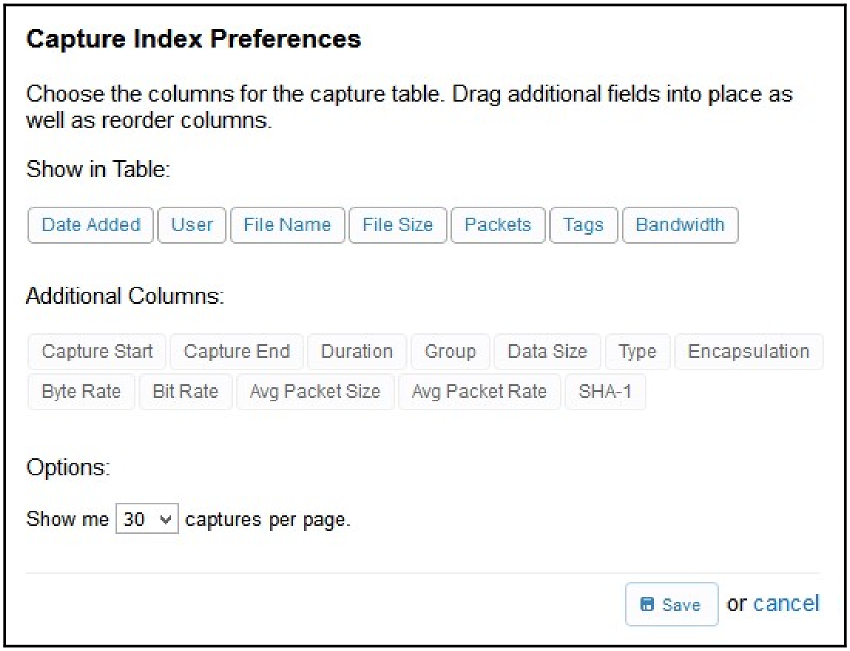
CS provides a way to securely share your captures and allows packet analysis from a wide array of devices. Several analysis tools are available via a web interface that is similar to that of Wireshark. Once you have created an account, CS provides a way to customize the interface for you and your team.

Once you become familiar with the CS interface, you can go in and adjust many aspects of your CS 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 set your preferences, as follows:

- **Account**: Here, you can view subscription information. For example, mine is a Hosted Trial plan. It offers the option to start a subscription when you are ready.
- **Capture Index Preferences**: This is where you can customize your column headers. As shown in the following screenshot, you can use the default layout, remove any of the visible fields as shown, or drag additional columns in 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

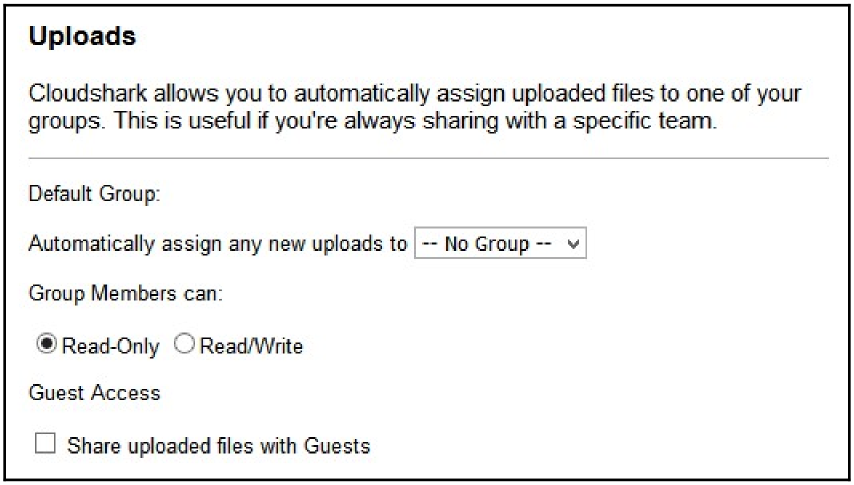
Options:

Show me 30 captures per page.

CS Index Preferences

When 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 loaded in CS. The **Uploads** preferences are where you tell CS to what group you want to assign the files you have uploaded, as shown in the following screenshot:



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 -- No Group --

Group Members can:

● Read-Only ○ Read/Write

Guest Access

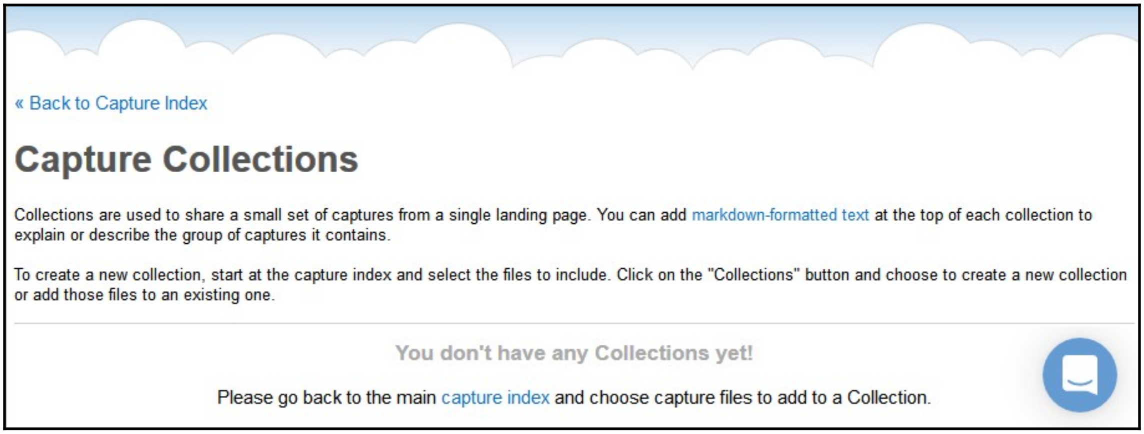
□ Share uploaded files with Guests

CS Uploads preferences

Once they are loaded, you can further restrict what the group members can do with the files, either read-only or read/write. In addition, CS can provide guest access to your uploaded files.

- **Decode Window**: When viewing a capture file, this is where you can enable settings to colorize the packet list or show any annotations.
- **API Tokens**: You may have your own tools that you want to interact with CS. Here is where you can find the **application program 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 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 limits, there is a link to upgrade.

- **Collections**: To organize your captures, you can create collections that can group and share a set of captures that will appear on a separate landing page, as shown in the following screenshot:



« Back to Capture Index

Capture Collections

Collections are used to share a small set of captures from a single landing page. You can add markdown-formatted text at the top of each collection to explain or describe the group of captures it contains.

To create a new collection, start at the capture index and select the files to include. Click on the "Collections" button and choose to create a new collection or add those files to an existing one.

You don't have any Collections yet!

Please go back to the main capture index and choose capture files to add to a Collection.

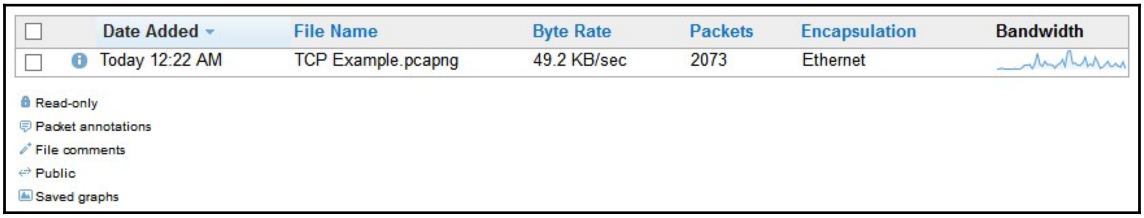
CS Capture Collections

The capture collections are similar to a folder, where you can house similar captures together.

After modifying your preferences, you're ready to upload your captures, to share with your team, or the world, as discussed next.

Uploading captures

When you're ready to upload and share your captures, go to the left-hand side of the CS welcome page, to the upload files area. 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, CloudShark will show a summary of the file, as shown here:

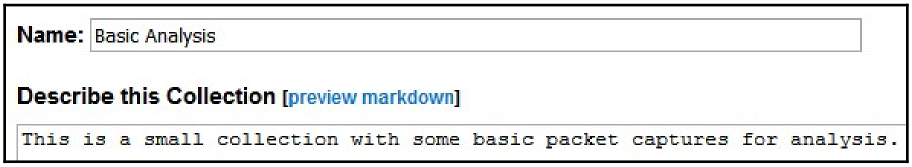


□	Date Added	File Name	Byte Rate	Packets	Encapsulation	Bandwidth
□	Today 12:22 AM	TCP Example.pcapng	49.2 KB/sec	2073	Ethernet	

File uploaded

When you are ready, click **Done** to return to the main menu. Once there, you will see your file along with a menu choice where you can select sharing settings or add to a collection. Select the **Collections** button. If you do not have any collections to add the files, you can create a new collection from the drop-down menu.

When you create a new collection, there is a form where you name your collection. In my example, I used the name `Basic Analysis`. After the name, you then can provide a brief description, as shown in the following screenshot:



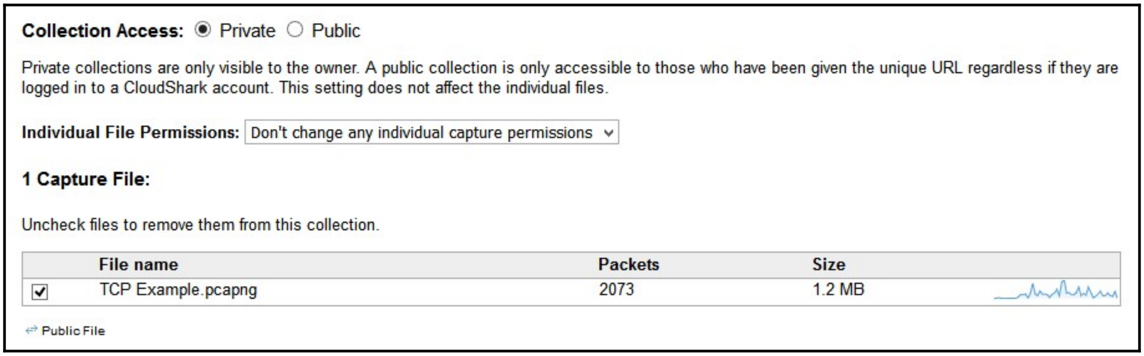
Name: Basic Analysis

Describe this Collection [preview markdown]

This is a small collection with some basic packet captures for analysis.

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: Don't change any individual capture permissions

1 Capture File:

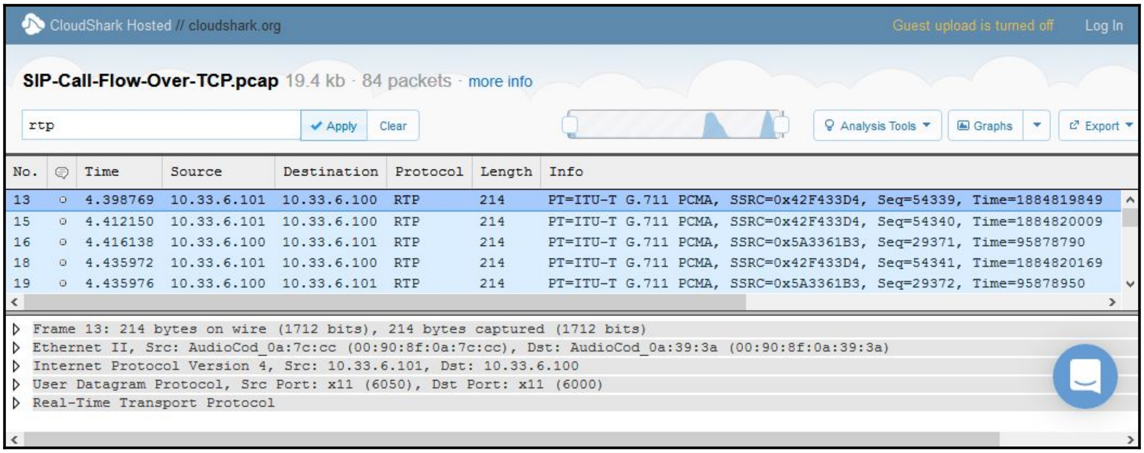
Uncheck files to remove them from this collection.

	File name	Packets	Size	
☑	TCP Example.pcapng	2073	1.2 MB	

Collection Access

When done, select **Save** to return to the main window, where you can select a file and then double-click to open it in the analysis window.

Once the file is open, you are ready to begin your analysis. The interface looks similar to the Wireshark interface. You can make some modifications; for example, to give you more room, you can pull the lower-pane window down so that you can expand the protocol trees, as shown in this screenshot:

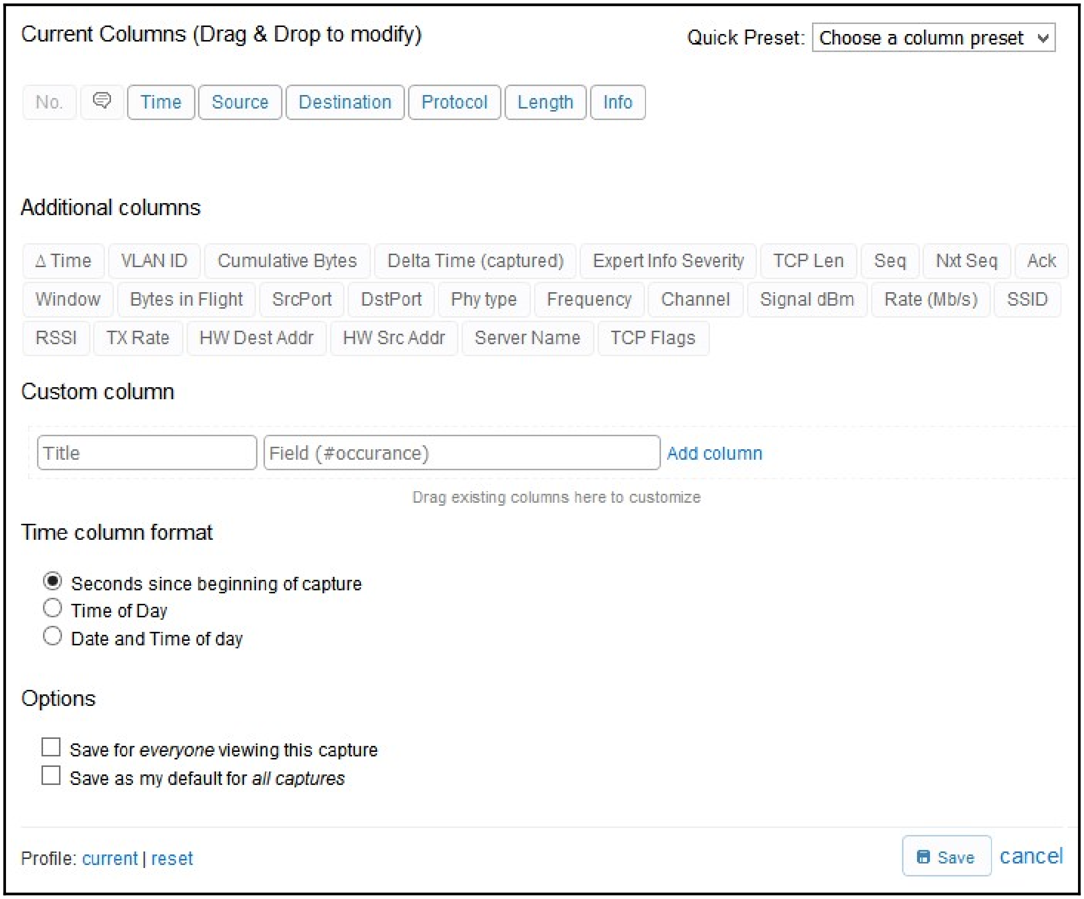

CloudShark Hosted // cloudshark.org
Guest upload is turned off
Log In
SIP-Call-Flow-Over-TCP.pcap 19.4 kb · 84 packets · more info
rtp
✓ Apply
Clear
Analysis Tools
Graphs
Export

No.		Time	Source	Destination	Protocol	Length	Info
13	○	4.398769	10.33.6.101	10.33.6.100	RTP	214	PT=ITU-T G.711 PCMA, SSRC=0x42F433D4, Seq=54339, Time=1884819849
15	○	4.412150	10.33.6.101	10.33.6.100	RTP	214	PT=ITU-T G.711 PCMA, SSRC=0x42F433D4, Seq=54340, Time=1884820009
16	○	4.416138	10.33.6.100	10.33.6.101	RTP	214	PT=ITU-T G.711 PCMA, SSRC=0x5A3361B3, Seq=29371, Time=95878790
18	○	4.435972	10.33.6.101	10.33.6.100	RTP	214	PT=ITU-T G.711 PCMA, SSRC=0x42F433D4, Seq=54341, Time=1884820169
19	○	4.435976	10.33.6.100	10.33.6.101	RTP	214	PT=ITU-T G.711 PCMA, SSRC=0x5A3361B3, Seq=29372, Time=95878950

```
▷ Frame 13: 214 bytes on wire (1712 bits), 214 bytes captured (1712 bits)
▷ Ethernet II, Src: AudioCod_0a:7c:cc (00:90:8f:0a:7c:cc), Dst: AudioCod_0a:39:3a (00:90:8f:0a:39:3a)
▷ Internet Protocol Version 4, Src: 10.33.6.101, Dst: 10.33.6.100
▷ User Datagram Protocol, Src Port: x11 (6050), Dst Port: x11 (6000)
▷ Real-Time Transport Protocol
```

Modified interface

In addition, you can modify the column headers by selecting the **Profile** drop-down menu choice and selecting **Custom Columns**, as shown in the following screenshot:



Time column format

● Seconds since beginning of capture
○ Time of Day
○ Date and Time of day

Options

□ Save for *everyone* viewing this capture
□ Save as my default for *all captures*

Column header preferences

Now that your capture is open and you have modified and customized the interface, you are ready for your analysis.

As you can see, CS offers a great solution to share packet captures with your team. In the next section, we'll evaluate the choices for filtering traffic and visually representing traffic.

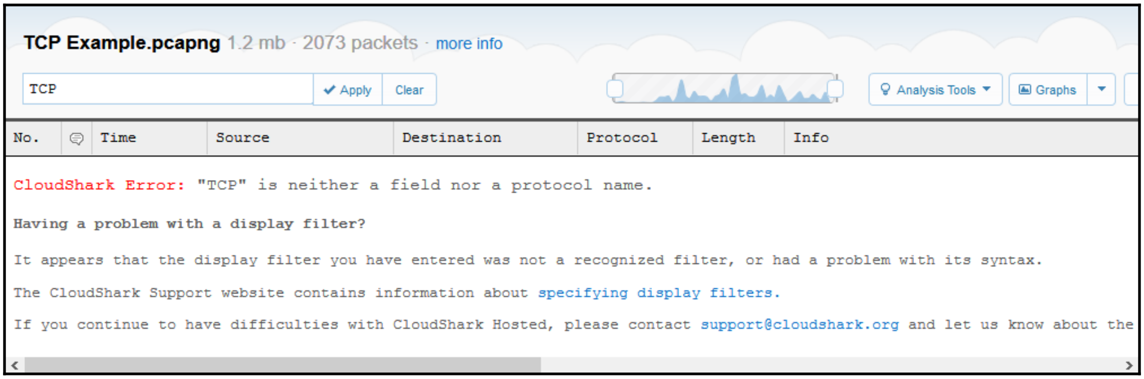
Outlining the various filters and graphs

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 apply a filter and narrow your scope. Let's undertake a deep dive to learn more in the next section.

Displaying data using filters

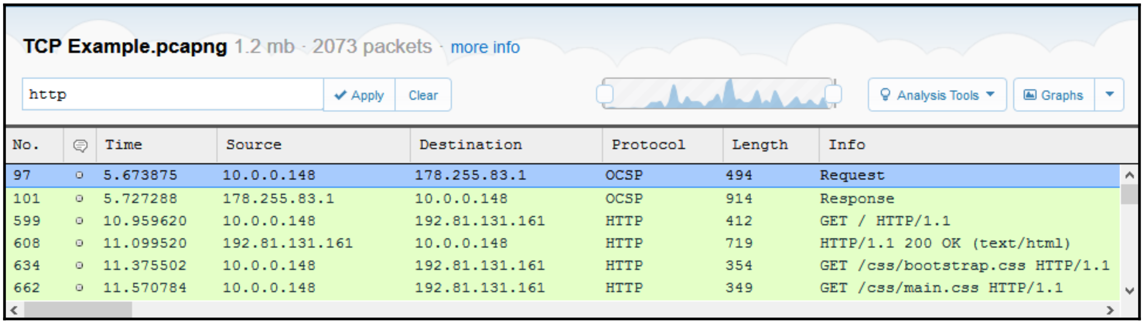
Display filters in CS are very similar to the way Wireshark filters data. Filters can be applied to identify packets with specific ports, IP addresses, or protocols by entering the filter in the upper left-hand side of the interface. Similar to Wireshark, the syntax must be correct, or you will see an error, as shown here:



```
CloudShark Error: "TCP" is neither a field nor a protocol name.

Having a problem with a display filter?

It appears that the display filter you have entered was not a recognized filter, or had a problem with its syntax.

The CloudShark Support website contains information about specifying display filters.

If you continue to have difficulties with CloudShark Hosted, please contact support@cloudshark.org and let us know about the
```

Syntax error

After you enter the filter, select **Apply** to run the filter. I entered the `http` filter, which narrowed the capture to show only HTTP traffic, as shown here:



TCP Example.pcapng 1.2 mb · 2073 packets · more info

http

No.		Time	Source	Destination	Protocol	Length	Info
97	○	5.673875	10.0.0.148	178.255.83.1	OCSP	494	Request
101	○	5.727288	178.255.83.1	10.0.0.148	OCSP	914	Response
599	○	10.959620	10.0.0.148	192.81.131.161	HTTP	412	GET / HTTP/1.1
608	○	11.099520	192.81.131.161	10.0.0.148	HTTP	719	HTTP/1.1 200 OK (text/html)
634	○	11.375502	10.0.0.148	192.81.131.161	HTTP	354	GET /css/bootstrap.css HTTP/1.1
662	○	11.570784	10.0.0.148	192.81.131.161	HTTP	349	GET /css/main.css HTTP/1.1

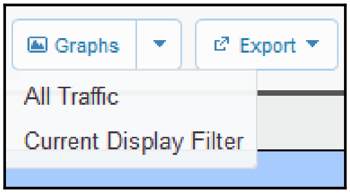
HTTP traffic

In addition to the standard filters, you can also create a search by string or hex values. For example, if I am looking for a specific image, I will enter `frame contains "adc_pet_dog_336x280"` in the display filter, and it will present the results, if any. In addition, you can apply a hex filter to search for hex values. For example, to search for a specific MAC address, you can use `frame contains 28:e3:47:8c:02:60`.

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 represent the data visually.

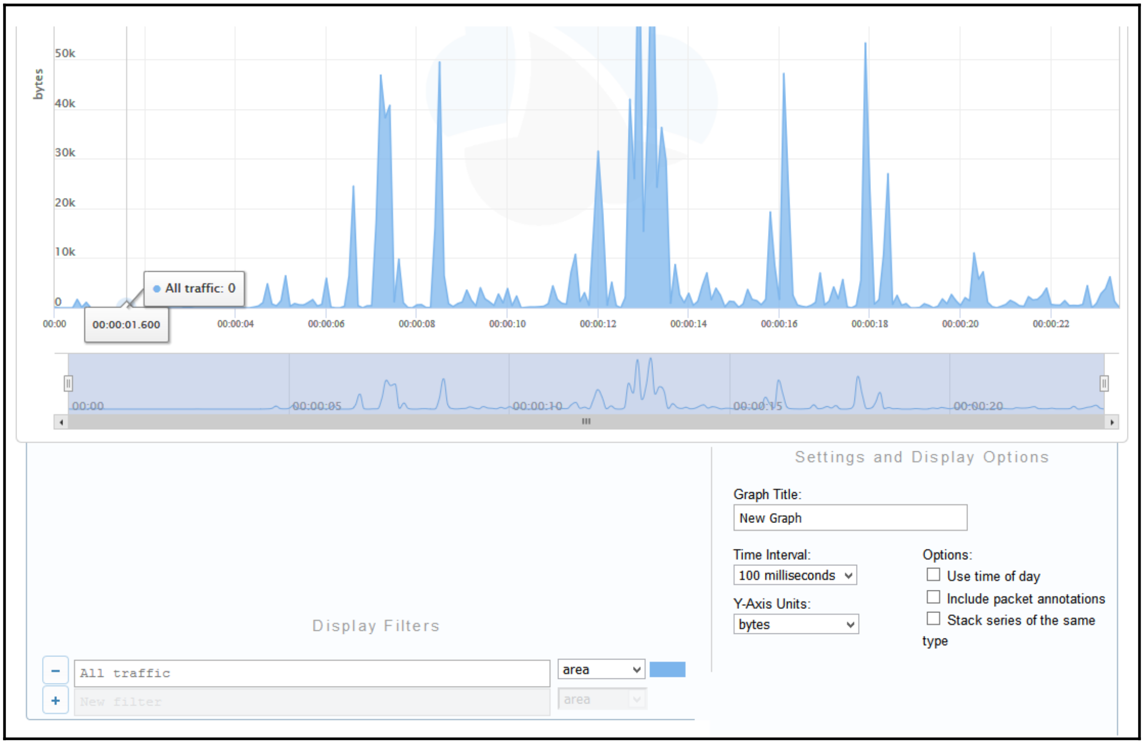
Viewing data using graphs

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 menu for graphs, shown here:



Graphs menu

After you select the type of graph you would like, CS presents the graph. If you would like to create a new graph, you can select the button in the lower left-hand corner. In addition, you can select the **Graphs** button without using the drop-down menu, and this will open a window, where you can select **Create a New Graph,** shown as follows:



Settings and Display Options

Graph Title: New Graph

Time Interval: 100 milliseconds

Y-Axis Units: bytes

Options:
□ Use time of day
□ Include packet annotations
□ Stack series of the same type

Display Filters

− All traffic | area
+ New filter | area

Create a New Graph

Selecting **Create a New Graph** will bring up a window, as shown in the preceding screenshot, that has more options to personalize and modify the graph.

As shown at the bottom of the preceding screenshot, you can add or modify the following:

- **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:
    - **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; that is, line, column, or spline.

Once you have completed the graph, you can export it as either a PNG, JPG, or SVG. In addition, you can print the completed graph.

In addition to graphs and filters, there are times when you need a more advanced analysis of data. The next section provides an overview of a variety of tools for quickly analyzing data.

Evaluating the different analysis tools

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. Once you drop the menu down, the various menu choices are displayed. If any are dimmed, 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.

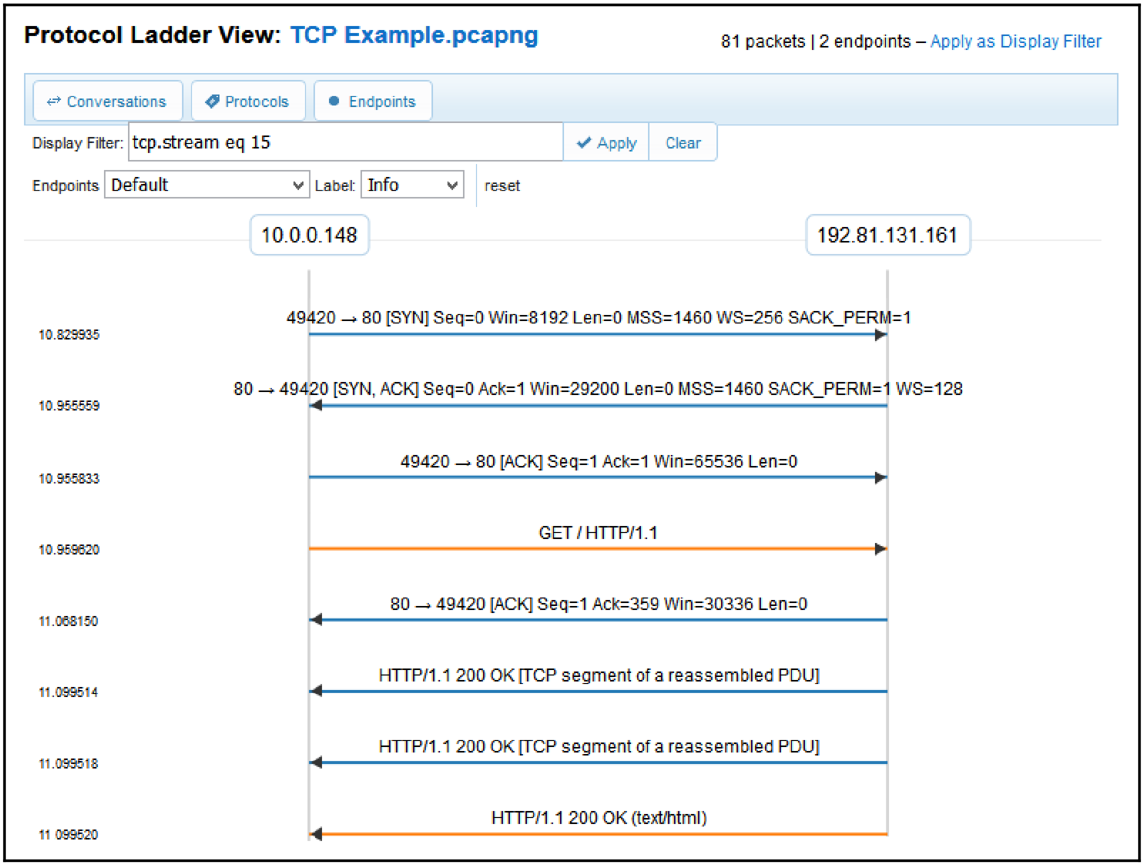
Following the stream and view conversations

Within Wireshark, we have many tools under **Statistics** 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 lists the first selections in the analysis tools menu choice, as follows:

- **Follow stream, SSL, and HTTP**: Similar to the `Follow the Stream` function in Wireshark, this provides a way to see the details of a single conversation between two endpoints.

- **Ladder Diagrams**: These are similar to the flow graphs in Wireshark, showing the endpoints communicating back and forth:


Protocol Ladder View: TCP Example.pcapng
81 packets | 2 endpoints – Apply as Display Filter
Conversations
Protocols
Endpoints
Apply
Clear
reset
10.0.0.148
192.81.131.161

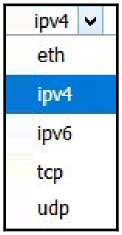
Display Filter: tcp.stream eq 15

Endpoints: Default
Label: Info

```
10.829935  49420 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
10.955559  80 → 49420 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
10.955833  49420 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
10.959620  GET / HTTP/1.1
11.068150  80 → 49420 [ACK] Seq=1 Ack=359 Win=30336 Len=0
11.099514  HTTP/1.1 200 OK [TCP segment of a reassembled PDU]
11.099518  HTTP/1.1 200 OK [TCP segment of a reassembled PDU]
11 099520  HTTP/1.1 200 OK (text/html)
```

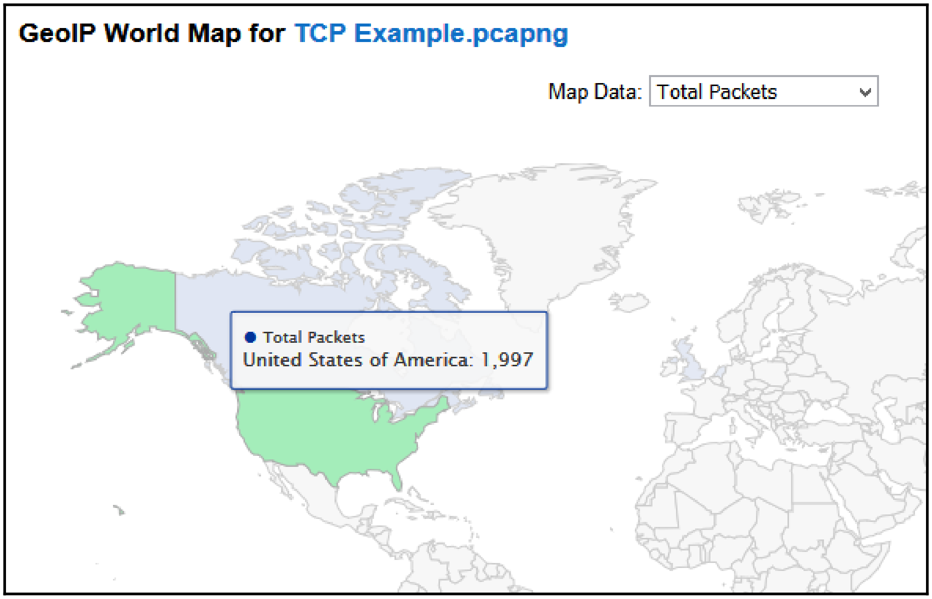
Ladder diagram

- **Network Endpoints**: This will provide a list of endpoints. Similar to Wireshark, while in the window, you can filter by the type of endpoint you would like to see; that is, **eth**, **ipv4**, **ipv6**, **tcp**, or **udp**, as shown in the following screenshot:



Endpoints

- **GeoIP World Map**: At the bottom of the endpoints report, you will see a button to select **GeoIP Map**. When selected, it will show where the packets originate, as shown here:


GeoIP World Map for TCP Example.pcapng

Map Data: Total Packets

● Total Packets
United States of America: 1,997

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: **eth**, **ipv4**, **ipv6**, **tcp**, or **udp**.

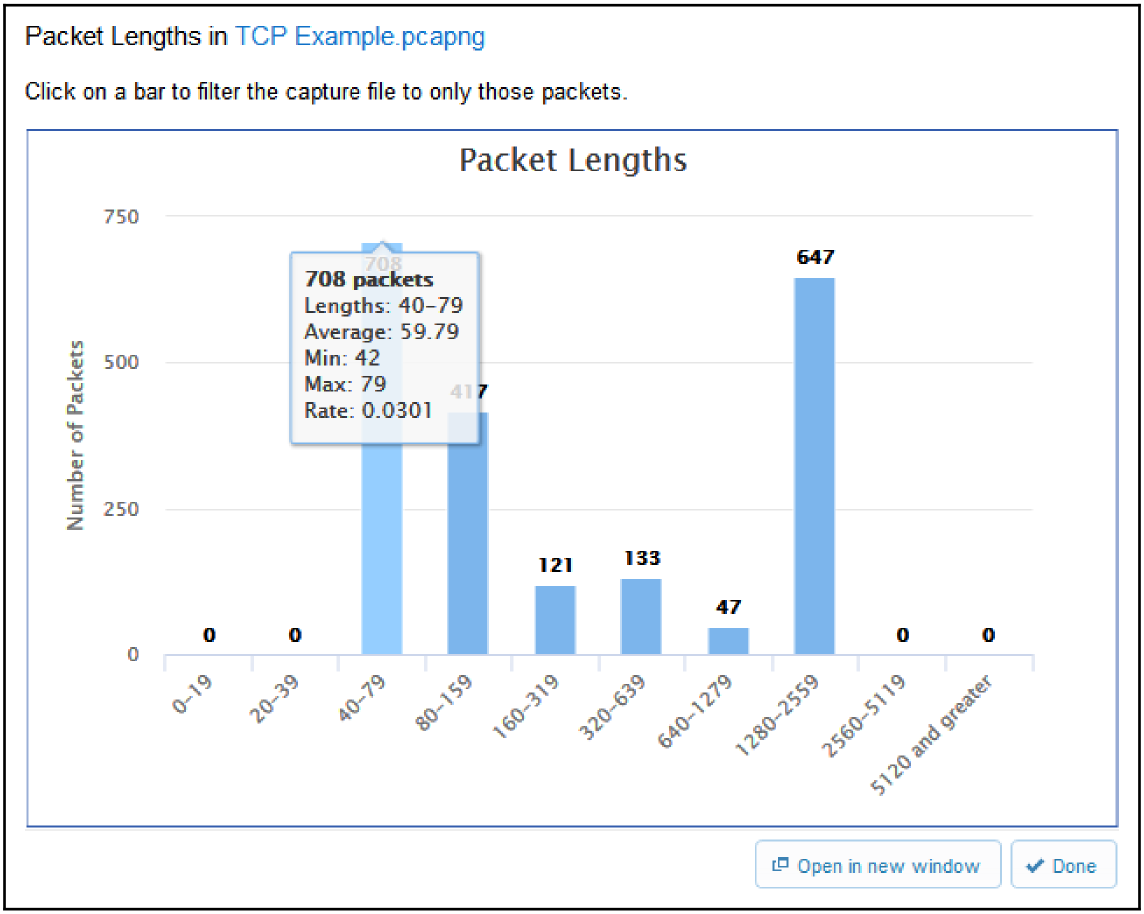
As you can see, CS is populated with many tools that you can use to analyze data. The next section shows how we can take a look at the details of a VoIP call, graph packet lengths, and DNS activity.

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. It's worth running a few of the graphs to see the results.

The next grouping of analysis tools includes the following:

- **Packet Lengths**: This provides an interactive graph of the packet lengths, and other information such as **Average**, **Min**, **Max**, and **Rate**:



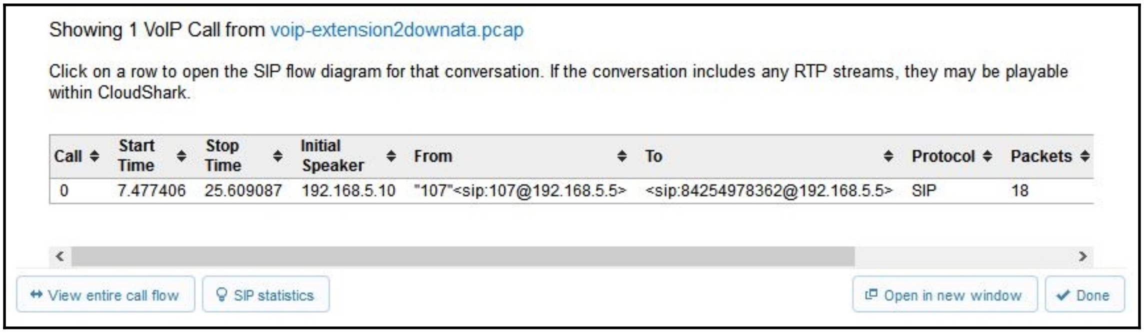
Packet Lengths in TCP Example.pcapng

Click on a bar to filter the capture file to only those packets.

708 packets
Lengths: 40–79
Average: 59.79
Min: 42
Max: 79
Rate: 0.0301

Packet Lengths

- **DNS Activity**: This provides a count of the DNS queries, responses, and **Resource Record** (**RR**) types.
- **VoIP Calls**: This provides a list of any VoIP calls found in the file along with a sortable summary of the **Call**, **Start Time**, **Stop Time**, **Initial Speaker**, **From**, **To**, **Protocol**, and **Packets**, as shown in the following screenshot:



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

CloudShark VoIP calls

- **RTP Streams**: This lists the **Real-Time Transport Protocol** (**RTP**) streams found in the file.

As you can see, there are several helpful analysis tools. This last section helps dissect wireless problems and conducts a quick threat assessment.

Exploring wireless, protocols, and possible threats

While there are many tools that are similar to those found in Wireshark, this last grouping contains an analysis tool unique to CS, which is Threat Assessments. This tool will allow you to run your capture and see whether any suspicious packets are flagged.

This last section covers the following tools:

- **HTTP Analysis**: This lists all of the URLs requested in the capture file, along with a count of the requests.
- **Decode Protocol As**: If CS doesn't decode the protocol correctly, you can provide values so that CS can properly decode the protocol.

- **Wireless Networks**: This provides a list of any wireless networks found in the file along with a sortable summary of the following: BSSID, SSID, vendor, `Signal_dBm`, channel, and security.
- **Wireless Keys**: This will open a dialog box to add any decryption keys for the wireless networks in the file.
- **Threat Assessments**: This is a more advanced option that will scan the capture for potentially malicious traffic within it. If none are found, the report will come back with the all clear!

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.

Discovering where to find sample captures

While learning about packet analysis, it's important to study a variety of captures until you are proficient. This may take a while, but it will be well worth the effort. There are many online repositories for packet captures. Here are a few websites that can give you a variety of real network traffic:

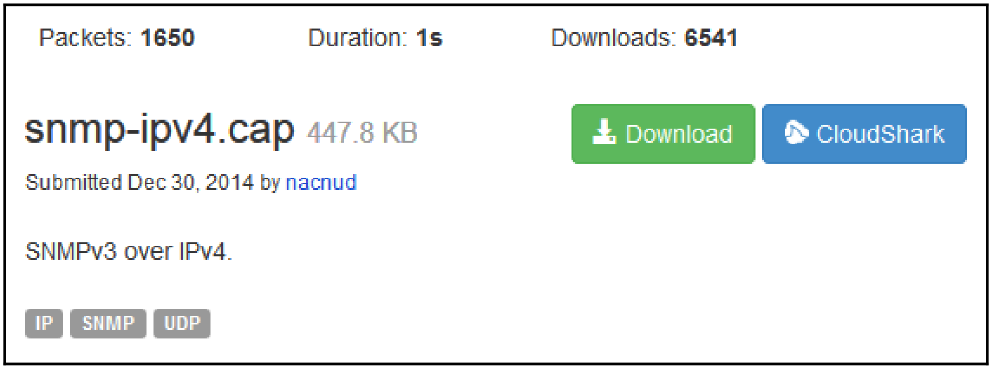
- `https://wiki.wireshark.org/SampleCaptures`
- `http://tcpreplay.appneta.com/wiki/captures.html`
- `https://www.netresec.com/?page=PcapFiles`
- `https://chrissanders.org/packet-captures/`

This is only a partial list of where you can get samples to hone your skills. Let's now take a look at a handy way to open a packet capture, right in CS.

Downloading captures

Now that you have seen where to get packet captures, you may want to learn about an unfamiliar protocol with your team. As shown, there are many places to obtain packet captures; however, one site I visit often is `https://packetlife.net/`.

Once on the PacketLife site, navigate to **Captures**, found at `http://packetlife.net/captures/`. On the packet capture page, you can upload a capture or search for captures. For example, I found `snmp-ipv4.pcap`, as shown in the following screenshot:



Packets: **1650** Duration: **1s** Downloads: **6541**

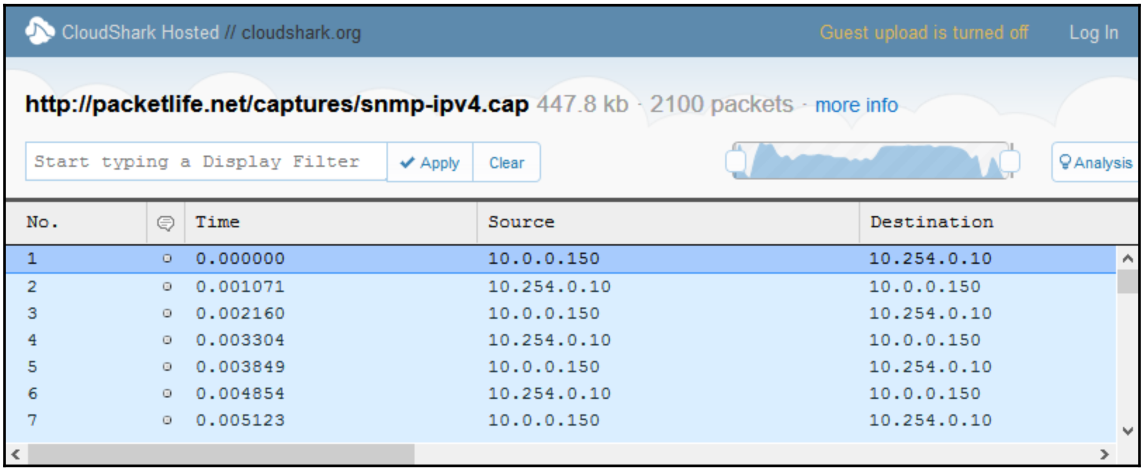
snmp-ipv4.cap 447.8 KB

Submitted Dec 30, 2014 by nacnud

SNMPv3 over IPv4.

Packet capture found at PacketLife

Once you have found a packet capture, you can either download it and open it in Wireshark, or open it directly in CS, as shown here:



CloudShark Hosted // cloudshark.org
Guest upload is turned off
Log In
Start typing a Display Filter
Apply
Clear
Analysis

http://packetlife.net/captures/snmp-ipv4.cap 447.8 kb · 2100 packets · more info

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

Packet capture opened in CS

Once in CS, you can use its variety of built-in tools to study the capture. You can even download the file and open it in Wireshark, for better visualization or a more advanced analysis of the data.

As you can see, there are many online repositories for sample captures. Visit them, download some captures, and continue to improve your packet analysis skills.

Summary

In this chapter, we took a look at CS, which allows you to view and analyze packet captures in a browser. We learned that CS provides several ways of examining captures that are similar to Wireshark.

We discovered that, in general, there are many resources for packet captures that you can visit and download a file to study and improve your packet analysis skills. We then took a look at PacketLife, which has an online repository of capture files for download, or an option to open them and analyze them in CS.

We saw that, with CS, you can filter a capture to show only a specific type of traffic, as well as creating a variety of graphs. In addition, CS has a rich variety of analysis tools that include Follow Stream, network endpoints, a GeoIP world map, packet lengths, DNS activity, VoIP calls, wireless networks, and threat assessments.

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*:

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.
    1. Account
    2. Uploads
    3. Decode Window
    4. Capture Index
2. In addition to the standard filters, you can also search for a specific image by using a _____ filter.
    1. Decode
    2. String
    3. Hex
    4. Craft

3. __ are similar to the flow graphs in Wireshark, which show the endpoints communicating back and forth.
    1. Ladder diagrams
    2. Step charts
    3. VoIP ladders
    4. Time intervals
4. At the bottom of the endpoints report, you will see a button to select ___. When selected, it will visually show where the packets originate.
    1. Step charts
    2. Craft
    3. Time interval
    4. GeoIP Map
5. __ is a more advanced option that will scan the capture for potentially malicious traffic within the capture.
    1. Malicious conversations
    2. Malicious endpoints
    3. Threat assessments
    4. Threat maps

Assessment

Chapter 1: Appreciating Traffic Analysis

1. 1990s
2. EINSTEIN
3. Chat
4. Baseline
5. Reactive

Chapter 2: Using Wireshark NG

1. 2006
2. 1998
3. pcap
4. Dissectors
5. Mergecap

Chapter 3: Installing on a PC or macOS

1. X11
2. libpcap
3. TShark
4. mmdbresolve
5. NpCap

Chapter 4: Exploring the Wireshark Interface

1. Clear
2. Previously Displayed Packet
3. Colorize Packet List
4. View
5. Edit

Chapter 5: Tapping into the Data Stream

1. LAN
2. manuf.txt
3. Output
4. Single mode
5. .pcapng

Chapter 6: Personalizing the Interface

1. Default
2. Edit and Appearance
3. Red
4. Edit and Preferences
5. Edit

Chapter 7: Using Display and Capture Filters

1. Green
2. Value
3. udp port 53
4. Expression
5. ...and not Selected

Chapter 8: Outlining the OSI Model

1. Session
2. Transport
3. Presentation
4. Well-known
5. Segment

Chapter 9: Decoding TCP and UDP

1. Socket
2. 724 and 725
3. PSH
4. 20 (0101 =5; 5 x 4 =20)
5. DHCP

Chapter 10: Managing TCP Connections

1. frame.marked==1
2. Window scale
3. SACK
4. FIN
5. 936

Chapter 11: Analyzing IPv4 and IPv6

1. Network control
2. Class B private IPv4 address
3. 128
4. Hop count
5. 3

Chapter 12: Discovering ICMP

1. Unreachable
2. Deprecated
3. Parameter problems
4. 3
5. Ping

Chapter 13: Understanding ARP

1. Data link
2. 1 and 2
3. RARP
4. Gratuitous ARP
5. Storm

Chapter 14: Troubleshooting Latency Issues

1. Latency
2. Throughput
3. Intelligent Scrollbar
4. Keep-alive
5. A note

Chapter 15: Subsetting, Saving, and Exporting Captures

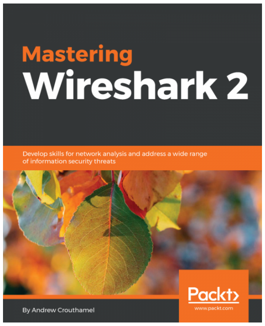
1. Conversation
2. Protocol Hierarchy
3. .pcapng
4. Mark
5. HTTP

Chapter 16:Using CloudShark for Packet Analysis

1. Capture Index
2. String
3. Ladder diagrams
4. GeoIP Map
5. Threat assessments

Other Books You May Enjoy

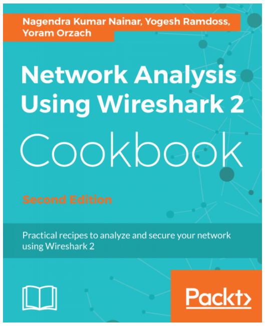
If you enjoyed this book, you may be interested in these other books by Packt:


Mastering
Wireshark 2
Develop skills for network analysis and address a wide range of information security threats
By Andrew Crouthamel
Packt>
www.packt.com

Mastering Wireshark 2
Andrew Crouthamel

ISBN: 978-1-78862-652-1

- Understand what network and protocol analysis is and how it can help you
- Use Wireshark to capture packets in your network
- Filter captured traffic to only show what you need
- Explore useful statistic displays to make it easier to diagnose issues
- Customize Wireshark to your own specifications
- Analyze common network and network application protocols


Nagendra Kumar Nainar, Yogesh Ramdoss, Yoram Orzach
Network Analysis Using Wireshark 2
Cookbook
Second Edition
Practical recipes to analyze and secure your network using Wireshark 2
Packt

Network Analysis Using Wireshark 2 Cookbook
Yoram Orzach, Nagendra Kumar Nainar, Et al

ISBN: 978-1-78646-167-4

- Configure Wireshark 2 for effective network analysis and troubleshooting
- Set up various display and capture filter
- Understand networking layers, including IPv4 and IPv6 analysis
- Explore performance issues in TCP/IP
- Get to know about Wi-Fi testing and how to resolve problems related to wireless LANs
- Get information about network phenomena, events, and errors
- Locate faults in detecting security failures and breaches in networks

Leave a review - let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

Index