



Community Experience Distilled

Packet Analysis with Wireshark

Leverage the power of Wireshark to troubleshoot your networking issues by using effective packet analysis techniques and performing an improved protocol analysis

Anish Nath

[PACKT] open source*
PUBLISHING community experience distilled

Packet Analysis with Wireshark

Table of Contents

[Packet Analysis with Wireshark](#)

[Credits](#)

[About the Author](#)

[About the Reviewers](#)

[www.PacktPub.com](#)

[Support files, eBooks, discount offers, and more](#)

[Why subscribe?](#)

[Free access for Packt account holders](#)

[Preface](#)

[What this book covers](#)

[What you need for this book](#)

[Who this book is for](#)

[Conventions](#)

[Reader feedback](#)

[Customer support](#)

[Downloading the example code](#)

[Errata](#)

[Piracy](#)

[Questions](#)

[1. Packet Analyzers](#)

[Uses for packet analyzers](#)

[Introducing Wireshark](#)

[Wireshark features](#)

[Wireshark's dumpcap and tshark](#)

[The Wireshark packet capture process](#)

[Other packet analyzer tools](#)

[Mobile packet capture](#)

[Summary](#)

[2. Capturing Packets](#)

[Guide to capturing packets](#)

[Capturing packets with Interface Lists](#)

[Common interface names](#)

[Capturing packets with Start options](#)

[Capturing packets with Capture Options](#)

[The capture filter options](#)

[Auto-capturing a file periodically](#)

[Troubleshooting](#)

[Wireshark user interface](#)

[The Filter toolbar](#)

[Filtering techniques](#)

[Filter examples](#)

[The Packet List pane](#)

[The Packet Details pane](#)

[The Packet Bytes pane](#)

[Wireshark features](#)

[Decode-As](#)

[Protocol preferences](#)

[The IO graph](#)

[Following the TCP stream](#)

[Exporting the displayed packet](#)

[Generating the firewall ACL rules](#)

[Tcpdump and snoop](#)

[References](#)

[Summary](#)

[3. Analyzing the TCP Network](#)

[Recapping TCP](#)

[TCP header fields](#)

[TCP states](#)

[TCP connection establishment and clearing](#)

[TCP three-way handshake](#)

[Handshake message – first step \[SYN\]](#)

[Handshake message – second step \[SYN, ACK\]](#)

[Handshake message – third step \[ACK\]](#)

[TCP data communication](#)

[TCP close sequence](#)

[Lab exercise](#)

[TCP troubleshooting](#)

[TCP reset sequence](#)

[RST after SYN-ACK](#)

[RST after SYN](#)

[Lab exercise](#)

[TCP CLOSE_WAIT](#)

[Lab exercise](#)

[How to resolve TCP CLOSE_STATE](#)

[TCP TIME_WAIT](#)

[TCP latency issues](#)

[Cause of latency](#)

[Identifying latency](#)

[Server latency example](#)

[Wire latency](#)

[Wireshark TCP sequence analysis](#)

[TCP retransmission](#)

[Lab exercise](#)

[TCP ZeroWindow](#)

[TCP Window Update](#)

[TCP Dup-ACK](#)

[References](#)

[Summary](#)

[4. Analyzing SSL/TLS](#)

[An introduction to SSL/TLS](#)

[SSL/TLS versions](#)

[The SSL/TLS component](#)

[The SSL/TLS handshake](#)

[Types of handshake message](#)

[Client Hello](#)

[Server Hello](#)

[Server certificate](#)

[Server Key Exchange](#)

[Client certificate request](#)

[Server Hello Done](#)

[Client certificate](#)

[Client Key Exchange](#)

[Client Certificate Verify](#)

[Change Cipher Spec](#)

[Finished](#)

[Application Data](#)

[Alert Protocol](#)

[Key exchange](#)

[The Diffie-Hellman key exchange](#)

[Elliptic curve Diffie-Hellman key exchange](#)

[RSA](#)

[Decrypting SSL/TLS](#)

[Decrypting RSA traffic](#)

[Decrypting DHE/ECHDE traffic](#)

[Forward secrecy](#)

[Debugging issues](#)

[Summary](#)

[5. Analyzing Application Layer Protocols](#)

[DHCPv6](#)

[DHCPv6 Wireshark filter](#)

[Multicast addresses](#)

[The UDP port information](#)

[DHCPv6 message types](#)

[Message exchanges](#)

[The four-message exchange](#)

[The two-message exchange](#)

[DHCPv6 traffic capture](#)

[BOOTP/DHCP](#)

[BOOTP/DHCP Wireshark filter](#)

[Address assignment](#)

[Capture DHCPv4 traffic](#)

[DNS](#)

[DNS Wireshark filter](#)

[Port](#)

[Resource records](#)

[DNS traffic](#)

[HTTP](#)

[HTTP Wireshark filter](#)

[HTTP use cases](#)

[Finding the top HTTP response time](#)

[Finding packets based on HTTP methods](#)

[Finding sensitive information in a form post](#)

[Using HTTP status code](#)

[References](#)

[Summary](#)

[6. WLAN Capturing](#)

[WLAN capture setup](#)

[The monitor mode](#)

[Analyzing the Wi-Fi networks](#)

[Frames](#)

[Management frames](#)

[Data frames](#)

[Control frames](#)

[802.11 auth process](#)

[802.1X EAPOL](#)

[The 802.11 protocol stack](#)

[Wi-Fi sniffing products](#)

[Summary](#)

[7. Security Analysis](#)

[Heartbleed bug](#)

[The Heartbleed Wireshark filter](#)

[Heartbleed Wireshark analysis](#)

[The Heartbleed test](#)

[Heartbleed recommendations](#)

[The DOS attack](#)

[SYN flood](#)

[SYN flood mitigation](#)

[ICMP flood](#)

[ICMP flood mitigation](#)

[SSL flood](#)

[Scanning](#)

[Vulnerability scanning](#)

[SSL scans](#)

[ARP duplicate IP detection](#)

[DrDoS](#)

[BitTorrent](#)

[Wireshark protocol hierarchy](#)

[Summary](#)

[Index](#)

Packet Analysis with Wireshark

Packet Analysis with Wireshark

Copyright © 2015 Packt Publishing

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

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

First published: November 2015

Production reference: 1261115

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78588-781-9

www.packtpub.com

Credits

Author

Anish Nath

Reviewers

Michael Downey

Robert Juric

Mikael Kanstrup

Acquisition Editor

Indrajit Das

Content Development Editor

Rohit Singh

Technical Editor

Mrunmayee Patil

Copy Editor

Stephen Copestake

Project Coordinator

Mary Alex

Proofreader

Safis Editing

Indexer

Monica Ajmera Mehta

Production Coordinator

Nilesh Mohite

Cover Work

Nilesh Mohite

About the Author

Anish Nath is a software engineer who has more than 10 years of experience. He works at CISCO, and at CISCO, he started using Wireshark for the first time. He is thankful to CISCO. He doesn't speak much, but likes to explore new things that he has not tried or not thought of. He also tries his best to be successful at this. Though he fails a lot of time, this gives him more experience, and when success comes, he thanks all of his efforts that had failed him initially.

You can reach him at <https://in.linkedin.com/in/anishnath>, and his Twitter handle is @anish2good.

I would like to thank my friends, Arnab Biswas, Arun John, Ganesh Choudhari, Mayank Johari, Pradeep Sivakumar, Prakash John, Deepak Kukrety, and Veeksha Vasant for supporting me in this venture. I've definitely learned a lot from their experience.

I would also like to thank, Alice Chen, Tin Nguyen, Sunil Menon, Saad Abderrazzaq, Ori Lior, Mahin Khani, Donn Coe, Rob Andrews, and Lon Barrett, for their support and belief in me all this time and also for providing me assistance when I needed it.

Special thanks to the Wireshark community and its developers for writing an awesome tool like this.

Thanks to all my reviewers who made an effort so that this book took the correct shape.

My apologies if I've missed anyone.

Thanks to Packt Publishing and the entire team, especially Indrajit Das and Rohit Singh for making this happen.

About the Reviewers

Michael Downey is a security analyst with a passion for *nix operating systems and network security monitoring. He is also the cofounder of the Evansville Linux User Group in Indiana, and a contributing member of OpenNSM (<http://www.open-nsm.net/>). In his free time, he enjoys security research and an occasional game of disc golf.

Robert Juric, while working as a network engineer, has supported government agencies, large corporations, and service providers. From his experience, he learned the value of packet analysis and has come to enjoy the details that it provides.

When not at work, Robert enjoys spending time outdoors with his wife and young son. He occasionally writes articles for his website, robertjuric.com, or can be found on Twitter at @robertj180.

Mikael Kanstrup is a software engineer with a passion for adventure and thrills in life. In his spare time, he spends his time kitesurfing, riding motocross, or just going outdoors with his family and two kids. Mikael has a BSc degree in computer science and years of experience in embedded software development and computer networking. For the past decade, he has been working as a professional software developer in the mobile phone industry.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at <service@packtpub.com> for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

I would like to dedicate this book to my 5-year old son, Arjun Nath; grandfather, Sri Rajeshwar Prasad; wife, Manisha Prasad; mother, Indu Sinha; and all my family members (my father, Anil Kumar Sinha; chote papa, Sunil Kumar Sinha; choti mummy, Poonam Sinha; and friends). Without them, this would not have been possible.

Preface

The purpose of this book is to identify, learn about, and solve issues related to protocol, network, and security, and see how Wireshark helps to analyze these patterns by allowing its features to troubleshoot effectively. This book has lab exercises and contains packet capture files for offline viewing and analyses. Most of the examples contain production-like scenarios and their solutions and steps to reproduce these solutions.

This book also contains effective capturing methods that can be used directly in production without installing Wireshark.

Wireshark is an awesome tool for troubleshooting and learning, and within the scope of this book, we have taken the best use cases for different types of audiences, such as network administrators, security auditors, protocol learners, and troubleshooters.

What this book covers

[Chapter 1](#), *Packet Analyzers*, covers the definition of packet analyzers and their use cases, network interfaces naming conventions, pcap/pcanpng file extensions, and types of network analyzer tools.

[Chapter 2](#), *Capturing Packets*, covers how to capture packets using Wireshark, tcpdump, and snoop; how to use Wireshark display filters; and how to use Wireshark's cool features such as Decode-As and protocol preferences. Also, we will cover the TCP stream, exporting images, generating a firewall ACL rule, autocapture setup, and the name resolution feature.

[Chapter 3](#), *Analyzing the TCP Network*, covers the TCP state machine, TCP connection establishment and closing sequence, practical troubleshooting labs such as (CLOSE_WAIT, TIME_WAIT), how to identify and fix latency issues, and Wireshark TCP sequence analysis flag (zero window, dup-ok, TCP retransmission, and window update) features.

[Chapter 4](#), *Analyzing SSL/TLS*, covers the TLS/SSL two-way mutual authentication process with Wireshark, SSL/TLS decryption with Wireshark, and the identification of handshake failure with Wireshark.

[Chapter 5](#), *Analyzing Application Layer Protocols*, covers how to analyze a protocol using the Wireshark display filter, how these protocols work, how to simulate these packets, capture, and display them using tcpdump/Wireshark.

[Chapter 6](#), *WLAN Capturing*, covers WLAN capture setup and monitor mode, capturing with tcpdump, 802.11 display filters, Layer-2 datagram frames types, Wireshark display filters, and other Wi-Fi Sniffing products available.

[Chapter 7](#), *Security Analysis*, covers the security aspect with Wireshark and discusses uses cases such as the Heartbleed bug, SYN flood/mitigation, ICMP flood/mitigation, MITM, BitTorrent, and host scanning.

What you need for this book

The topics covered in this book require a basic understanding of TCP/IP. The examples used in this book are independent of an operating system. All the examples are executed in a MAC and Linux OS. Windows users can install Cygwin to use a Linux command-line utility. The following executables are used in this book:

- Wireshark
- tcpdump
- snoop
- dig
- nslookup
- java
- wget
- dhclient
- nmap

Who this book is for

This book provides background information to help readers understand the topics that are discussed. The intended audience for this book includes the following:

- Network/system administrators
- Security consultants and IT officers
- Architects/protocol developers
- White Hat hackers

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: “Start Wireshark by clicking on the Wireshark icon or type `wireshark` in the command line.”

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

```
[bash ~]# cat /proc/sys/net/ipv4/tcp_fin_timeout 60
```

New **terms** and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: “Click on **Interface List**; Wireshark will show a list of available network interfaces in the system.”

Note

Warnings or important notes appear in a box like this.

Tip

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail <feedback@packtpub.com>, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files from your account at <http://www.packtpub.com> for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at <copyright@packtpub.com> with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

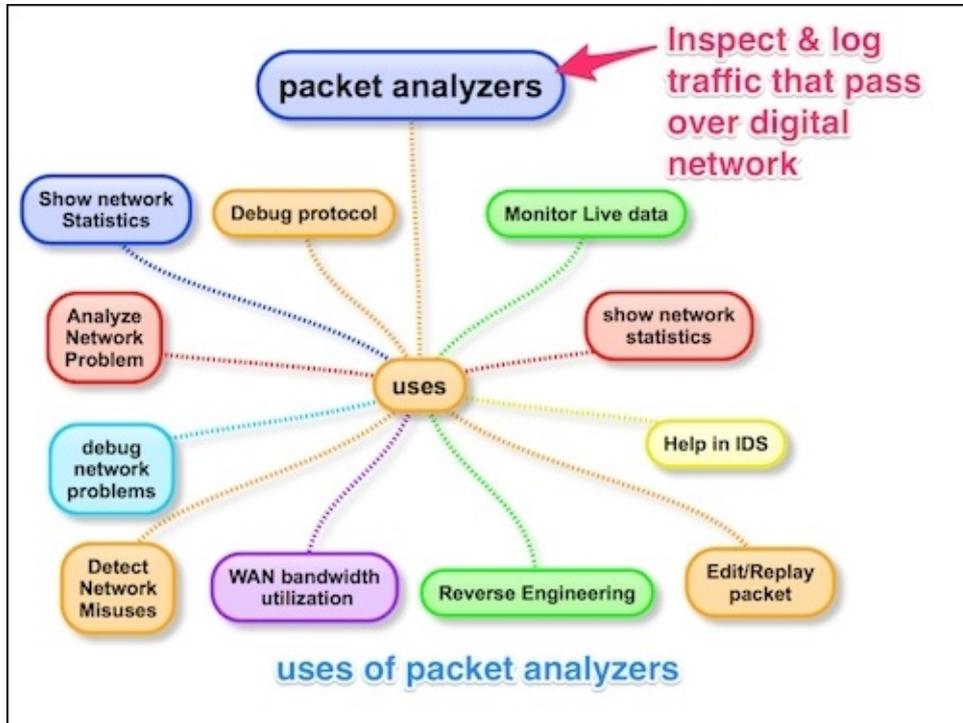
Chapter 1. Packet Analyzers

A packet analyzer is also known as a packet sniffer or a network protocol analyzer. Packet analyzer has the ability to grab the raw packet from the wire, wireless, Bluetooth, VLAN, PPP, and other network types, without getting processed by the application. By doing so it brings the whole science and innovation to this field. In this chapter we will see a few use cases of the packet analyzer by covering the following topics:

- Uses for packet analyzers
- Introducing Wireshark
- Other packet analyzer tools
- Mobile packet capturing

Uses for packet analyzers

More practically, packet analyzers are employed in network security and to analyze raw traffic so as to detect scans and attacks, and for sniffing, network troubleshooting, and many more uses, as shown in the following image:



Packet analyzers can be used as follows:

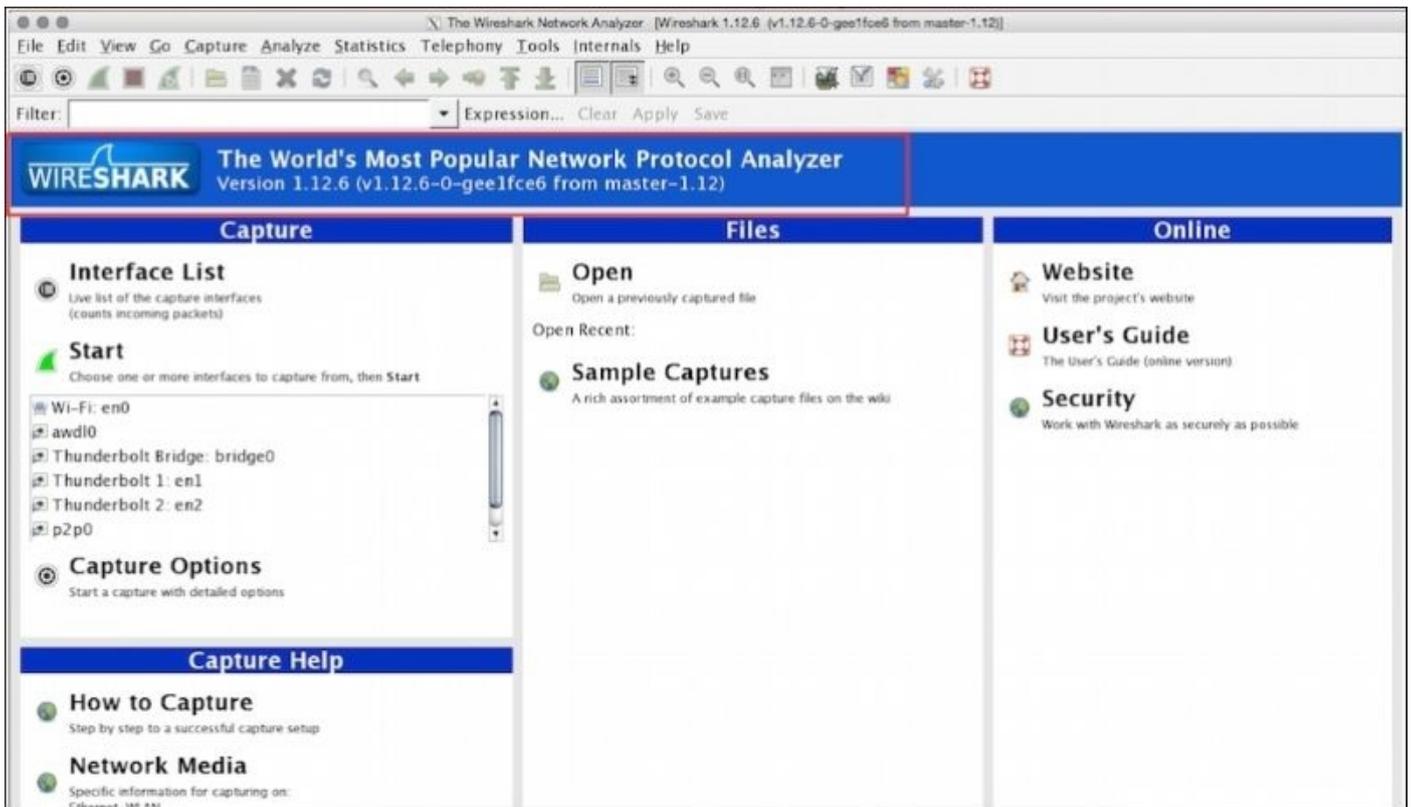
- Network administrators can diagnose problems on a network
- Security architects can perform a security audit on a packet
- Protocol developers can diagnose/learn protocol-related issues
- White-hat hackers can find vulnerabilities in the application and fix them before black-hat hacker find them

The use is not limited to these bullet point, there are lots of new tools and innovations happening in this area. Find a use case and build your own packet analyzer; the best example is Wireshark.

Introducing Wireshark

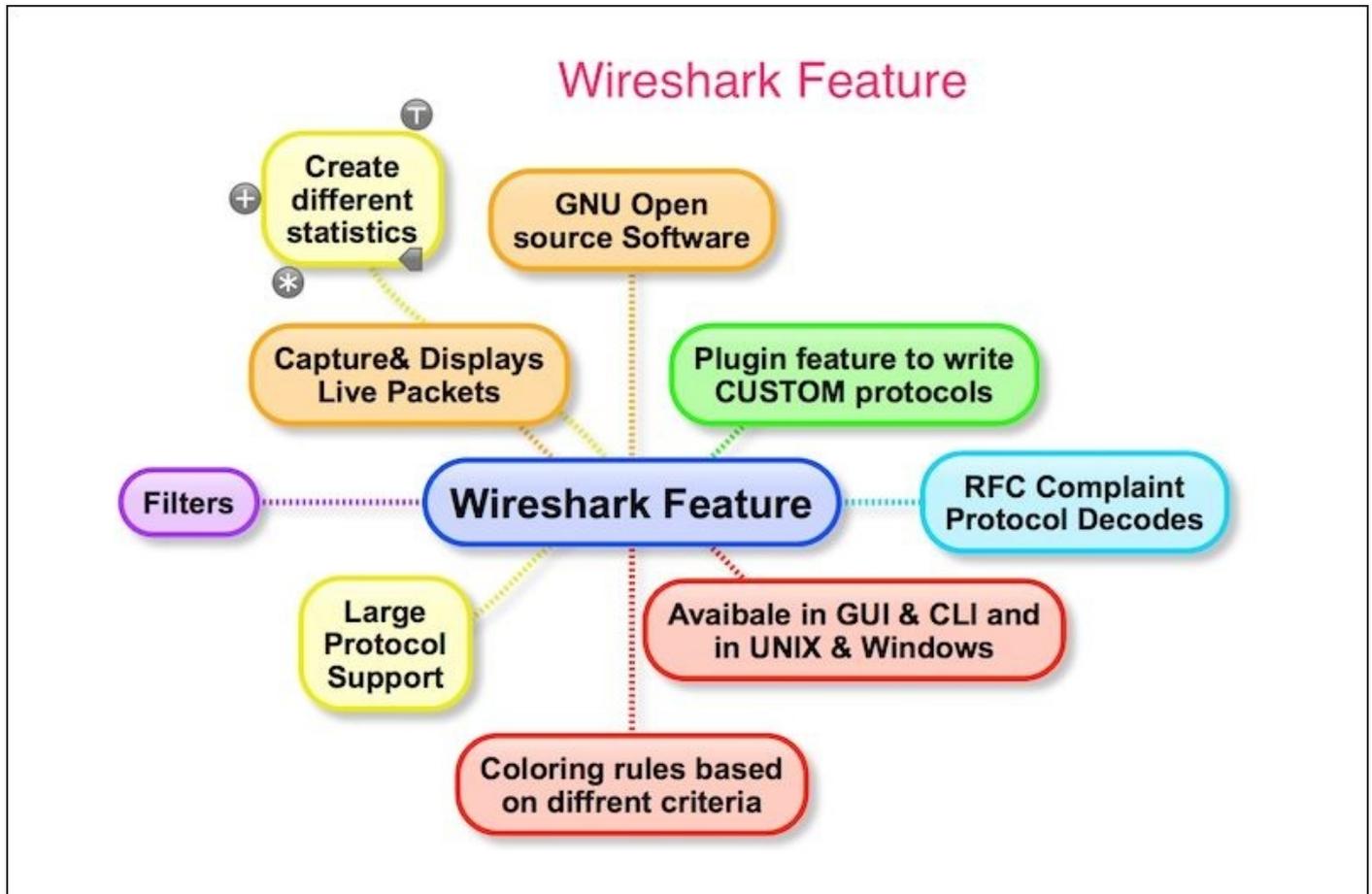
Wireshark is perhaps one of the best open source packet analyzers available today. Wireshark is a powerful packet analyzer tool, with an easy-to-use, rich GUI and a command-line utility with very active community support: <http://ask.wireshark.org>.

Wireshark uses pcap (libpcap) to capture packets, which means it can capture packets in offline mode—to view the captured packets—and online mode (live traffic) to capture and display the traffic in the Wireshark GUI. Once open, the Wireshark GUI looks like this:



Wireshark features

We will see some of the important features that are available in Wireshark in the following figure:



Wireshark has the following cool built-in features, few of them are listed as follows:

- Available in both UNIX and Windows
- Ability to capture live packets from various types of interface
- Filters packets with many criteria
- Ability to decode larger sets of protocols
- Can save and merge captured packets
- Can create various statistics
- User-friendly GUI and command-line interface
- Active community support (<http://ask.wireshark.org>)

Wireshark's dumpcap and tshark

The Wireshark installation provides some command-line tools such as `dumpcap` and `tshark`. Wireshark and `tshark` rely on `dumpcap` to capture traffic; more advanced functionality is performed by `tshark`. Also note that `dumpcap` can be run as its own standalone utility. `tshark` is a command-line version of Wireshark and can be used in the remote terminal.

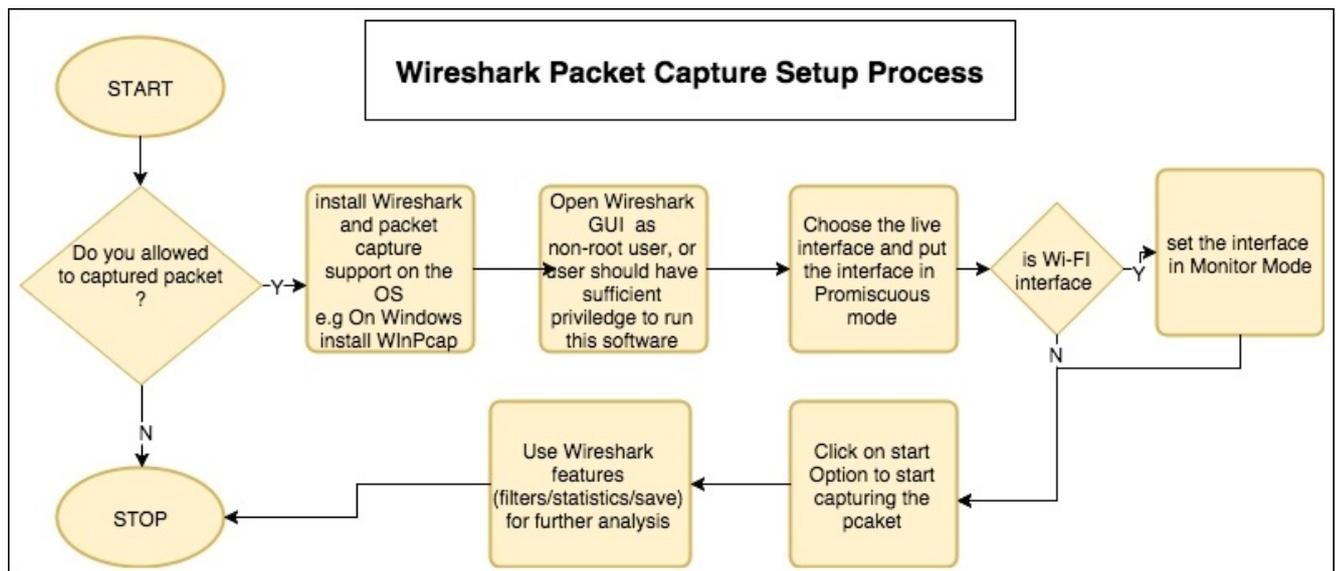
The Wireshark packet capture process

The user must be aware of where Wireshark is installed and it should be obliged with your organization policy before start capturing on the **TAP (Test Access Point)** or **Switch Port Analyzer (SPAN)** port.

Usually developers install Wireshark on their personal laptop/desktop and capture packets, which goes in-out from the box.

Certain guidelines should be followed to perform this:

1. Make sure you're allowed to do what you're going to do; check your corporate policies before capturing a packet.
2. The operating system must support packet capturing:
 - Linux packet socket support is enabled in the kernel by default
 - Windows requires WinPcap to be installed
3. Choose the interface and enable the promiscuous mode on it. Promiscuous mode accepts all packets whether they are addressed to the interface or not.
4. If using a Wi-Fi interface, enable the monitor mode for WLAN capturing.
5. Start capturing and use Wireshark's different features like (filters/statistics/IO/save) for further analysis



Other packet analyzer tools

Wireshark is a packet analysis tool to use features such as packet editing/replaying, performing MITM, ARPspooof, IDS, and HTTP proxy, and there are other packet analyzer tools available and can be used as well.

The following is a list (not limited) of notable packet analyzer tools on the market; many others are commercially available. The table lists tools and their features:

Tools	Packet editing	Packet replay	ARPspooof/MITM	Password sniffing	Intrusion detection	HTTP debugger
WireEdit (https://wireedit.com/)	Y	N	N	N	N	N
Scapy (http://www.secdev.org/)	Y	Y	Y	Y	N	Y
Ettercap (https://ettercap.github.io/ettercap/)	Y	N	Y	Y	N	N
Tcpreplay (http://tcpreplay.synfin.net/)	N	Y	N	N	N	N
Bit-Twist (http://bittwist.sourceforge.net/)	Y	N	N	N	N	N
Cain (http://www.oxid.it/cain.html)	N	N	Y	Y	N	N
Snort (https://www.snort.org/)	N	N	N	N	Y	N

Mobile packet capture

Wireshark is not available on mobile platforms such as Android, iOS, or Windows. In order to capture mobile traffic the following tools are suggested based on the platform:

Platform	Packet capture tool used	URL
Windows	Microsoft Network Analyzers	http://www.microsoft.com/en-in/download/details.aspx?id=19484
iOS	Paros	http://sourceforge.net/projects/paros/
Android	Shark for Root	http://www.appbrain.com/app/shark-for-root/lv.n3o.shark
	Kismet Android PCAP	http://www.kismetwireless.net/android-pcap/

Various other techniques are used to capture mobile traffic using Wireshark. One such technique is creating a Wi-Fi hotspot on the laptop, allowing the mobile phone to use this Wi-Fi, and sniffing traffic on your Wi-Fi interface using Wireshark.

Summary

In this chapter we learned what packet analyzers are and what their use cases are. After a quick introduction to Wireshark, we covered what goes on behind-the-scenes when Wireshark captures packets; Wireshark benefits and important features; the necessary prerequisites before capturing packets; and other packet analyzer tools for packet editing/sniffing/replaying and so on. We also provided a brief overview of mobile packet capturing.

The next chapter will be more specific to Wireshark and its tips and tricks. After that we will explore TCP troubleshooting, then plunge into SSL, and other application protocols such as DHCPv6, DHCP, DNS, and HTTP. We will also analyze Wi-Fi capturing and carry out some security analyses with the help of Wireshark and tcpdump.

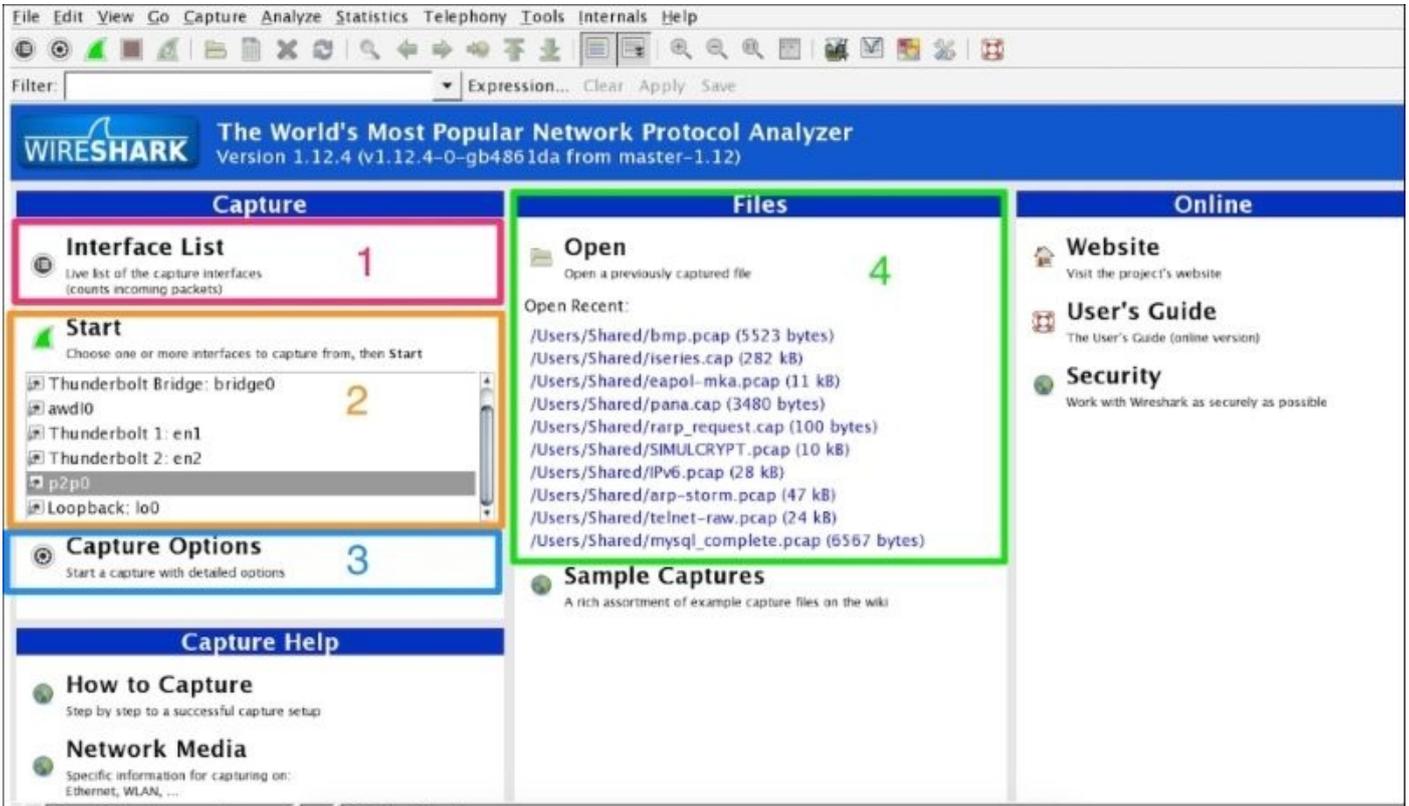
Chapter 2. Capturing Packets

In the previous chapter, we learned what packet analyzers are used for. In this chapter we will learn more about the Wireshark GUI features, and see how it helps in capturing and analyzing packets effectively, by covering the following topics:

- Capturing packets with Wireshark interface lists
- Capturing packets with Wireshark start options
- Capture options
- Wireshark filter examples
- Wireshark Packet List pane
- Wireshark Packet Details pane
- Wireshark features
- The tcpdump and snoop examples

Guide to capturing packets

Start Wireshark by clicking on the Wireshark icon or type `wireshark` in the command line. When Wireshark starts it launches the following screen and provides the following ways to capture packets:



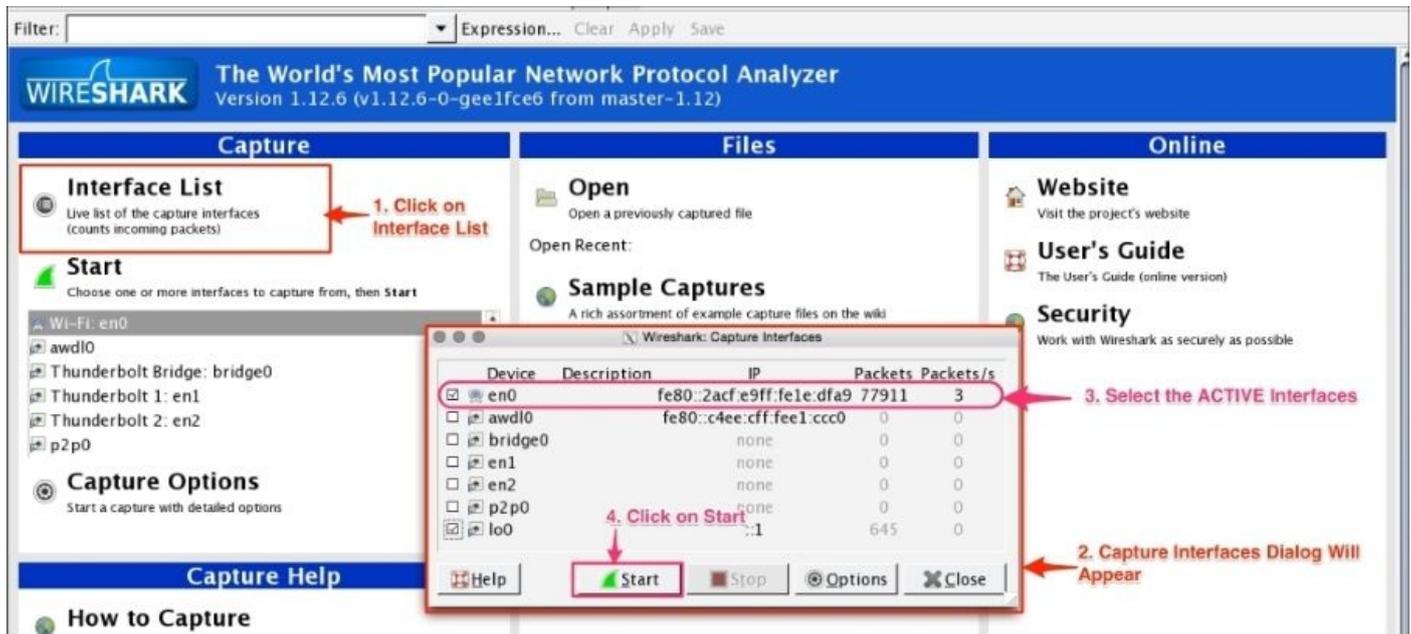
The following table explains the various options that we have on the Start screen:

Sr. no.	Wireshark capture options	What is this?
1	Interface List	Opens up a live list of capture interfaces, and counts the incoming/outgoing packets
2	Start	You can choose an interface from the list and start capturing packets
3	Capture Options	Provides various options for capturing and displaying packets
4	Open Recent	Wireshark displays recently used packets

We will cover each capturing option in detail one by one.

Capturing packets with Interface Lists

Click on **Interface List**; Wireshark will show a list of available network interfaces in the system and which one is active, by showing packets going in and out of the Interface, as shown in the following screenshot:

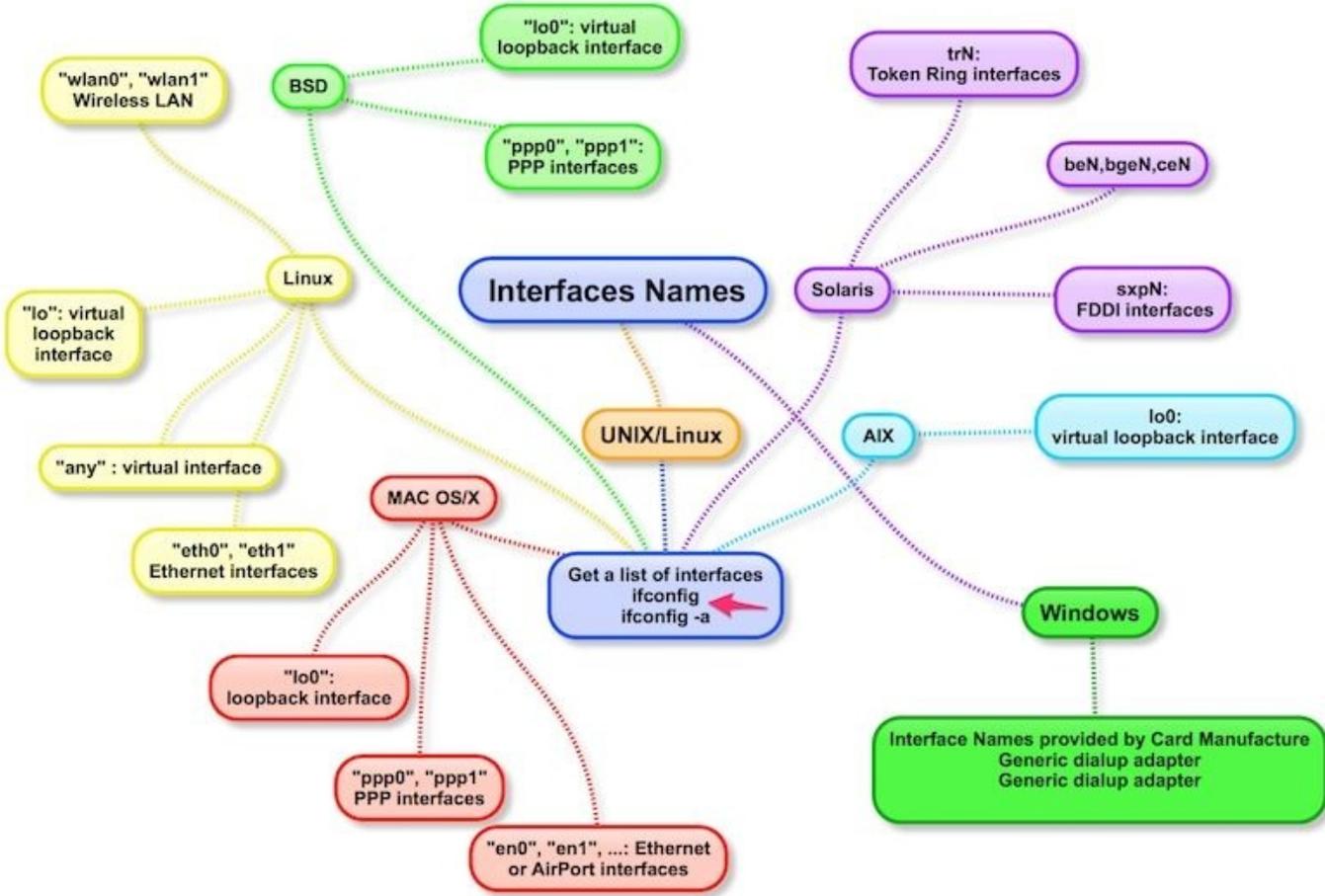


Choose the right (live) interfaces and click on the **Start** button to start capturing packets. If you want to capture packets on loopback (127.0.0.1), select the interface **lo0**.

Common interface names

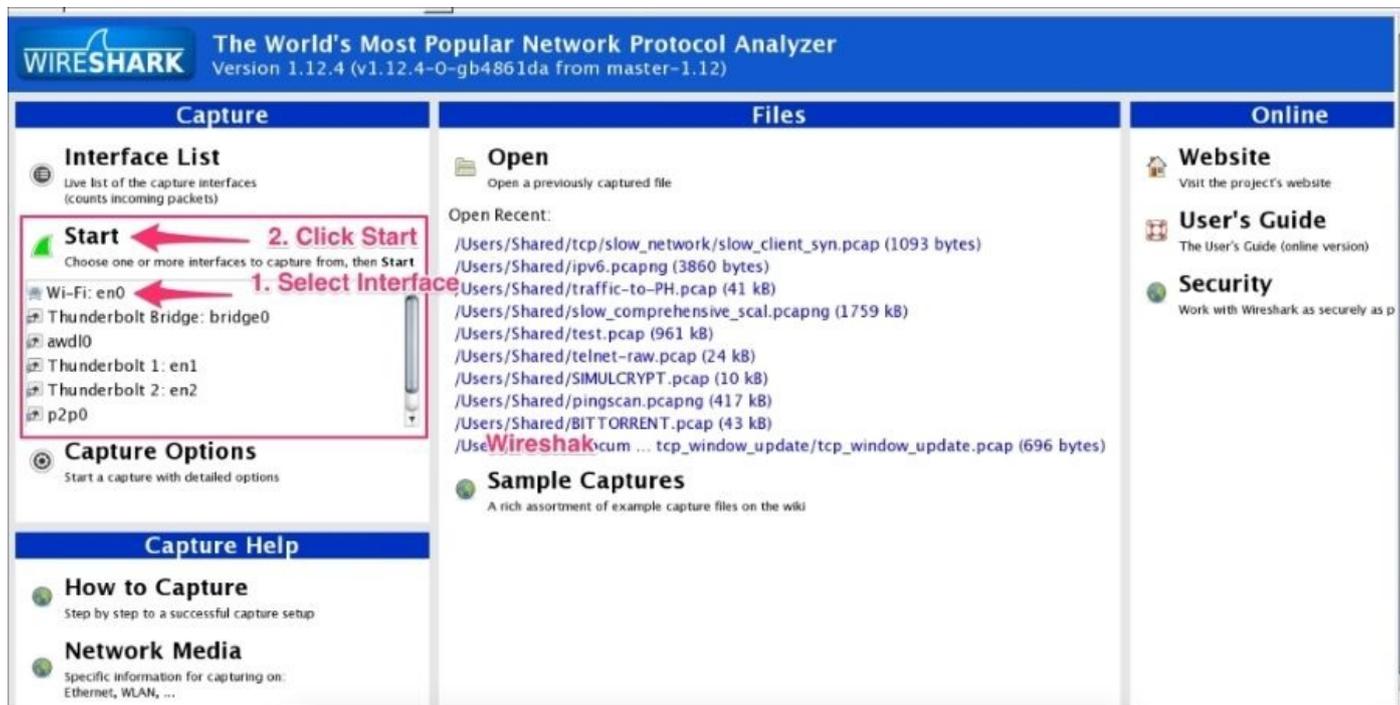
The interface name tells you the network type; by looking at the name of the interface the user should understand what network the capture setup is associated with—for example, eth0 stands for Ethernet. A few of them are shown in the following diagram:

common interface names



Capturing packets with Start options

In **Start** options, users can multiselect or select the interface displayed in the list and then click on Start. This doesn't give you the flexibility to see on which interface the packets are active. Users can configure the capture options by double clicking on the interface or by clicking on **Capture Options**:



Capturing packets with Capture Options

Wireshark provides the flexibility to configure packets that need to be captured with various capture options. To begin, try these basic settings:

1. Choose the live interface, where packets are going in and out.
2. Click on **Capture Options**, Wireshark will open the **Capture Options** dialog box.
3. Enable the promiscuous mode, which will allow the network interface to receive all packets.
4. Check the snaplength size. This option will tell you the size of data for each frame that should be captured by Wireshark; this is useful when capturing the header frame or to keep the packet size small.
5. **Name Resolution** tries to resolve the numerical address (for example, the MAC address, the IP address, and port) to its corresponding name, under the category where the following options are defined:
 - **Resolve MAC addresses:** This is used to convert the MAC address to a human-readable format; for example `28:cf:e9:1e:df:a9` will translate to `192.168.1.101`.
 - **Resolve network-layer names (IP name resolution):** This is used to convert the IP address to its corresponding hostname (for example, `216.58.220.46` will translate to `google.com`).
 - **Resolve transport-layer name (TCP/UDP port name resolution):** This is used to convert well-known ports to human-readable format (for example, `443` will translate to `https`).
6. Use the external network name resolver to perform a reverse DNS lookup for each unique IP address (for example `216.58.196.14` will translate to `ns4.google.com`) also referred to as reverse DNS lookup.

Users can also choose these options by selecting the Wireshark **View** menu and applying the following settings:

- **View | Name Resolution | Use External Network Name Resolver**
- **View | Name Resolution | Enable for MAC Layer**
- **View | Name Resolution | Enable for Transport Layer**
- **View | Name Resolution | Enable for Network Layer**

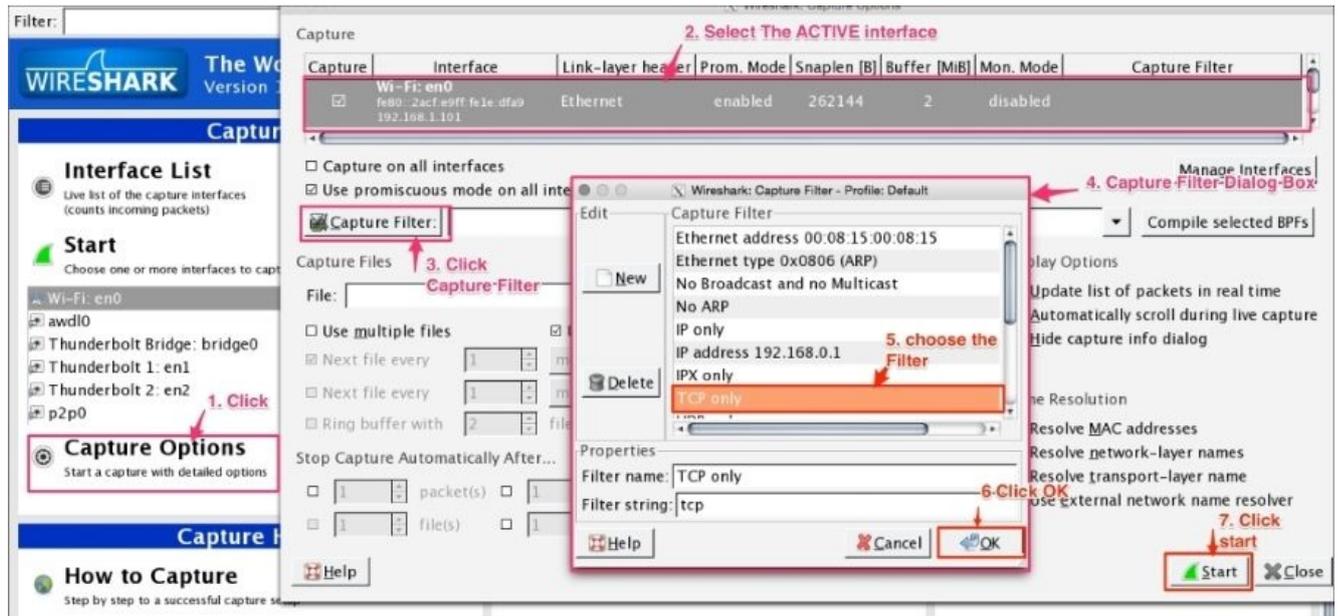
The drawbacks of name resolution are as follows:

- Once you have enabled these name resolution options, Wireshark will generate extra packets to resolve the name from the name server if the traffic is huge and there are high numbers of unique IP addresses. With these settings Wireshark will become very slow.
- Wireshark caches the resolved DNS name, so if the name server information changes, manual reload is required.

The capture filter options

Wireshark provides a range of capture filter options, use these options to decide which packets will save to the disk. These options are useful when capturing packets over a longer period of time. Wireshark uses the **Berkeley Packet Filter (BPF)** syntax for this purpose, for example `tcp src port 22`. This option also saves disk space. For example, to capture only TCP packets, follow the given steps:

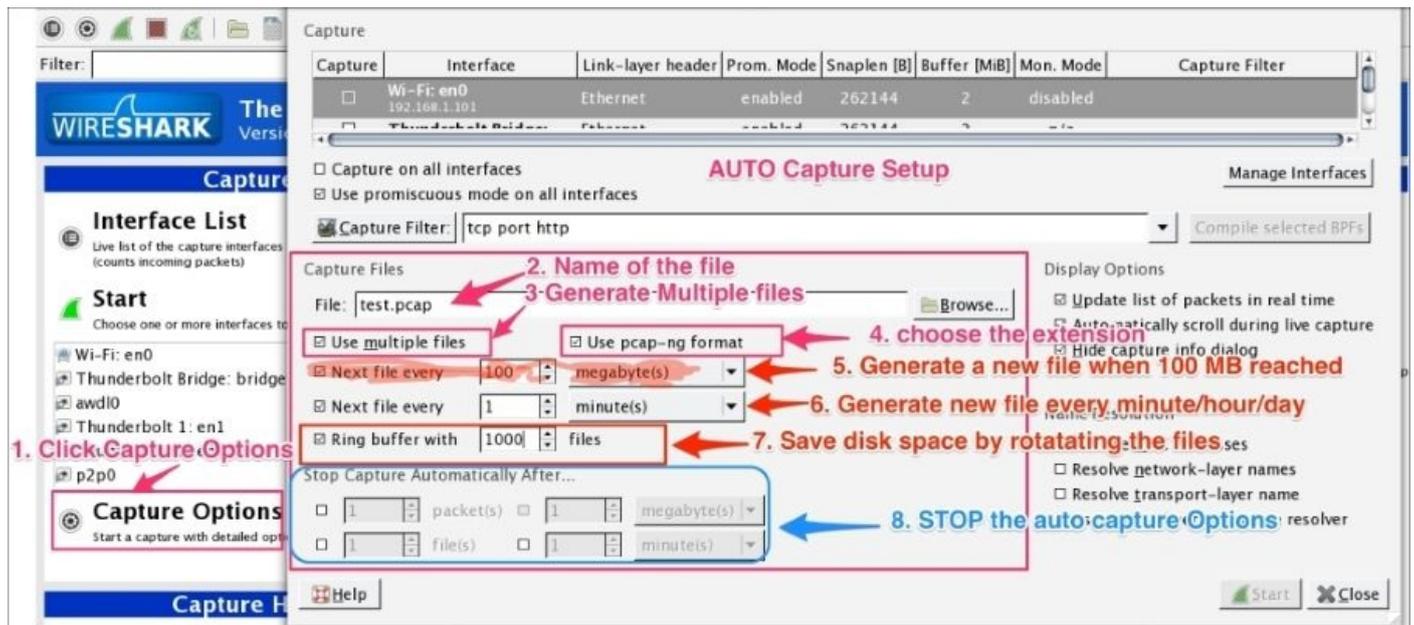
1. Click on **Capture Options**. The dialog box will open as shown in the screenshot.
2. Select the *active* interface and set the promiscuous mode setting to enabled or disabled.
3. Click on **Capture Filter**. Once the dialog box appears, choose the **TCP only** filter and click on **OK**.



4. Click on the **Start** button to start capturing just the TCP packets.

Auto-capturing a file periodically

Users can fine-tune Wireshark to auto-capture files periodically. To do this, click on **Capture Options | Capture Files**, as shown in the following screenshot:



Wireshark will generate files such as `test_00001_20150623001728.pcap` and `test_00002_20150623001818.pcap`.

The formats of the multiple generated files are as follows:

- test: This is the filename
- 00001: This is the file number
- 20150623001728: This is the date/time stamp
- pcap: This is the file extension

Troubleshooting

If a packet doesn't appear in the Wireshark main window, perform the following actions:

- Check the right network interface; make sure there is live traffic
- Try turning off/on promiscuous mode

If no interface appears on which captures can be performed, do the following:

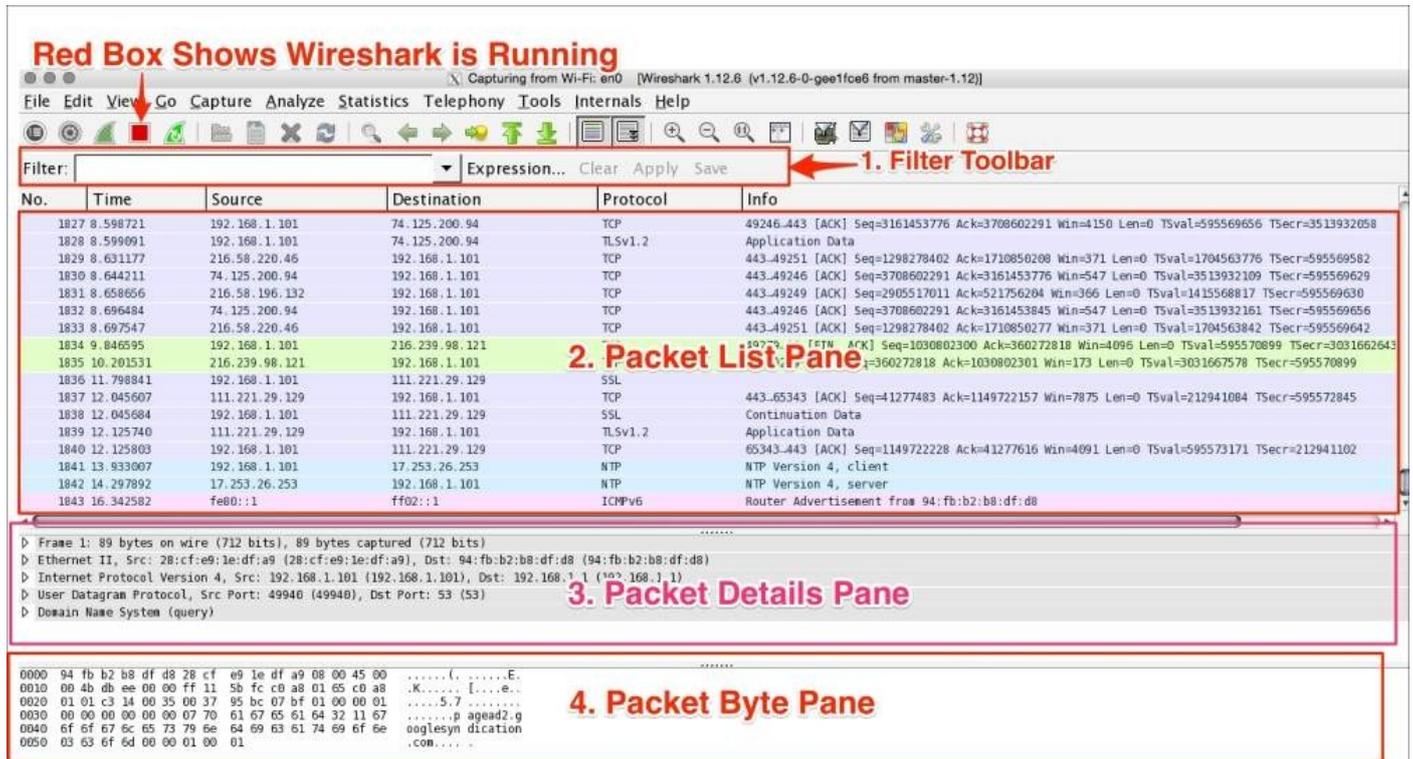
- Check if Wireshark has sufficient rights to use a network card to capture data
- Verify capture privileges from <http://wiki.wireshark.org/CaptureSetup/CapturePrivileges>

Note

You can also use the Wireshark community at <https://ask.wireshark.org/> if queries aren't resolved.

Wireshark user interface

The Wireshark main window appears when Wireshark starts capturing a packet, or when a .pcap file is open for offline viewing. It looks similar to the following screenshot:

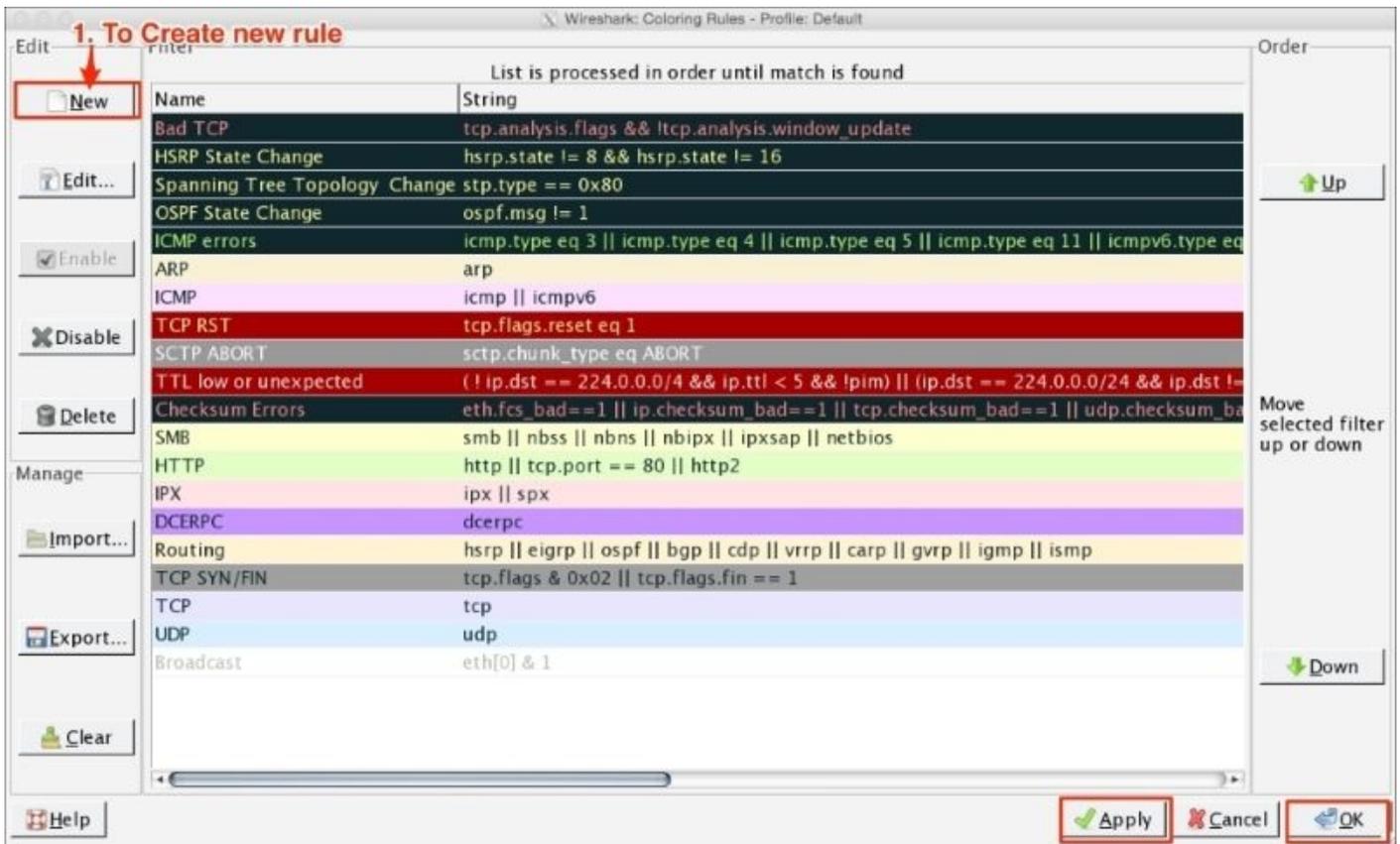


The Wireshark UI interface consists of different panes and provides various options to the user for customizing it. In this chapter, we will cover these panes in detail:

Item	What is it?
The red box	This shows that Wireshark is running and capturing a packet
1	This is the Filter toolbar, used for filtering packets based on the applied filter
2	This is the Packet List pane, which displays all captured packets
3	This is the Packet Details pane, which shows the selected packet in a verbose form
4	This is the Packet Byte pane, which shows the selected packet in a hex dump format

First, just observe pane 2 in the screen; the displayed packets appear with different colors. This is one of Wireshark's best features; it colors packets according to the set filter and helps you visualize the packet you are looking for.

To manage (view, edit, or create) a coloring rule, go to **View | Coloring Rules**. Wireshark will display the **Coloring Rules** dialog box, as shown in the screenshot:



Users can create a new rule by clicking on the **New** button, choosing the filter name and filter string, and then applying a foreground and background color to it, to customize the packet with a specific color.

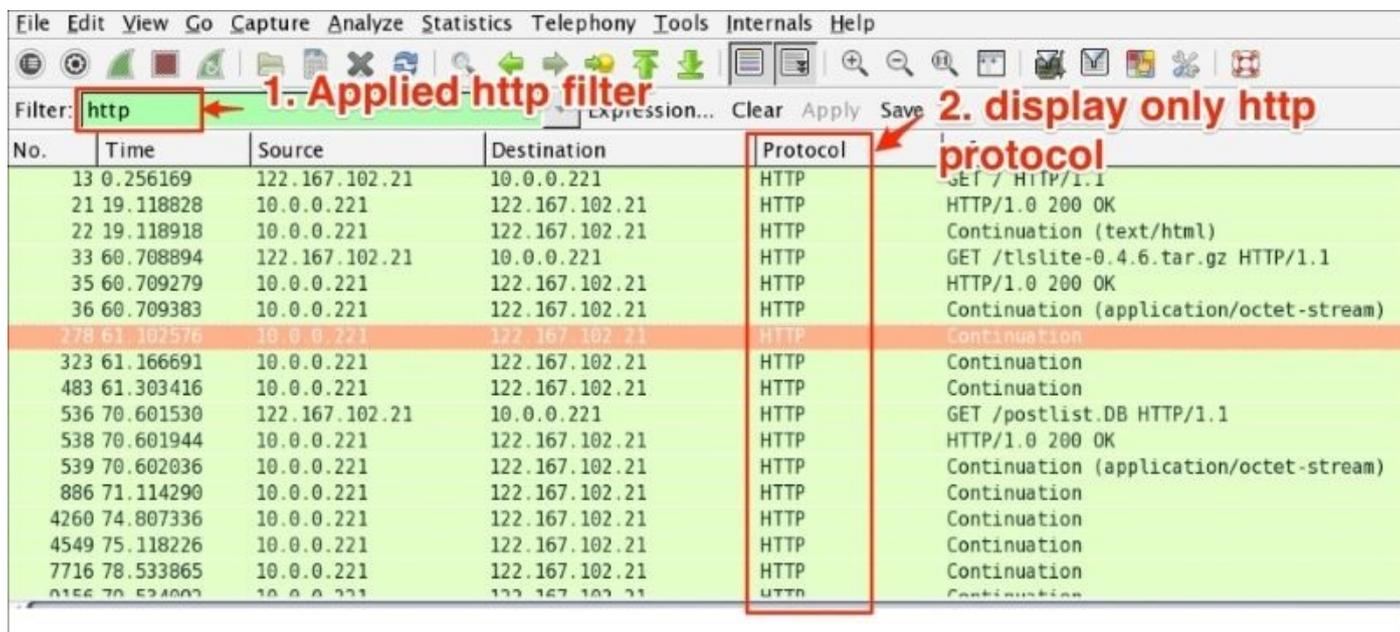
The Filter toolbar

The Wireshark display filter displays packets with its available coloring options. Wireshark display filters are used to change the view of a capture file by providing the full dissection of all packets, which helps analyzing a network tracefile efficiently. For example, if a user is interested in only HTTP packets, the user can set the display filter to `http`, as shown in the next screenshot.

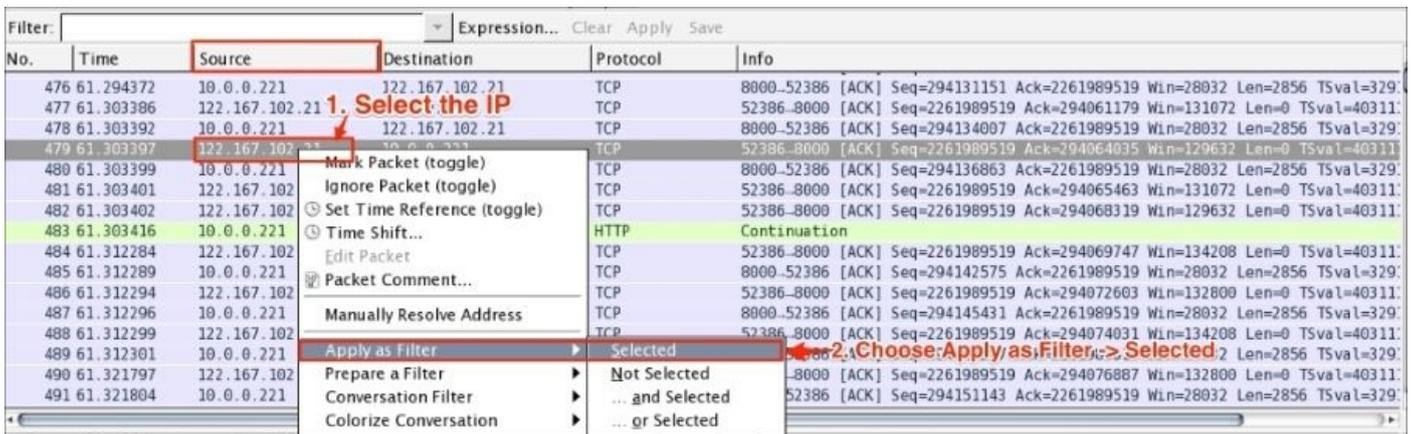
The steps to apply display filters are as follows:

1. Open the `http_01.pcap` file.
2. Type the `http` protocol in the filter area and click on **Apply**.

Once the filter is applied, the Packet List pane will display only HTTP protocol-related packets:



Wireshark display filter can be applied or prepared from the column displayed in the Packet List pane by selecting the column, then right-clicking and going to **Apply as Filter | Selected** (as shown in the following screenshot) to create the filter from the source IP address `122.167.102.21`:

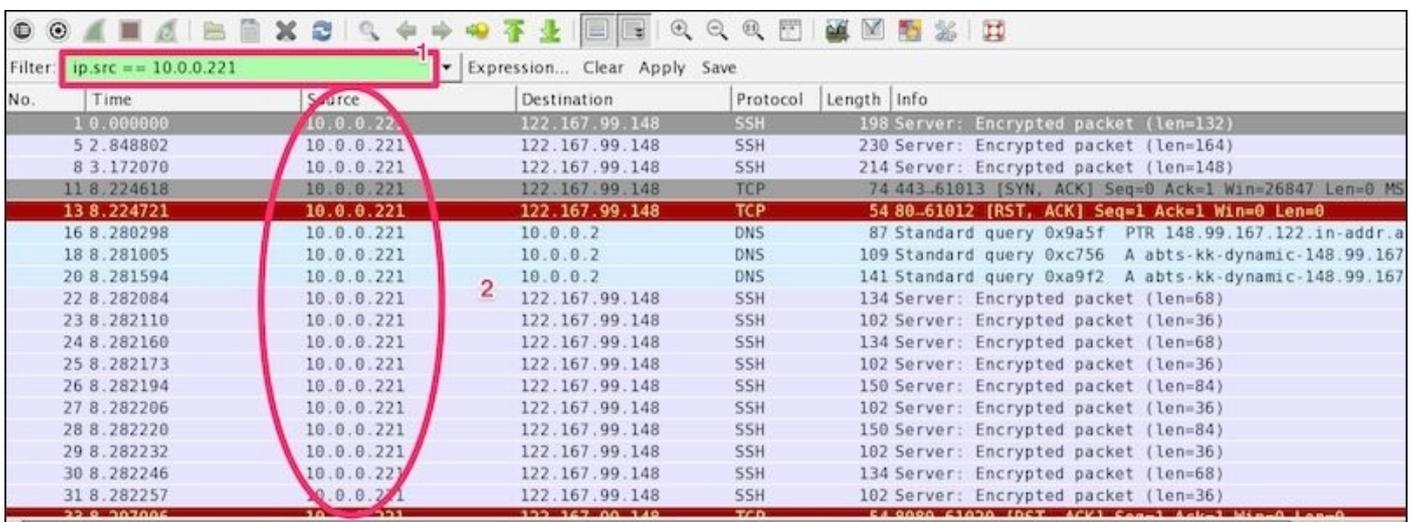


Wireshark provides the flexibility to apply filters from the Details pane; the steps remain the same.

Wireshark also provides the option to clear the filter. To do this click on **Clear** (available in the **Filter** toolbar) to display the entire captured packet.

Filtering techniques

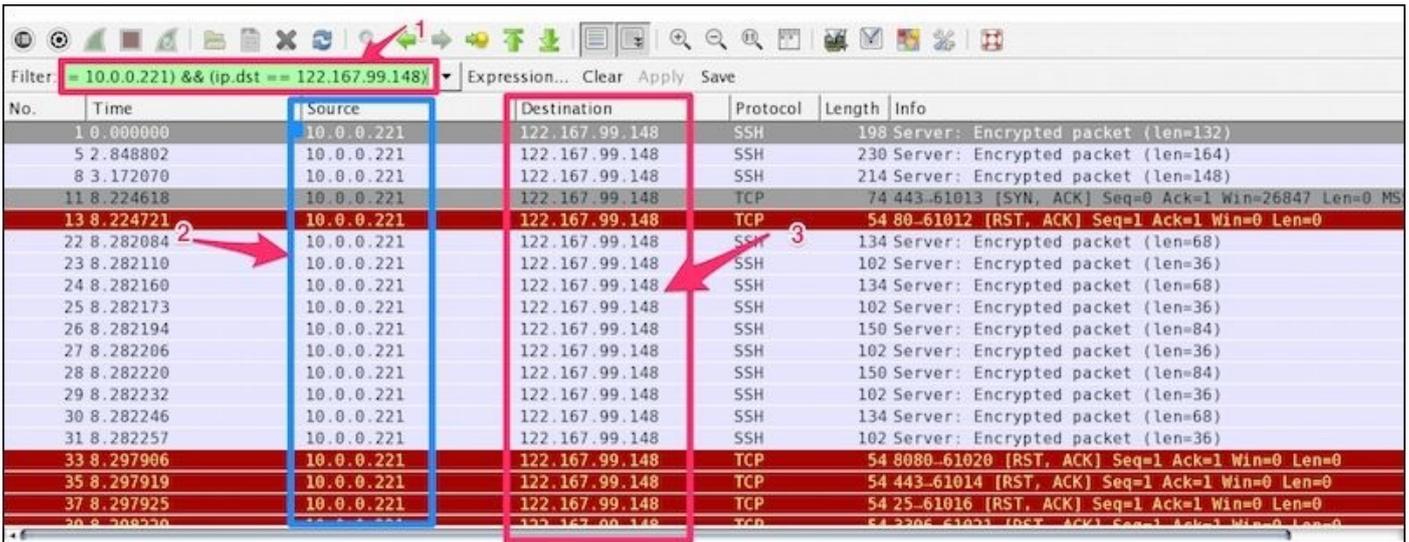
Capturing and displaying packets properly will help you with packet captures. For example, to track a packet exchanged between two hosts: HOSTA (10.0.0.221) and HOSTB (122.167.99.148), open the SampleCapture01.pcap file and apply the filter `ip.src == 10.0.0.221` as shown:



Let's see what the highlighted sections depict:

Item	Description
1	Apply filter <code>ip.src == 10.0.0.221</code> .
2	The Packet List pane displays the traffic from source to destination. The source shows the constant IP address 10.0.0.221. There is no evidence as to which packet is sent from host 122.167.99.148 to host 10.0.0.221.

Now modify the filter (`ip.src == 10.0.0.221`) && (`ip.dst == 122.167.99.148`) to (`ip.src == 10.0.0.221`) or (`ip.dst == 122.167.99.148`). This will give the result shown in the following screenshot:

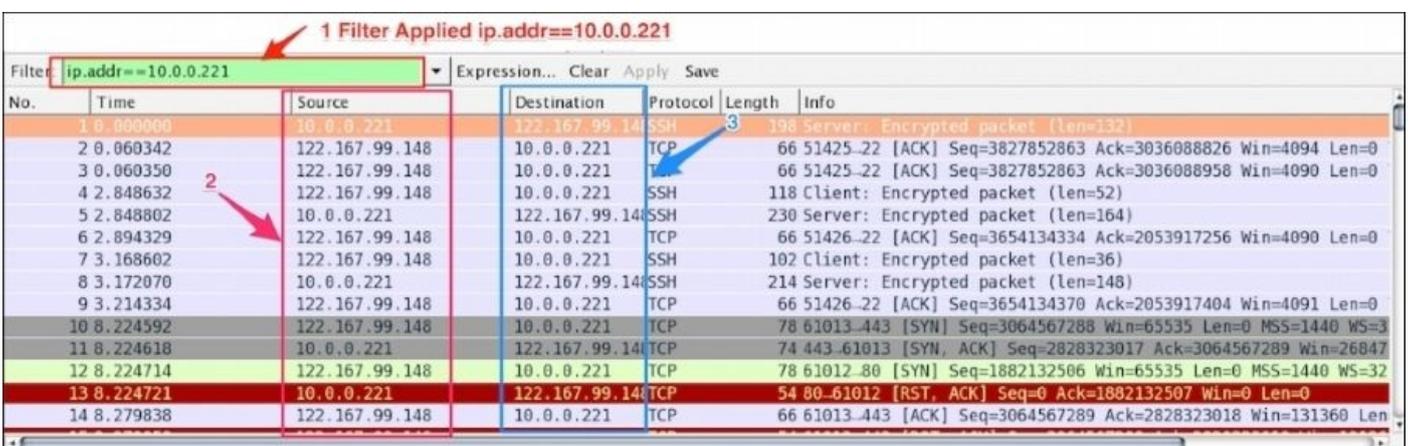


The highlighted sections in the preceding screenshot are explained as follows:

Item	Description
1	Applied filter (<code>ip.src == 10.0.0.221</code>) && (<code>ip.dst == 122.167.99.148</code>)
2	The source IP address (<code>10.0.0.221</code>) is not changed
3	The destination IP address (<code>122.167.99.148</code>) is not changed

Again the Packet List pane is not displaying the conversation between the two hosts.

Now modify the filter `ip.addr == 122.167.99.148`. The `ip.addr` field will match the IP header for both the source and destination address and display the conversation between the hosts. Remember to choose the destination IP address as shown:



Let's see what the highlighted sections depict:

Item	Description
1	Applied filter <code>ip.addr == 122.167.99.148</code>
2	The source IP is not constant; it shows the conversation between the two hosts
3	The destination IP is not constant; it shows the conversation between the two hosts

The same conversation is captured by choosing the destination MAC address using the display filter `eth.addr == 06:73:7a:4c:2f:85`.

Filter examples

Some common filter examples are as follows:

Filter/capture name	Filter value
Packet on a given port	<code>tcp.port == 443</code>
Packet on the source port	<code>tcp.srcport=2222</code>
SYN packet on port 443	<code>(tcp.port == 443) && (tcp.flags == 0x0010)</code>
The HTTP protocol	<code>http</code>
Based on the HTTP get method	<code>http.request.method == "GET"</code>
Using &&, tcp, and http	<code>tcp && http</code>
Checking the tcp window size	<code>tcp.window_size <2000</code>
No Arp used for normal traffic	<code>!arp</code>
The MAC address filter	<code>eth.dst == 06:43:7b:4c:4f:85</code>
Filter out TCP ACK	<code>tcp.flags.ack==0</code>
Check only RST and ACK packets	<code>(tcp.flags.ack == 1) && (tcp.flags.reset == 1)</code>
Filter all SNMP	<code>Snmp</code>
HTTP or DNS or SSL	<code>http dns ssl</code>

There is no need to memorize the filter; there is an easy way to apply it. The display filter Autocomplete feature lists all dissectors after the first period “.” that have been added to the display filter, as shown in the following screenshot:

Filter: tcp. ←

Expression... Clear Apply Save

No.	Destination	Protocol	Length	Info
tcp.ack				
tcp.ack.nonzero	122.167.99.148	SSH	198	Server: Encrypted packet (len=132)
tcp.analysis	10.0.0.221	TCP	66	51425-22 [ACK] Seq=3827852863 Ack=3036088826 Win=4094 Len=0 TS
tcp.analysis.ack_lost_segment	10.0.0.221	TCP	66	51425-22 [ACK] Seq=3827852863 Ack=3036088958 Win=4090 Len=0 TS
tcp.analysis.ack_rtt	10.0.0.221	SSH	118	Client: Encrypted packet (len=52)
tcp.analysis.acks_frame	122.167.99.148	SSH	230	Server: Encrypted packet (len=164)
tcp.analysis.bytes_in_flight	10.0.0.221	TCP	66	51426-22 [ACK] Seq=3654134334 Ack=2053917256 Win=4090 Len=0 TS
tcp.analysis.duplicate_ack	10.0.0.221	SSH	102	Client: Encrypted packet (len=36)
3.214334	122.167.99.148	SSH	214	Server: Encrypted packet (len=148)
9.3.214334	122.167.99.148	TCP	66	51426-22 [ACK] Seq=3654134370 Ack=2053917404 Win=4091 Len=0 TS
10.8.224592	122.167.99.148	TCP	78	61013-443 [SYN] Seq=3064567288 Win=65535 Len=0 MSS=1440 WS=32
11.8.224618	10.0.0.221	TCP	74	443-61013 [SYN, ACK] Seq=2828323017 Ack=3064567289 Win=26847
12.8.224714	122.167.99.148	TCP	78	61012-80 [SYN] Seq=1882132506 Win=65535 Len=0 MSS=1440 WS=32
13.8.224721	10.0.0.221	TCP	54	80-61012 [RST, ACK] Seq=0 Ack=1882132507 Win=0 Len=0
14.8.224728	122.167.99.148	TCP	66	51426-22 [ACK] Seq=3064567288 Ack=3036088826 Win=4094 Len=0 TS

tcp dissectors

Note

It's worth checking the following links for a complete display filter reference:

- Check out the TCP display filter reference:
<https://www.wireshark.org/docs/dfref/t/tcp.html>
- Check out this alternative protocol display filter reference:
<https://www.wireshark.org/docs/dfref/>

The Packet List pane

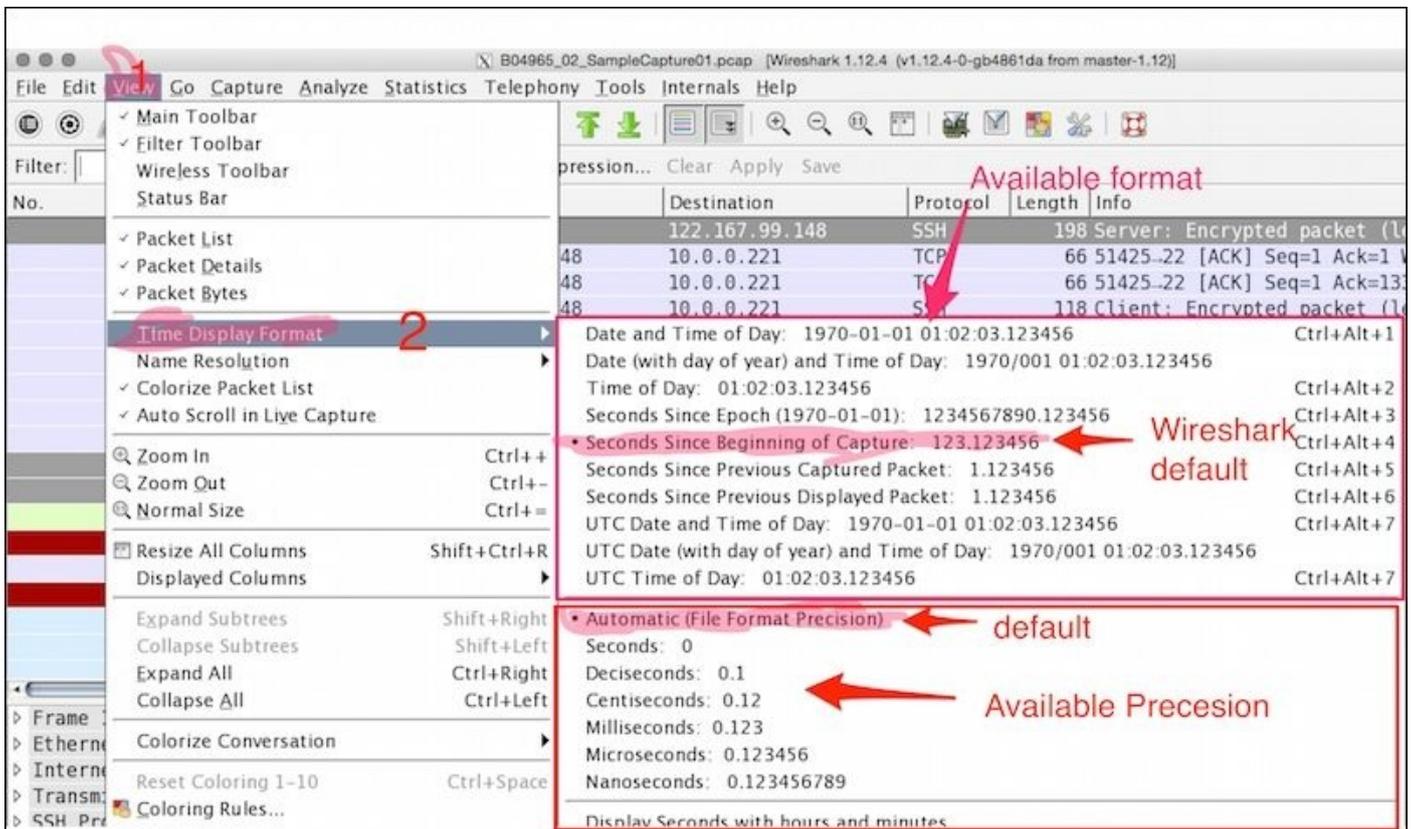
The Packet List pane displays packets from the .pcap (or accepted Wireshark extensions) file or from live capture, as shown:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.800000	10.0.0.221	122.167.99.148	SSH	198	Server: Encrypted packet (len=132)
2	0.060342	122.167.99.148	10.0.0.221	TCP	66	51425->22 [ACK] Seq=1 Ack=1 Win=4094 Len=0 TSval=704438813 TSecr=1742584
3	0.060350	122.167.99.148	10.0.0.221	TCP	66	51425->22 [ACK] Seq=1 Ack=133 Win=4090 Len=0 TSval=704438813 TSecr=1742584
4	2.848632	122.167.99.148	10.0.0.221	SSH	118	Client: Encrypted packet (len=52)
5	2.848802	10.0.0.221	122.167.99.148	SSH	230	Server: Encrypted packet (len=164)
6	2.894329	122.167.99.148	10.0.0.221	TCP	66	51426->22 [ACK] Seq=53 Ack=165 Win=4090 Len=0 TSval=704441647 TSecr=1742584
7	2.895502	122.167.99.148	10.0.0.221	TCP	66	51426->22 [ACK] Seq=53 Ack=165 Win=4090 Len=0 TSval=704441647 TSecr=1742584

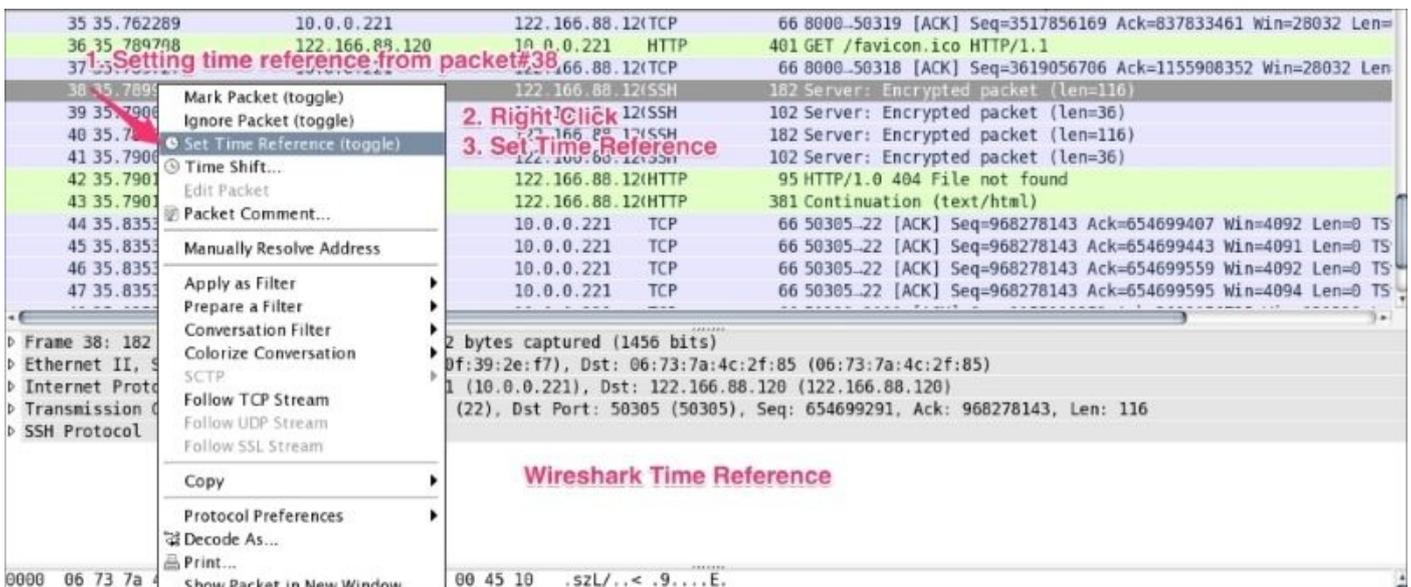
Let's discuss the fields shown:

Item	What is it?
	Shows different packets; each row corresponds to a different packet called a frame
1. No.	Number of packets in the current live/offline capture
2. Time	Shows time-stamped information when the packet was captured The Automatic setting for libpcap files is microseconds; all packets will be captured with the time in microseconds, as shown in the next screenshot
3. Source	The IP address of the source from where the packet originates
4. Destination	The IP address of the destination where the packet ends
5. Protocol	Wireshark will display information about the packet protocol based on the standard port
6. Length	The packet length in bytes
7. Info	Shows a high-level summary of the packet and the nature of the packet

To change the time-stamped information of the packet go to **View | Time Display Format** to view the available presentation formats, as shown:



The Wireshark **Set Time Reference** feature gives you the ability to view the time reference from the selected packet. Open the capture file `http.pcap` and set the time reference from packet 38. To do this, select packet 38, right-click, and select **Set Time Reference (toggle)**, as shown in the following screenshot:



After ***REF*** is set, it becomes the starting point for all subsequent packet time calculations, as shown in the following screenshot:

35	35.762289	10.0.0.221	122.166.88.12	TCP	66 8000_50319 [ACK] Seq=3517856169 Ack=837833461 Win=28032 Len=
36	35.789708	10.0.0.221	122.166.88.12	HTTP	401 GET /favicon.ico HTTP/1.1
37	35.789727	10.0.0.221	122.166.88.12	TCP	66 8000_50318 [ACK] Seq=3619056706 Ack=1155908352 Win=28032 Len=
38	*REF*	10.0.0.221	122.166.88.12	SSH	102 Server: Encrypted packet (len=116)
39	0.000028	10.0.0.221	122.166.88.12	SSH	102 Server: Encrypted packet (len=36)
40	0.000081	10.0.0.221	122.166.88.12	SSH	182 Server: Encrypted packet (len=116)
41	0.000099	10.0.0.221	122.166.88.12	SSH	102 Server: Encrypted packet (len=36)
42	0.000123	10.0.0.221	122.166.88.12	HTTP	95 HTTP/1.0 404 File not found
43	0.000186	10.0.0.221	122.166.88.12	HTTP	381 Continuation (text/html)
44	0.045313	10.0.0.221	10.0.0.221	TCP	66 50305_22 [ACK] Seq=968278143 Ack=654699407 Win=4092 Len=0 TS
45	0.045320	10.0.0.221	10.0.0.221	TCP	66 50305_22 [ACK] Seq=968278143 Ack=654699443 Win=4091 Len=0 TS
46	0.045321	122.166.88.120	10.0.0.221	TCP	66 50305_22 [ACK] Seq=968278143 Ack=654699559 Win=4092 Len=0 TS
47	0.045323	122.166.88.120	10.0.0.221	TCP	66 50305_22 [ACK] Seq=968278143 Ack=654699595 Win=4094 Len=0 TS

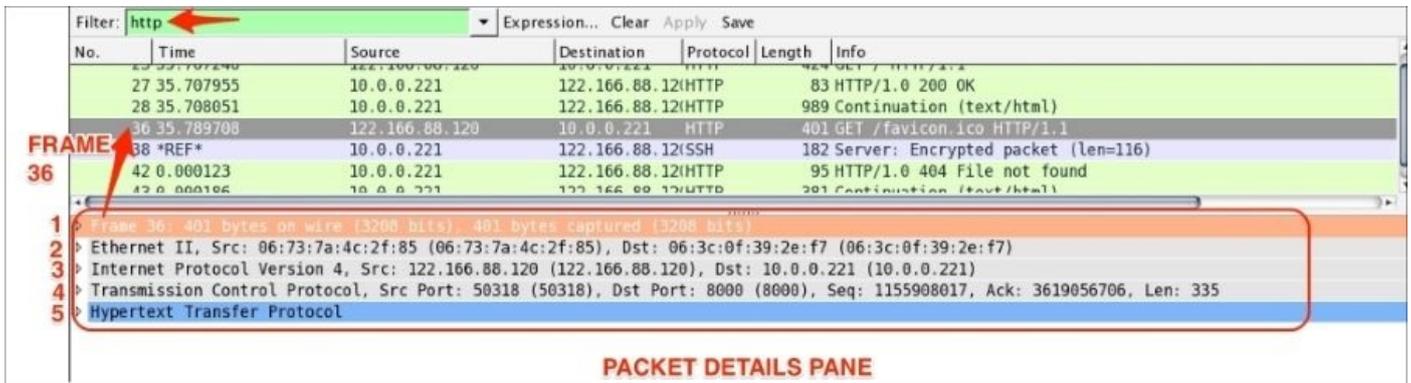
1. Before Time reference

REF SET

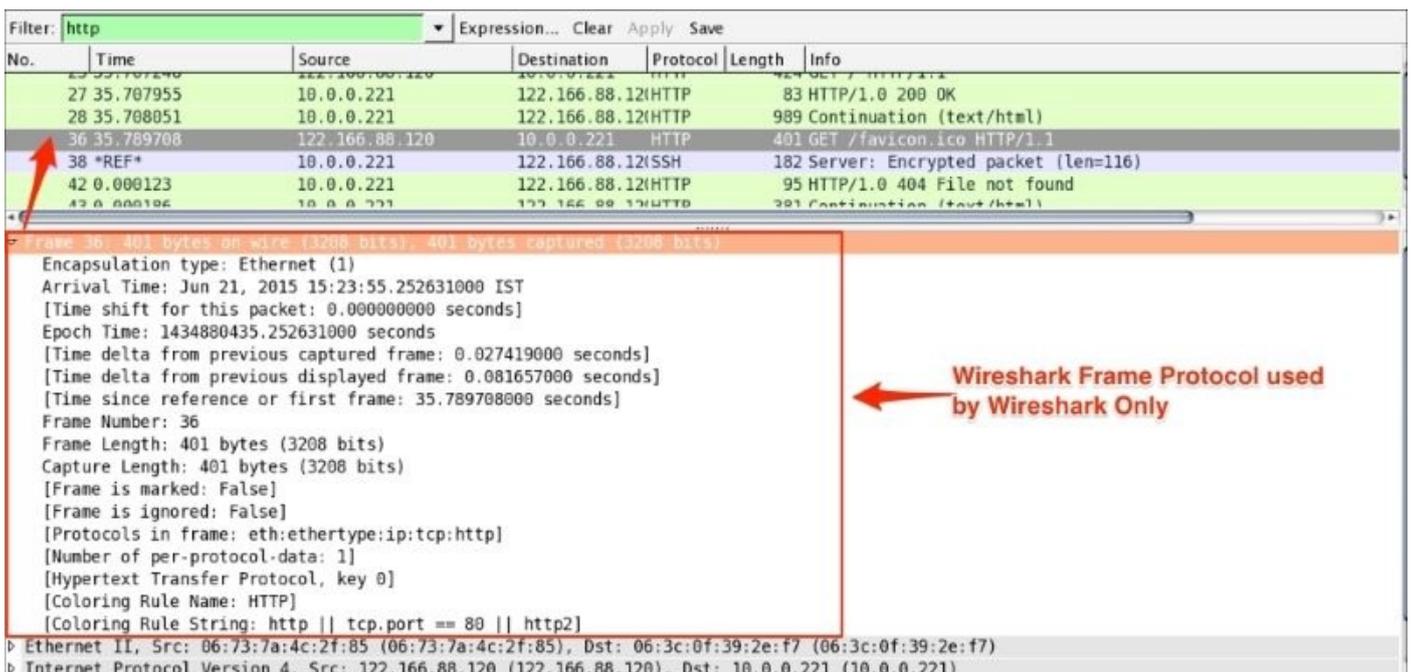
2. After Time reference
time adjusted based on REF

The Packet Details pane

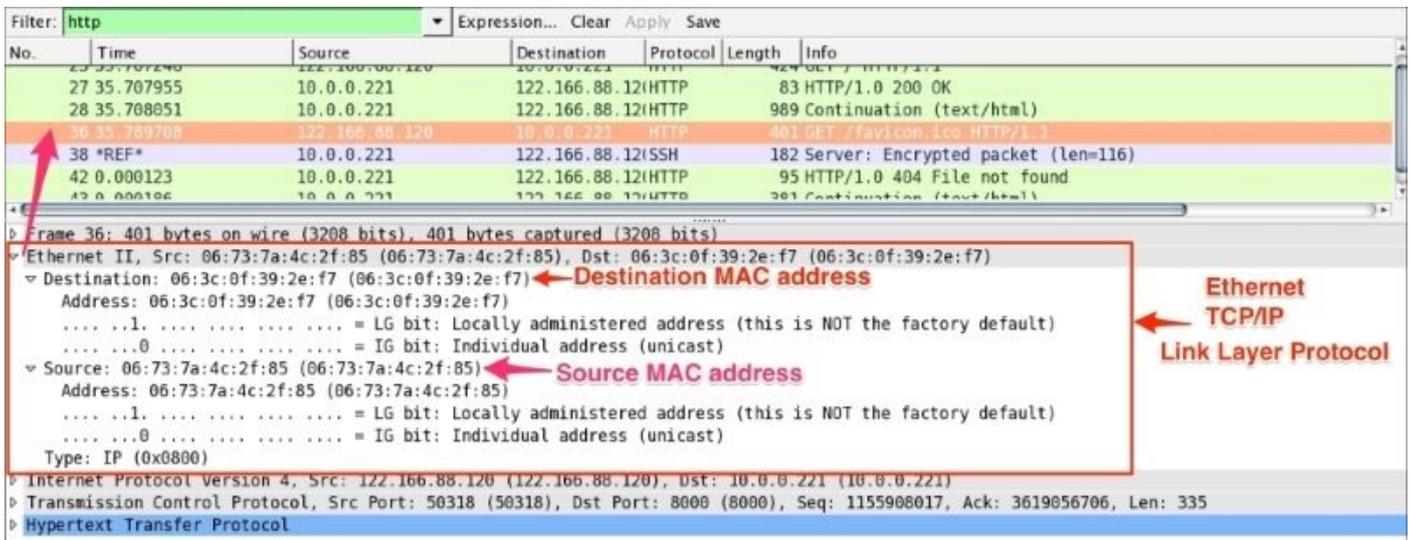
The Packet Details pane will show the currently selected packet in a more detailed form. In the following screenshot, an HTTP packet is selected and its details are shown in the information labeled with numbers 1 to 5. Let's see what these are:



The frame protocol is only used by Wireshark. All the TCP/IP protocols sits on top of this. The frame shows at what time the packet was captured, as shown in the following screenshot:



Ethernet is the link layer protocol in the TCP/IP stack. It sends network packets from the sending host to one (Unicast) or more (Multicast/Broadcast) receiving hosts, as shown:



Useful filters in Ethernet are:

- `eth.dst == 06:3c:0f:39:2e:f7`: This shows packets sent to this MAC address only
- `eth.dst==ff:ff:ff:ff:ff:ff`: This shows broadcast traffic only

The packet structure of Ethernet frames is described in the following table:

Preamble	Destination MAC address	Source MAC address	Type/length	User-data	Frame check sequence (FCS)
8	6	6	2 0800 for IPv4 86DD for IPv6 0806 for ARP	46-1500	4

The preamble (8 bytes) and FCS (4 bytes) are not part of the frame and Wireshark will not capture this field.

So the total Ethernet header is 14 bytes—6 bytes for the destination address, 6 bytes for the source address, and 2 bytes for the EtherType.

The Internet Protocol information relates to how the IP packet is delivered and whether it has used IPv4 or IPv6 to deliver the datagram packets.

Filter: **http** Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
27	35.707955	10.0.0.221	122.166.88.12	HTTP	83	HTTP/1.0 200 OK
28	35.708051	10.0.0.221	122.166.88.12	HTTP	989	Continuation (text/html)
36	35.789708	122.166.88.120	10.0.0.221	HTTP	401	GET /favicon.ico HTTP/1.1
38	*REF*	10.0.0.221	122.166.88.12	SSH	182	Server: Encrypted packet (len=116)
42	0.000123	10.0.0.221	122.166.88.12	HTTP	95	HTTP/1.0 404 File not found
43	0.000186	10.0.0.221	122.166.88.12	HTTP	381	Continuation (text/html)

Frame 36: 401 bytes on wire (3208 bits), 401 bytes captured (3208 bits)

Ethernet II, Src: 06:73:7a:4c:2f:85 (06:73:7a:4c:2f:85), Dst: 06:3c:0f:39:2e:f7 (06:3c:0f:39:2e:f7)

Internet Protocol Version 4, Src: 122.166.88.120 (122.166.88.120), Dst: 10.0.0.221 (10.0.0.221)

Version: 4
 Header Length: 20 bytes
 Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
 Total Length: 387
 Identification: 0xd119 (53529)
 Flags: 0x02 (Don't Fragment)
 Fragment offset: 0
 Time to live: 56
 Protocol: TCP (6)
 Header checksum: 0x9260 [validation disabled]
 Source: 122.166.88.120 (122.166.88.120)
 Destination: 10.0.0.221 (10.0.0.221)
 [Source GeoIP: Unknown]
 [Destination GeoIP: Unknown]

Transmission Control Protocol, Src Port: 50318 (50318), Dst Port: 8000 (8000), Seq: 1155908017, Ack: 3619056706, Len: 335

Hypertext Transfer Protocol

The IP Protocol

The preceding screenshots show that an IPv4 protocol is used to deliver the datagram packet. Useful display filters in the IP protocol are:

- `ip.src == 122.166.88.120/24` shows traffic from the subnet
- `ip.addr==122.166.88.120` shows traffic to or from the given host
- `Host 122.166.88.120` captures/filters traffic from the host

The TCP protocol packet contains all TCP-related protocol data. If the communication is over UDP, the TCP will be replaced by the UDP, as shown in the following screenshot. The SEQ/ACK analysis will be done by Wireshark based on the sequence number and expert info will be provided:

Filter: **http** Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
27	35.707955	10.0.0.221	122.166.88.12	HTTP	83	HTTP/1.0 200 OK
28	35.708051	10.0.0.221	122.166.88.12	HTTP	989	Continuation (text/html)
36	35.789708	122.166.88.120	10.0.0.221	HTTP	401	GET /favicon.ico HTTP/1.1
38	*REF*	10.0.0.221	122.166.88.12	SSH	182	Server: Encrypted packet (len=116)
42	0.000123	10.0.0.221	122.166.88.12	HTTP	95	HTTP/1.0 404 File not found
43	0.000186	10.0.0.221	122.166.88.12	HTTP	381	Continuation (text/html)

Frame 36: 401 bytes on wire (3208 bits), 401 bytes captured (3208 bits)

Ethernet II, Src: 06:73:7a:4c:2f:85 (06:73:7a:4c:2f:85), Dst: 06:3c:0f:39:2e:f7 (06:3c:0f:39:2e:f7)

Internet Protocol Version 4, Src: 122.166.88.120 (122.166.88.120), Dst: 10.0.0.221 (10.0.0.221)

Transmission Control Protocol, Src Port: 50318 (50318), Dst Port: 8000 (8000), Seq: 1155908017, Ack: 3619056706, Len: 335

Source Port: 50318 (50318)
 Destination Port: 8000 (8000)
 [Stream index: 8]
 [TCP Segment Len: 335]
 Sequence number: 1155908017
 [Next sequence number: 1155908352]
 Acknowledgment number: 3619056706
 Header Length: 32 bytes
 ... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
 Window size value: 4105
 [Calculated window size: 131360]
 [Window size scaling factor: 32]
 Checksum: 0x219a [validation disabled]
 Urgent pointer: 0
 Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
 [SEQ/ACK analysis]

The TCP Protocol in the Transport Layer

← Wireshark tcp.analysis

The <<APPLICATION-LAYER>> protocol is shown if the packet contains any application protocols. As shown in the following screenshot, the selected packet 36 has HTTP protocol data. Wireshark has the ability to decode the protocol based on the standard port and present this information in the Packet Details pane in a readable (RFC-defined) format.

Filter: ip.src == 122.166.88.120/24

No.	Time	Source	Destination	Protocol	Length	Info
32	35.762273	122.166.88.120	10.0.0.221	TCP	66	50305->22 [ACK] Seq=968278143 Ack=654699291 Win=4094 Len=0 TS
33	35.762275	122.166.88.120	10.0.0.221	TCP	66	50319->8000 [ACK] Seq=837833460 Ack=3517856169 Win=130432 Len
34	35.762280	122.166.88.120	10.0.0.221	TCP	66	50319->8000 [FIN, ACK] Seq=837833460 Ack=3517856169 Win=13107
36	35.789798	122.166.88.120	10.0.0.221	HTTP	401	GET /favicon.ico HTTP/1.1
38	*REF*	10.0.0.221	122.166.88.120	SSH	182	Server: Encrypted packet (len=116)
44	0.045313	122.166.88.120	10.0.0.221	TCP	66	50305->22 [ACK] Seq=968278143 Ack=654699407 Win=4092 Len=0 TS

Frame 36: 401 bytes on wire (3208 bits), 401 bytes captured (3208 bits)

Ethernet II, Src: 06:73:7a:4c:2f:85 (06:73:7a:4c:2f:85), Dst: 06:3c:0f:39:2e:f7 (06:3c:0f:39:2e:f7)

Internet Protocol Version 4, Src: 122.166.88.120 (122.166.88.120), Dst: 10.0.0.221 (10.0.0.221)

Transmission Control Protocol, Src Port: 50318 (50318), Dst Port: 8000 (8000), Seq: 1155908017, Ack: 3619056706, Len: 335

Hypertext Transfer Protocol

- GET /favicon.ico HTTP/1.1\r\n
- Host: 52.74.246.190:8000\r\n
- Connection: keep-alive\r\n
- User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.124 Safari/537.36\r\n
- Accept: */*\r\n
- Referer: http://52.74.246.190:8000/\r\n
- Accept-Encoding: gzip, deflate, sdch\r\n
- Accept-Language: en-US,en;q=0.8\r\n
- \r\n
- [Full request URI: http://52.74.246.190:8000/favicon.ico]
- [HTTP request 1/1]
- [Response in frame: 42]

**The Application Layer Protocol
HTTP protocol Information**

In the coming chapters we will discuss the application-related protocol in greater detail.

The Packet Bytes pane

The Packet Bytes pane displays the bytes contained in the frame, with the highlighted area being set to the node selected in the Packet Details pane.

Wireshark features

Wireshark is loaded with some awesome features. Let's go through a few, though there are more.

Decode-As

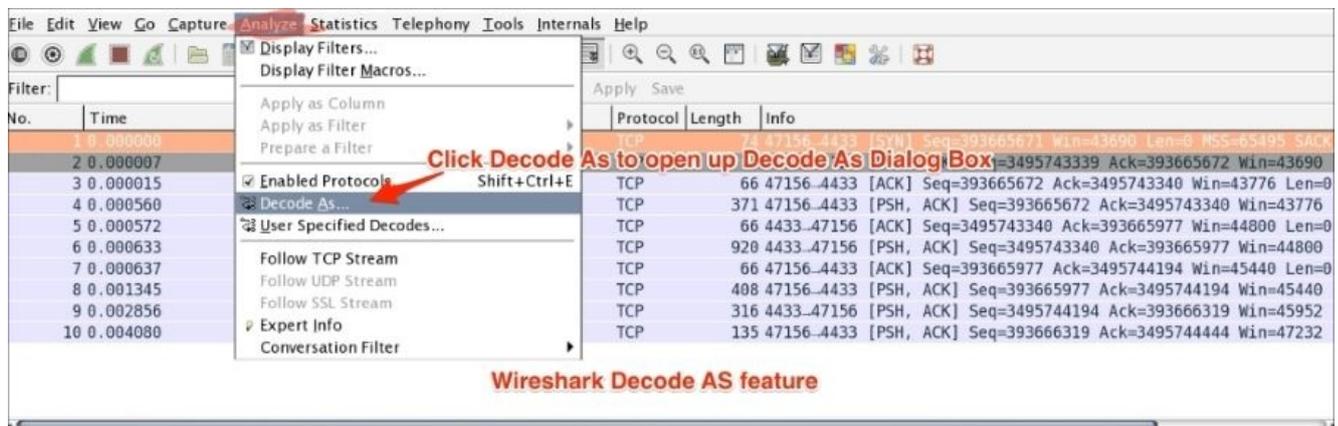
The Decode-As feature allows Wireshark to decode the packet based on the selected protocol. Usually Wireshark will automatically identify and decode incoming packets based on the standard port—for example, port 443 will be decoded as SSL. If the services are running on the non-standard port, for example SSL standard port is 443 and the service is running on 4433, in this case the Decode-As feature can be used to decode this communication using the SSL protocol preference.

Open the sample `https.pcap` file from. HTTPS traffic is captured when the file is opened in Wireshark. It doesn't show SSL-related data; instead it just shows all TCP communications:

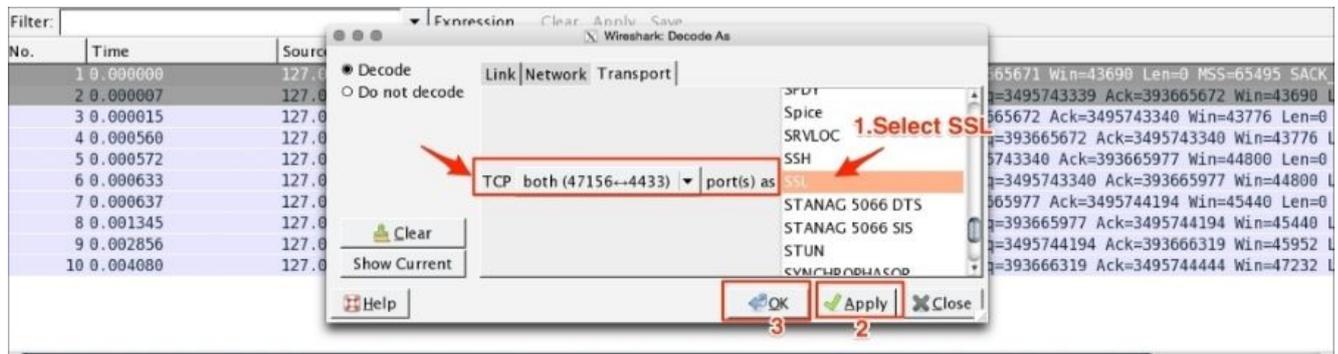
No.	Time	Source	Destination	Protocol	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	47156-4433 [SYN] Seq=393665671 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=322096422 TSecr=0
2	0.000007	127.0.0.1	127.0.0.1	TCP	4433-47156 [SYN, ACK] Seq=3495743339 Ack=393665672 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=322096422 TSecr=0
3	0.000015	127.0.0.1	127.0.0.1	TCP	47156-4433 [ACK] Seq=393665672 Ack=3495743340 Win=43776 Len=0 TSval=322096422 TSecr=0
4	0.000560	127.0.0.1	127.0.0.1	TCP	47156-4433 [PSH, ACK] Seq=393665672 Ack=3495743340 Win=43776 Len=305 TSval=32209643 TSecr=0
5	0.000572	127.0.0.1	127.0.0.1	TCP	4433-47156 [ACK] Seq=3495743340 Ack=393665977 Win=44800 Len=0 TSval=32209643 TSecr=0
6	0.000633	127.0.0.1	127.0.0.1	TCP	4433-47156 [PSH, ACK] Seq=3495743340 Ack=393665977 Win=44800 Len=854 TSval=32209643 TSecr=0
7	0.000637	127.0.0.1	127.0.0.1	TCP	47156-4433 [ACK] Seq=393665977 Ack=3495744194 Win=45440 Len=0 TSval=32209643 TSecr=0
8	0.001345	127.0.0.1	127.0.0.1	TCP	47156-4433 [PSH, ACK] Seq=393665977 Ack=3495744194 Win=45440 Len=342 TSval=32209643 TSecr=0
9	0.002856	127.0.0.1	127.0.0.1	TCP	4433-47156 [PSH, ACK] Seq=3495744194 Ack=393666319 Win=45952 Len=250 TSval=32209643 TSecr=0

To decode this traffic as SSL, follow these steps:

1. Click on **Analyze | Decode As**:



2. The **Decode As** popup will appear as shown in the following screenshot. Choose the protocol (**SSL** in this example) that is required for decoding the given traffic:



3. The SSL traffic protocol is shown in Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	74	47156-4433 [SYN] Seq=393665671 win=43690 Len=0 MSS=65495 SACK...
2	0.000007	127.0.0.1	127.0.0.1	TCP	74	4433-47156 [SYN, ACK] Seq=3495743339 Ack=393665672 Win=43690 L...
3	0.000015	127.0.0.1	127.0.0.1	TCP	66	47156-4433 [ACK] Seq=393665672 Ack=3495743340 Win=43776 Len=0
4	0.000560	127.0.0.1	127.0.0.1	TLSv1.2	371	Client Hello
5	0.000572	127.0.0.1	127.0.0.1	TCP	66	4433-47156 [ACK] Seq=3495743340 Ack=393665977 Win=44800 Len=0
6	0.000633	127.0.0.1	127.0.0.1	TLSv1.2	920	Server Hello, Certificate, Server Hello Done
7	0.000637	127.0.0.1	127.0.0.1	TCP	66	47156-4433 [ACK] Seq=393665977 Ack=3495744194 Win=45440 Len=0
8	0.001345	127.0.0.1	127.0.0.1	TLSv1.2	408	Client Key Exchange, Change Cipher Spec, Encrypted Handshake M...
9	0.002856	127.0.0.1	127.0.0.1	TLSv1.2	316	New Session Ticket, Change Cipher Spec, Encrypted Handshake Me...
10	0.004080	127.0.0.1	127.0.0.1	TLSv1.2	135	Application Data

Note

SSL decoding doesn't mean it has decrypted the SSL data.

Protocol preferences

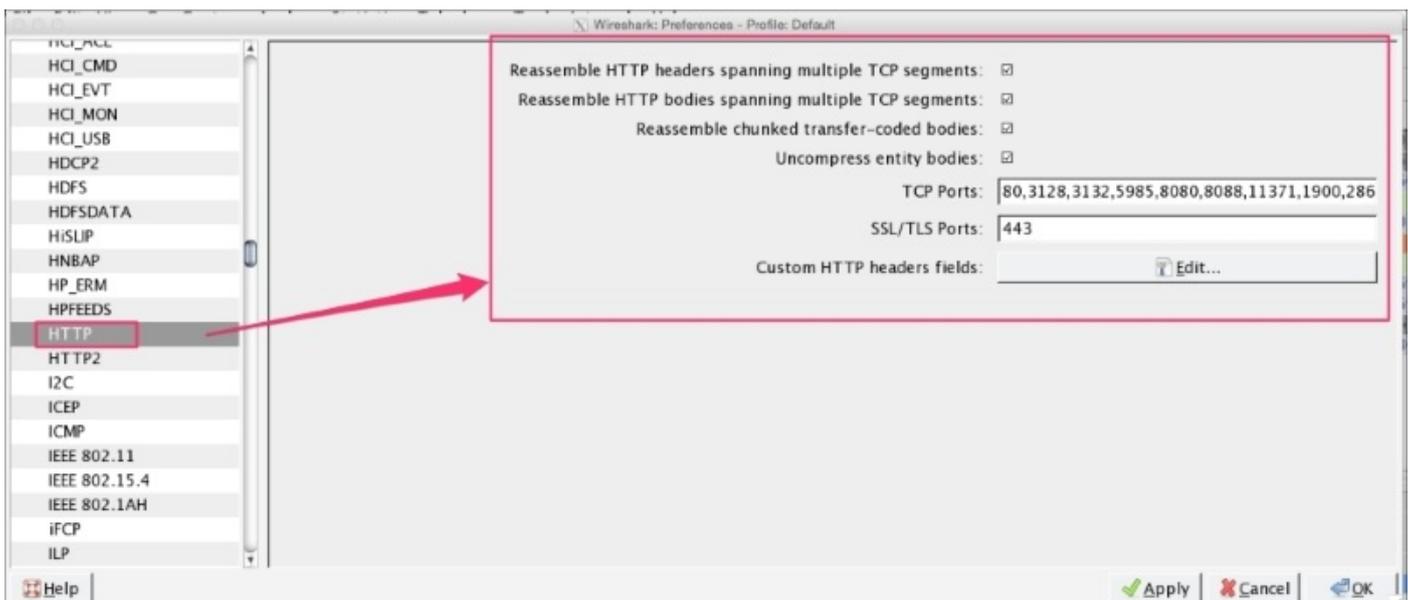
The protocol preference feature provides the flexibility for you to customize how the Wireshark display is processed, and how packets are analyzed. You can set protocol preferences by one of the following methods:

- Go to **Edit | Preferences | Protocols** to adjust the settings
- A simple way is to right-click on a protocol in the Packet Details pane and select **Protocol Preferences**

Wireshark supports a large set of protocols and it's preferences, for example HTTP protocol preferences and their meanings as defined in the following table:

HTTP protocol preferences	What does this mean?
Reassemble HTTP headers spanning multiple TCP segments	HTTP dissector will reassemble the HTTP header if it has been transmitted over more than one TCP segment
Reassemble HTTP bodies spanning multiple TCP segments	HTTP dissector will reassemble the HTTP body if it has been transmitted over more than one TCP segment
Reassemble chunked transfer-coded bodies	Reassemble all chunks across the segments and add them to the payload
Decompress entity bodies	Used for the visualization of compressed data (.gzip or encoded)
SSL/TLS ports	Add/remove SSL/TLS ports (default is 443)
Custom HTTP header fields	Define new header fields

The following screenshot shows HTTP protocol preferences in Wireshark:



Tip

Refer to the example of finding the top HTTP response time in [Chapter 05](#), *Analyze the DHCP, DHCPv6, DNS, HTTP Protocols* when using protocol preferences.

The IO graph

Use the IO graph to check client and server interaction data for a meaningful analysis. The Wireshark IO graph measures throughput (the rate is packet-per-tick), where each tick is one second. In this example we will see how to make use of the IO graph. Open the file `http_01.pcap` in Wireshark and follow the given steps:

1. Click on **Statistics | IO graph**.
2. The **IO graph** dialog box will appear.
3. In the **IO graph** dialog box try to find the spike and click on it.
4. When you click on the graph (the high area), Wireshark will automatically show the corresponding packet in the Packet List pane.

Note

In the given example there are lots of duplicate ACKs.

5. Go back to the **IO graph** dialog box.
6. Choose **Graph2** and enter `tcp.analysis.duplicate_ack`.
7. Click on **Graph2** to apply the filter.
8. The **IO graph** dialog will show the throughput of the duplicate ACK.

There are a lot of use cases for IO graphs. Some of them are as follows:

- Use IO graphs to analyze traffic patterns, for example how the traffic is distributed by plotting graphs on protocols for example `tcp`, `http`, `udp`, `ntp`, and `ldap`.
- IO graphs come in handy when performing security analysis. More examples of IO graphs are available in [Chapter 07](#), *Network Security Analysis*.

The following screenshots show the results of the preceding steps:

01 Generate IO Graph Statistics->IO Graph

02 - Locate the High select the graph

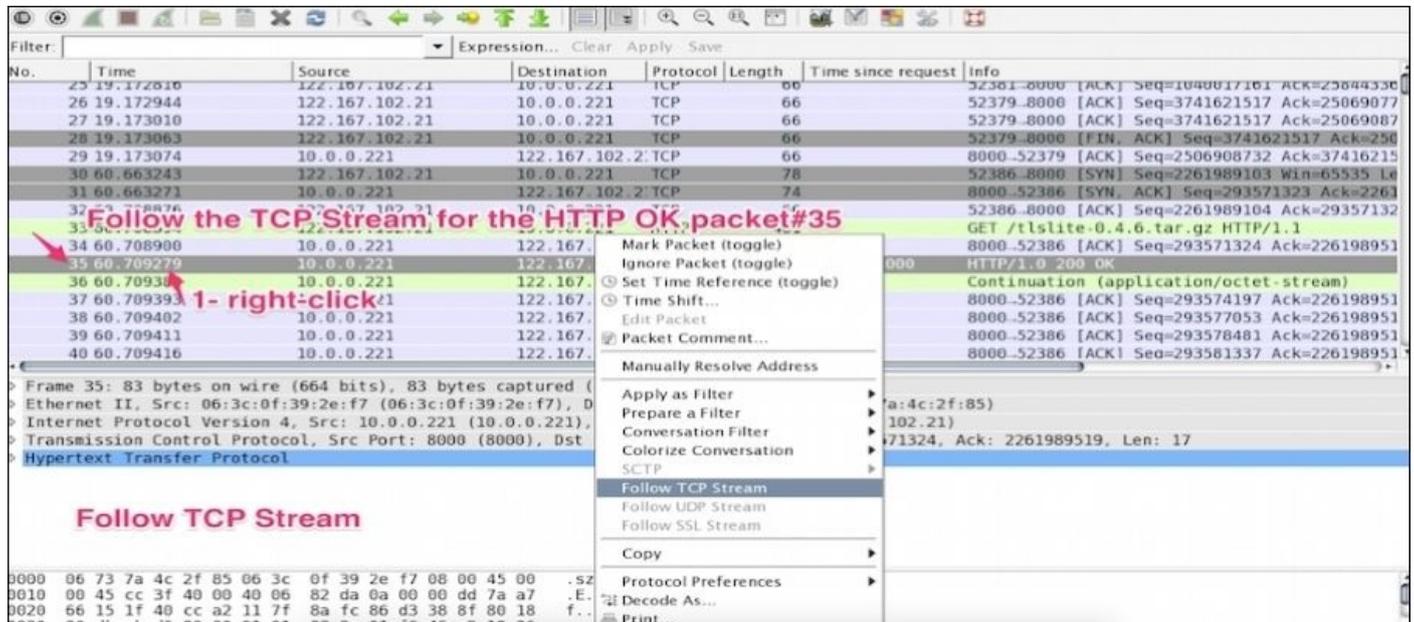
03 - packet list pane shows tcp out of order/dup ack

04 - Apply filter tcp.analysis.duplicate ack

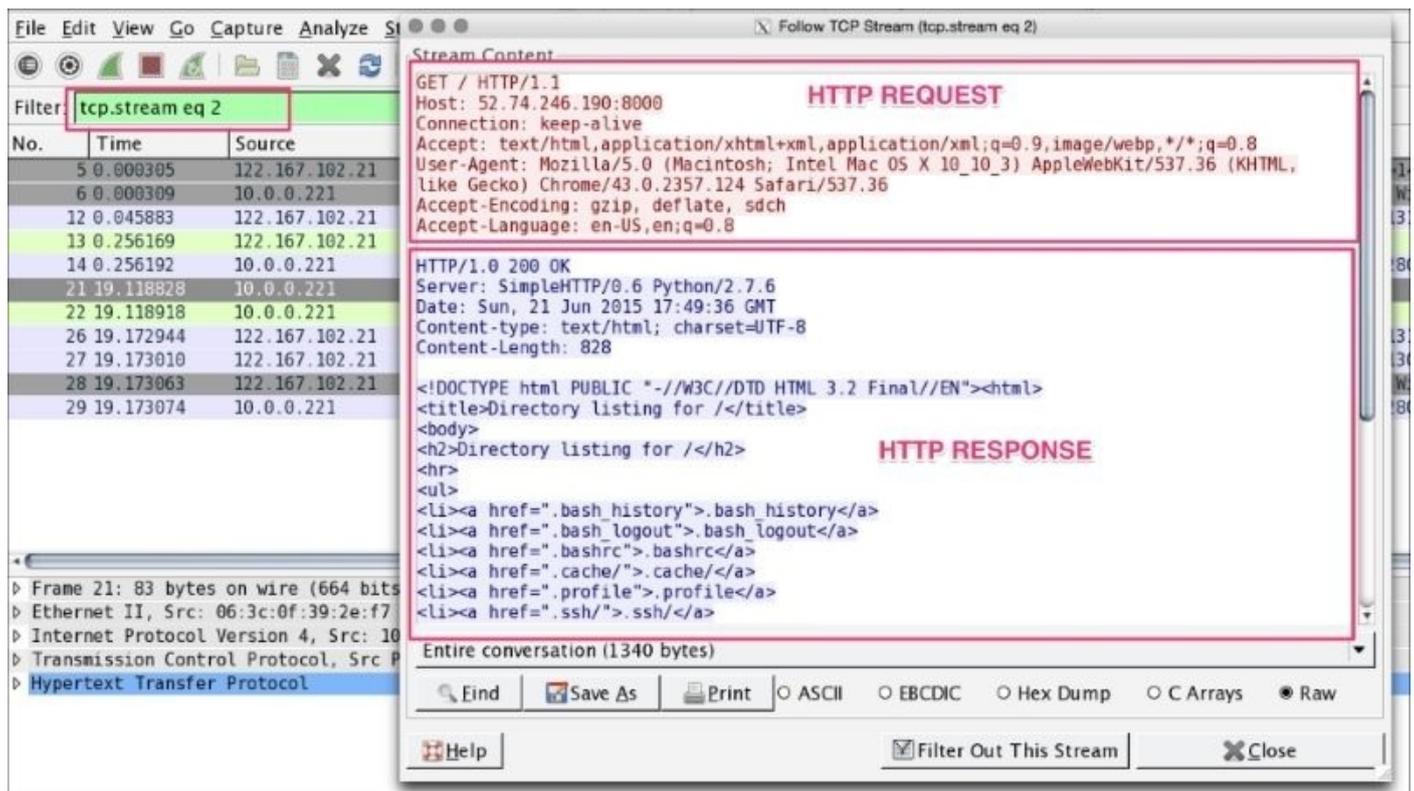
Following the TCP stream

The TCP stream feature allows users to see the data from a TCP stream. Open the file http_01.pcap in Wireshark and follow the TCP stream to get the first HTTP OK, as shown:

In this example we have located the HTTP OK on packet#35 and then right clicked and selected **Follow TCP Stream**:



Once the stream is applied, a TCP stream dialog box will open displaying which request is sent and what response is received in this HTTP conversation:



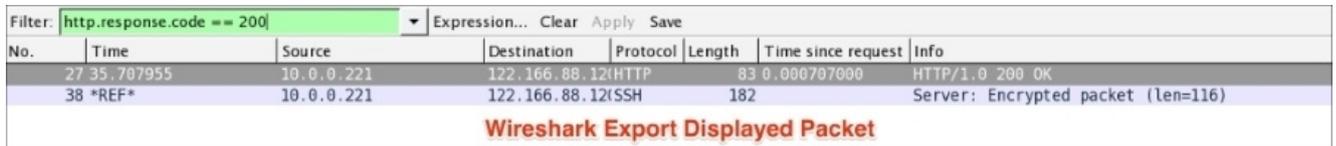
The stream content is available in six formats as shown; the red content in the screenshot is the request, the blue content in the screenshot is the response:



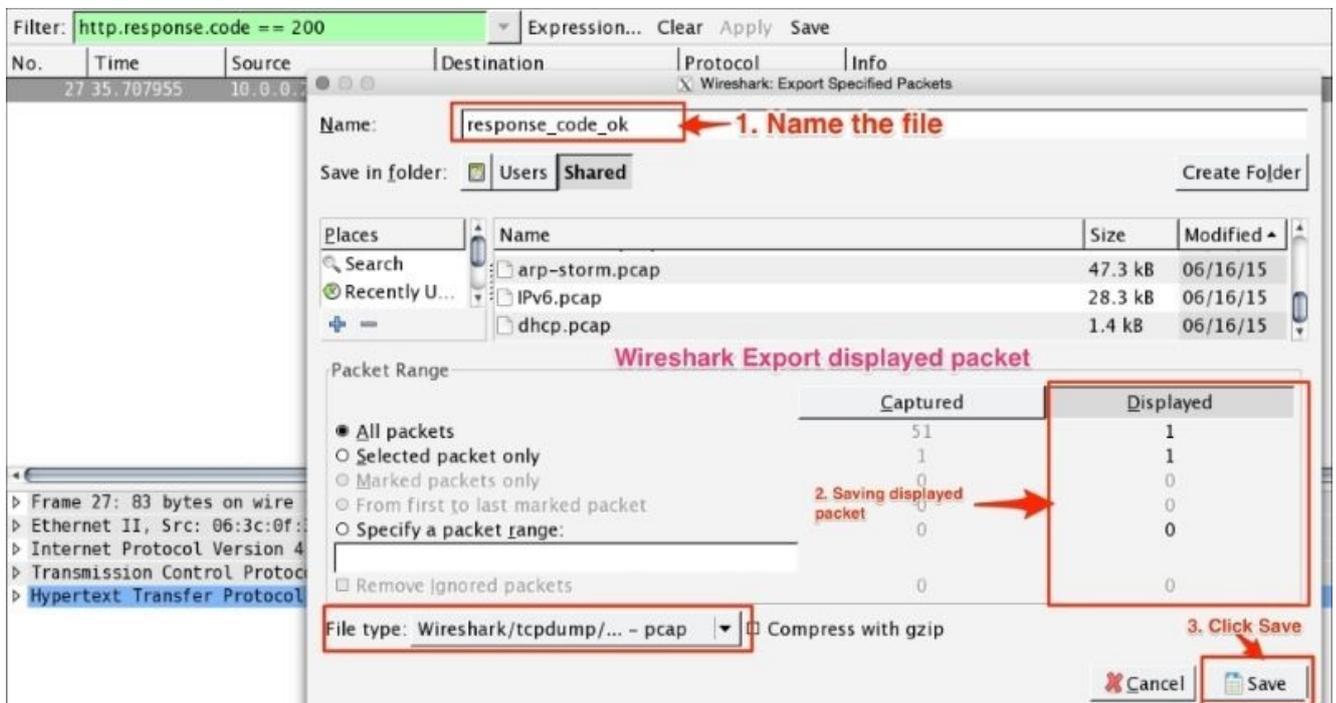
Exporting the displayed packet

The **Export Specified Packets** feature allows you to export the filtered packet in different files. For example, open `http.pcap` in Wireshark and export the HTTP OK packet. The steps for exporting a specified packet are as follows:

1. Apply the filter `http.response.code == 200` in the **Filter** bar:



2. Go to **File | Export Specified Packets**. This opens up the dialog box with the export options, as shown:



Generating the firewall ACL rules

Using Wireshark, network administrators can generate ACL rules for firewall products such as:

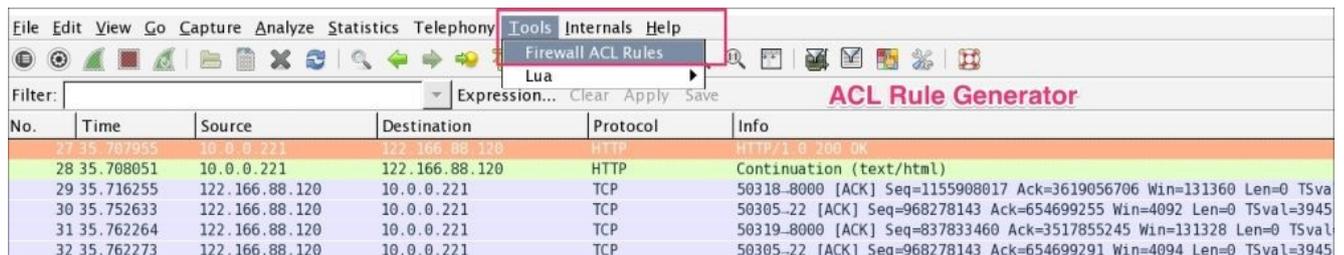
- Cisco IOS
- IP Filter (ipfilter)
- IP Firewall (ipfw)
- Netfilters (iptables)
- Packet Filter (pf)
- Windows Firewall (netsh)

Tip

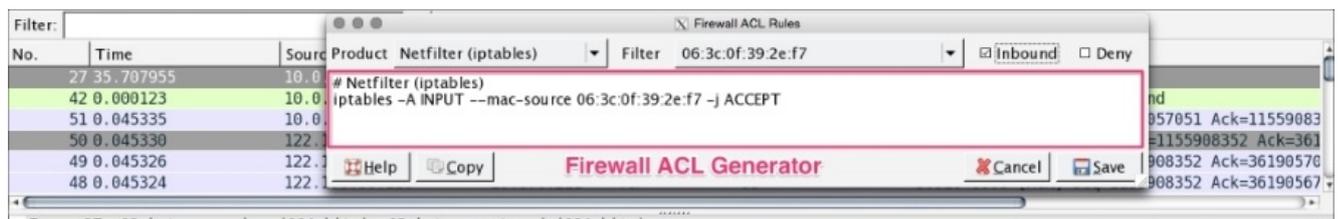
Rules for MAC addresses and IPv4 addresses are present; the filter supports TCP, UDP ports, and IPv4 port combinations.

The steps to generate an ACL rule in Wireshark are as follows:

1. Go to **Tool | Firewall ACL Rules:**



2. The **Firewall ACL Rules** dialog box will appear. Choose **Product** and **Filter**, specify the **ACCEPT/DENY** criteria, and a rule will be generated by Wireshark in this dialog box, as shown:



Tcpdump and snoop

In production environments, packet-capturing tools such as Wireshark are usually not installed. In such scenarios, a default-capturing tool can be used such as tcpdump for (Linux systems) and snoop (the Solaris default); later the captured file can be used in Wireshark for analysis:

- snoop: This tool captures and inspects network packets and runs on Sun Microsystems CLI
- tcpdump: This tool dumps traffic on a network and runs on Windows, OS X, and Linux

For example, the following table shows how to check packets from interfaces:

Description	Solaris	Linux
How to check packets from all interfaces	bash# snoop	bash#tcpdump -nS
How to capture with hostname	bash# snoop hostname	bash# tcpdump host hostname
How to write the captured information to a file	snoop -o filename	bash# tcpdump -w filename
How to capture packets between host1 and host2 and save them to a file	snoop -o capture_file.pcap host1 host2	tcpdump -w capture_file.pcap src host1 and dst host2
How to capture traffic with verbose output to screen	snoop -v -d eth0 snoop -d eth0 -v port 80	tcpdump -i eth0 Very Verbose tcpdump options: tcpdump -i eth0 -v port 80 tcpdump -i eth0 -vv port 80
How to set the snaplength	snoop -s 500	tcpdump -s 500
How to capture all bytes	snoop -s0	tcpdump -s0
How to capture the IPv6 traffic	snoop ip6	tcpdump ip6
How to capture protocols	snoop multicast snoop broadcast snoop bootp snoop dhcp snoop dhcp6 snoop pppoe snoop ldap	tcpdump -n "broadcast or multicast" tcpdump udp tcpdump tcp tcpdump port 67 tcpdump port 546 tcpdump port 389

References

You can also refer to the following links for more information on the topics covered in this chapter:

- https://www.wireshark.org/docs/wsug_html_chunked/
- <https://wiki.wireshark.org/CaptureSetup/Ethernet>
- <https://goo.gl/vxI2jk>

Summary

In this chapter we have learned how to use the Wireshark GUI. Then we explored what capture filters and display filters are, how to set up a capture, keeping performance in mind, and how to make use of other capturing tools such as tcpdump and snoop in production or in remote capturing. Then we learned about a few Wireshark features such as ACL rule generation, IO graph, Decode-As, exporting packets, and protocol preferences.

In the next chapter we will learn the TCP protocol and will discuss its practical use cases with a lab exercise that will help in troubleshooting common network problems (we will also provide the solution).

Chapter 3. Analyzing the TCP Network

TCP is intended to be a host-to-host protocol in common use in multiple networks. In this chapter, we will analyze the TCP protocol in detail with lab exercises and examples.

This chapter covers the following topics:

- Recapping TCP
- TCP connection establishment and clearing
- TCP troubleshooting
- TCP latency issues
- Wireshark TCP sequence analysis

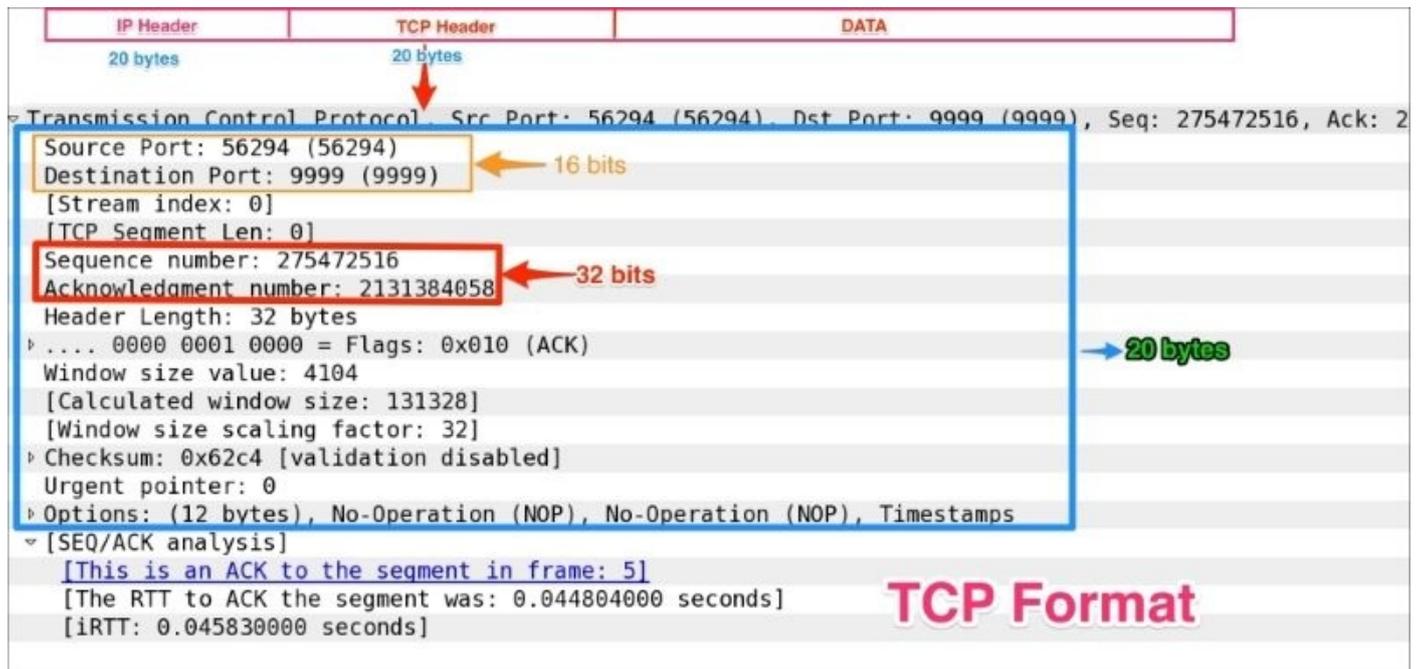
Recapping TCP

Transmission Control Protocol (TCP) was first defined in RFC 675, and the v4 specification came out in RFC 793. TCP provides:

- Connection-oriented setup and tear-down of TCP sessions
- The service sends and receives a stream of bytes, not messages, and guarantees that all bytes received will be identical with bytes sent and in the correct order
- Reliable, in-order delivery, uses sequence number to recover from data that is damaged, lost, duplicated, or delivered out of order by the Internet communication system
- Flow control prevents the receiver's buffer space from overflowing
- Congestion control (as defined in RFC 5681) algorithms are: slow start, congestion avoidance, fast retransmit, and fast recovery
- Multiplexing; every TCP conversation has two logical pipes; an outgoing and incoming pipe

TCP header fields

Each TCP segment has a 20-byte header with optional data values, as shown in the following screenshot displaying a TCP frame in the Wireshark Packet Details pane:



The following table describes the header fields and Wireshark filters along with their descriptions:

TCP header	Wireshark filter name	Description	
Source port (16 bits)	tcp.srcport	Sender port	
Destination port (16 bits)	tcp.dstport	Receiver port	
Sequence Number (32 bits)	tcp.seq	Defines the ISN and controls the state of the TCP	
Acknowledgement number (32 bits)	tcp.ack	The ACK contains the next SEQNo that a host wants to receive	
Flags (9 bits)	tcp.flags	Control bits	
	Reserved	tcp.flags.res	For future use
	Nonce	tcp.flags.ns	Experimental
	CWR	tcp.flags.cwr	Congestion window reduced
	ECN	tcp.flags.ecn	ECN-Echo
	Urgent	tcp.flags.urg	Urgent pointer field is set
	Acknowledgement	tcp.flags.ack	Acknowledgement is set

Push	tcp.flags.push	Push the data
Reset	tcp.flags.reset	Reset the connection
SYN	tcp.flags.syn	Synchronize sequence numbers
FIN	tcp.flags.fin	No more data
Window size (16 bits)	tcp.window_size	Used to advertise the window size in a three-way handshake
Checksum (16 bits)	tcp.checksum	Error checking
Urgent pointer (16 bits)	tcp.urgent_pointer	Inform the receiver that some data in the segment is <i>urgent</i> (SEQNo <= urgent message <= SEQNo + urgent pointer)
Options (0-132 bits) divisible by 32	tcp.options	Options such as maximum segment size, No-Operation (NOP) , window scale, timestamps, SACK permitted

TCP states

A connection progresses through a series of states during its lifetime. The states are:

TCP state	Description
LISTEN	The server is open for incoming connection.
SYN-SENT	The client has initiated the connection.
SYN-RECEIVED	The server has received the connection request.
ESTABLISHED	The client and server are ready for the data transfer, a connection has been established.
FIN-WAIT-1	The client or server has closed the socket. In Linux the default is 60 ms: <pre>[bash ~]# cat /proc/sys/net/ipv4/tcp_fin_timeout 60</pre>
FIN-WAIT-2	The client or server has released the connection. In Linux the default is 60 ms: <pre>[bash ~]# cat /proc/sys/net/ipv4/tcp_fin_timeout 60</pre>
CLOSE-WAIT	Either client or server has not closed the socket. The CLOSE_WAIT state will not expire.
LAST-ACK	Waiting for pending ACK from the client. It's the final stage of the TCP conversation with the client.
TIME-WAIT	TIME_WAIT indicates that the local application closed the connection, and the other side acknowledged and sent a FIN of its own. In Linux the default is 60 ms: <pre>[bash ~]# cat /proc/sys/net/ipv4/tcp_fin_timeout 60</pre>
CLOSED	Fictional state

Note

This socket command-line utility can be used to monitor network connections and their states:

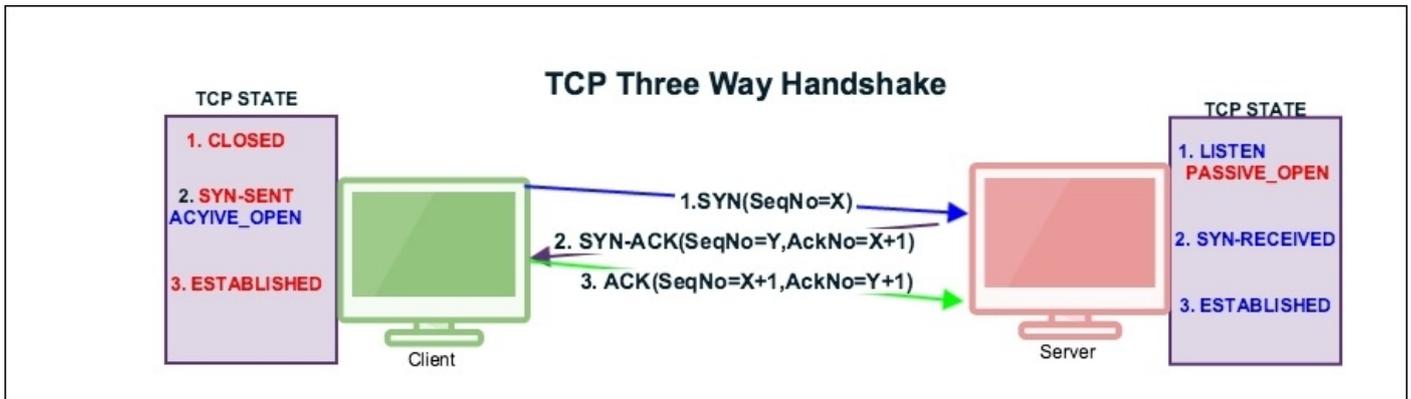
```
[bash ~]ss -nt4 state CLOSE-WAIT  
[bash ~]ss -nt4 state ESTABLISHED  
[bash ~]netstat -an | grep CLOSE-WAIT  
[bash ~]netstat -an | grep ESTABLISHED
```


TCP connection establishment and clearing

In this section we will learn how the TCP opens and closes its connections. In order to establish a connection, the three-way handshake procedure is used as described in the following section.

TCP three-way handshake

The three-way handshake is a connection establishment procedure from the client socket to the server socket, as shown in the following image:



Before the start of the TCP three-way handshake, the client will be in the CLOSED state and the server will be in the LISTEN state as shown:

SN	TCP-A (122.167.84.137) state		Flow CTL	TCP-B (10.0.0.221) state	
	From	To		From	To
1	CLOSED			CLOSED	LISTEN

The TCP state machine

To examine a three-way handshake in Wireshark, open the normal-connection.pcap file provided in the book.

Handshake message – first step [SYN]

The first step of the handshake process is that the socket client will construct a SYN packet and send it to the server. During this process the socket client will perform the following tasks:

1. `tcp.flags.syn` is set to 1 and its SYN packet is sent by the client.
2. The client generates and sets the `tcp.seq=3613047129` the **initial sequence number (ISN)**. Wireshark shows, by default, relative sequence numbers; a user can change this setting under: **Edit | Preferences | Protocols | TCP | Relative sequence numbers**.
3. The client sets `tcp.ack = 0`.
4. The `tcp.window_size` is advertised to the server and its value is in the packet `tcp.window_size_value == 65535`, which tells it that it can transmit up to 65535 bytes of data depending on MSS. For example if MSS is 1440 bytes, the client can transmit 45 segments.
5. TCP client includes other `tcp.options` such as **Maximum Segment Size (MSS)**,

No-Operation (NOP), window scale, timestamps, and SACK permitted.

6. The client chooses `tcp.options.sack_perm == 1` in the “selective acknowledgements” processing.
7. `TSval/TSecr` is the timestamp `tcp.options.timestamp.tsval == 123648340`.

The following table depicts the state transition of the first handshake message:

Sr. No.	TCP-A (122.167.84.137) state		Flow CTL	TCP-B (10.0.0.221) state	
	From	To		From	To
1	CLOSED			CLOSED	LISTEN
2	CLOSED	SYN_SENT	<SEQ=3613047129><CTL=SYN>	LISTEN	

TCP state machine changes SYN_SENT

Handshake message – second step [SYN, ACK]

In this process the server responds to the client’s SYN:

1. The server sets `tcp.flags.syn =1` and `tcp.flags.ack=1`, confirming that the SYN has been accepted.
2. The server generates and sets ISN `tcp.seq=2581725269`.
3. The server sets `tcp.ack=3613047130` as the client `tcp.seq+1`.
4. The server sets `tcp.window_size_value == 26847` as the server window size.
5. The server sets `tcp.options` and responds to the client.

The following table depicts the state transitions of the second handshake message:

Sr. No.	TCP-A (122.167.84.137) state		Flow CTL	TCP-B (10.0.0.221) state	
	From	To		From	To
1	CLOSED			CLOSED	LISTEN
2	CLOSED	SYN_SENT	<SEQ=3613047129><CTL=SYN>	LISTEN	
3	SYN_SENT		<SEQ=2581725269><ACK=3613047130> <CTL=SYN,ACK>	LISTEN	SYN-RECEIVED

TCP state machine changes when SYN-RECEIVED is sent by the server

Handshake message – third step [ACK]

After successfully exchanging this message, the TCP connection will be established in this connection:

1. The client sets `tcp.flags.ack == 1` and sends to the server.
2. The client `tcp.seq=3613047130` is `ISN+1` and `tcp.ack=2581725270` is `SYN_ACK(tcp.seq+1)`.
3. The client window size is set again and this will be used by the server `tcp.window_size_value == 4105`.

Tip

`tcp.analysis.flags` shows you packets that have some kind of expert message from Wireshark.

The following table depicts the state transitions of the third handshake message:

Sr. No.	TCP-A (122.167.84.137) state		Flow CTL	TCP-B (10.0.0.221) state	
	From	To		From	To
1	CLOSED			CLOSED	LISTEN
2	CLOSED	SYN_SENT	<SEQ=3613047129><CTL=SYN>	LISTEN	
3	SYN_SENT		<SEQ=2581725269><ACK=3613047130> <CTL=SYN,ACK>	LISTEN	SYN-RECEIVED
4	SYN_SENT	ESTABLISHED	<SEQ=3613047130>><ACK=2581725270> <CTL=ACK>	SYN-RECEIVED	ESTABLISHED

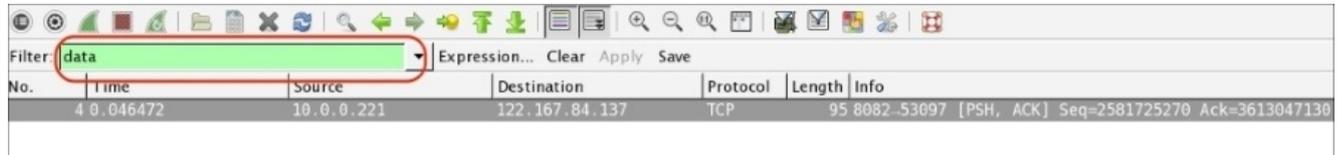
TCP state machine when the client sends ACK

414e495348204e415448204e4f524d414c20434f4e4e4543...

5. The server sets (tcp.flags.ack == 1) && (tcp.flags.push == 1); that is, the [PSH, ACK] flag indicates that the host is acknowledging receipt of some previous data and also transmitting some more data.

The useful Wireshark display filters are:

- data: Displays the packet that contains the data information, for all IPs:

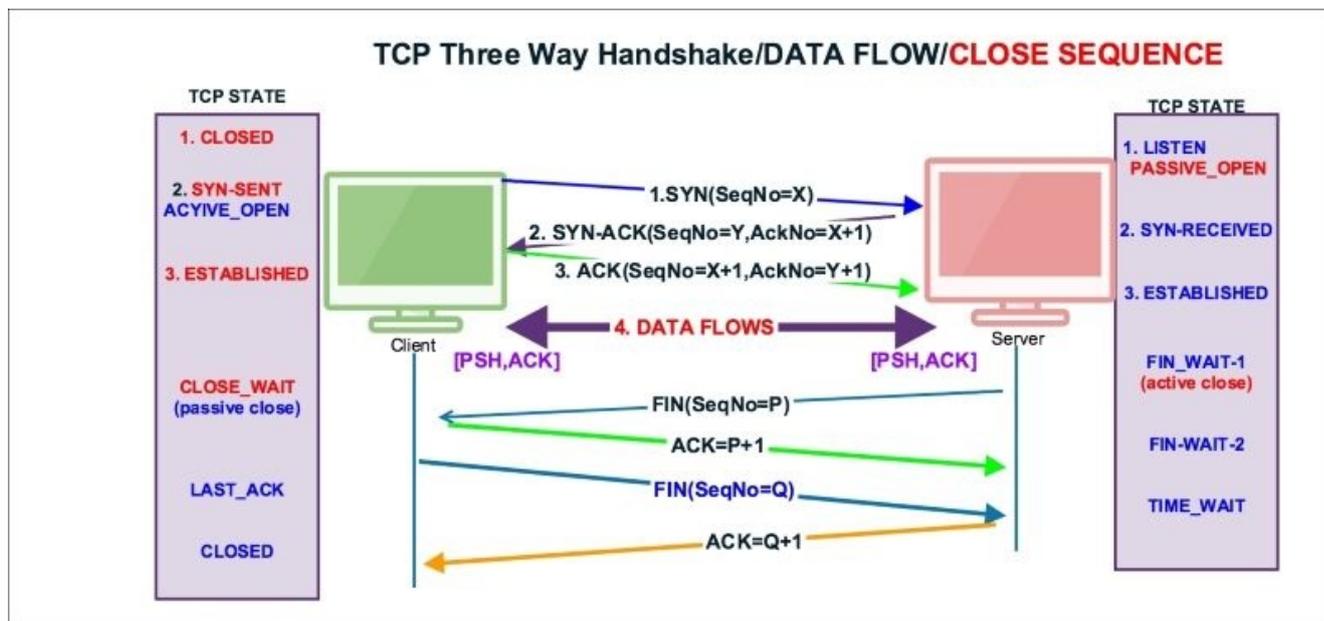


- data && ip.addr==10.0.0.221: Displays a list of packets that have data and are exchanged with the given IP address
- tcp.flags.push == 1: Displays all PUSH packets
- tcp.flags.push == 1 && ip.addr==10.0.0.221: Displays PUSH packets between hosts
- tcp.flags == 0x0018: Display all PSH, ACK packets
- tcp.flags == 0x0011: Displays all FIN, ACK packets
- tcp.flags == 0x0010: Displays all ACK packets

TCP close sequence

TCP normal close appears when the client or server decides that all data has been sent to the receiver and we can close the connection. There are three ways a TCP connection is closed:

- The client initiates closing the connection by sending a FIN packet to the server
- The server initiates closing the connection by sending a FIN packet to the client
- Both client and server initiate closing the connection



Open the normal-connection.pcap file and select packet #5 in the Packet List pane. Go to the Wireshark Packet Details pane, as shown in the screenshot, and examine the TCP protocol.

In Wireshark add the **Sequence number** and **Acknowledgement number** to the column. To add the sequence number and acknowledgement number, choose the TCP header packet, right-click on the field (**Sequence number / Acknowledgement number**) in the packet details and select **Display as Column**. Or implement these settings to add a new column:

- Go to **Edit | Preferences | Columns**. Then add a new column and select “**custom**” : **tcp.seq**.
- Go to **Edit | Preferences | Columns**. Then add a new column and select “**custom**” : **tcp.ack**.

The server has initiated the FIN packet. When the data transfer is completed, see packet#5 in the following screenshot:

No.	Time	Source	Destination	Sequence number	Acknowledgment number	Info
1	0.000000	122.167.84.137	10.0.0.221	3613047129	0	53097-8082 [SYN] Seq=3613047129 Win=65535
2	0.000025	10.0.0.221	122.167.84.13	2581725269	3613047130	8082-53097 [SYN, ACK] Seq=2581725269 Ack=
3	0.045726	122.167.84.137	10.0.0.221	3613047130	2581725270	53097-8082 [ACK] Seq=3613047130 Ack=25817
4	0.046472	10.0.0.221	122.167.84.13	2581725270	3613047130	8082-53097 [PSH, ACK] Seq=2581725270 Ack=
5	0.046656	10.0.0.221	122.167.84.13	2581725299	3613047130	8082-53097 [FIN, ACK] Seq=2581725299 Ack=
6	0.100657	122.167.84.137	10.0.0.221	3613047130	2581725299	53097-8082 [ACK] Seq=3613047130 Ack=25817
7	0.100668	122.167.84.137	10.0.0.221	3613047130	2581725300	53097-8082 [ACK] Seq=3613047130 Ack=25817
8	0.100675	122.167.84.137	10.0.0.221	3613047130	2581725300	53097-8082 [FIN, ACK] Seq=3613047130 Ack=
9	0.100683	10.0.0.221	122.167.84.13	2581725300	3613047131	8082-53097 [ACK] Seq=2581725300 Ack=36130

As you can see in the preceding screenshot:

- The server initiates the FIN packet to close the connection in packet#5
- The server set [FIN,ACK] (tcp.flags.fin == 1) && (tcp.flags.ack == 1) and sends it to the client
- The server sequence number tcp.seq == 2581725299 is acknowledged in packet#7
- The client is initiating FIN to close the connection in packet#8
- The client sets [FIN,ACK] (tcp.flags.fin == 1) && (tcp.flags.ack == 1) and sends it to the server
- The client sequence number tcp.seq == 3613047130 is acknowledged in packet#9

The TCP state machine when the server and client close the socket connection, server initiated FIN:

Sr. No.	TCP-A (122.167.84.137) state		Flow CTL	TCP-B (10.0.0.221) state	
	From	To		From	To
1	CLOSED			CLOSED	LISTEN
2	CLOSED	SYN_SENT	<SEQ=3613047129><CTL=SYN>	LISTEN	
3	SYN_SENT		<SEQ=2581725269> <ACK=3613047130><CTL=SYN,ACK>	LISTEN	SYN-RECEIVED
4	SYN_SENT	ESTABLISHED	SEQ=3613047130>> <ACK=2581725270><CTL=ACK>	SYN-RECEIVED	ESTABLISHED
5	ESTABLISHED	ESTABLISHED	<SEQ=3613047130>> <ACK=2581725270><CTL=PSH,ACK>	ESTABLISHED	ESTABLISHED
6	ESTABLISHED	ESTABLISHED	<SEQ=3613047130>> <ACK=2581725299><CTL=ACK>	ESTABLISHED	ESTABLISHED
7	ESTABLISHED	ESTABLISHED	<SEQ=2581725299>> <ACK=3613047130><CTL=FIN,ACK>	ESTABLISHED	FIN_WAIT-1
8	ESTABLISHED	CLOSE_WAIT	<SEQ=3613047130>> <ACK=2581725300><CTL=ACK>	FIN_WAIT-1	FIN_WAIT-2
9	CLOSE_WAIT	LAST_ACK	SEQ=3613047130>> <ACK=2581725300><CTL=FIN,ACK>	FIN_WAIT-2	TIME_WAIT

10	LAST_ACK	CLOSED		TIME_WAIT	CLOSED
----	----------	--------	--	-----------	--------

Wireshark filters used in this scenario are as follows:

- `tcp.analysis:SEQ/ACK`: Provides links to the segments of the matching sequence/ack numbers
- `tcp.connection.fin`: Provides expert information
- `tcp.flags == 0x0011`: Displays all the [FIN,ACK] packets

Lab exercise

The steps to capture the normal TCP connection flow (a sample program is provided as part of this book) are as follows:

1. Open Wireshark, start capturing the packets, and choose display filter `tcp.port==8082`.
2. Compile the Java program `TCPServer01.java` using the `javac` command:

```
bash$ ~ javac TCPServer01.java
```

3. Run `TCPServer01` using the `java` command:

```
bash$ ~ java TCPServer01
```

4. Verify the server is listening on port 8082:

```
bash$ ~ netstat -an | grep 8082  
tcp46      0      0 *.8082          *.* LISTEN
```

5. Compile the client program `Client0301.java` using the `javac` command:

```
bash$ ~ javac Client0301.java
```

6. Run the client program:

```
bash$ ~ java Client0301
```

7. View and analyze the packet in Wireshark.

TCP troubleshooting

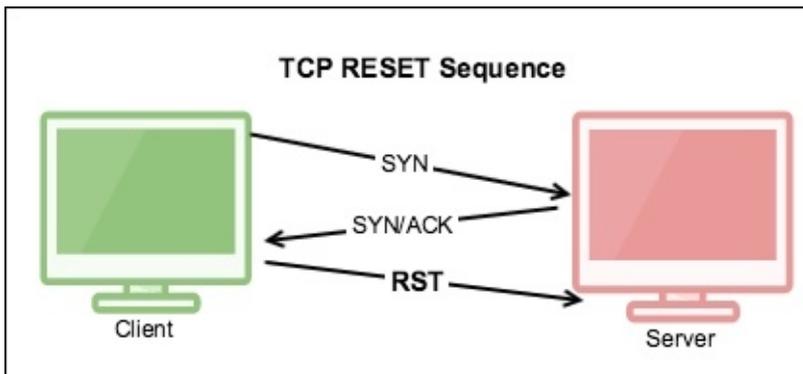
In this section we will learn about different network problems that occur and try to analyze and solve them with lab exercises. Let's start with the Reset (RST) packet.

TCP reset sequence

The TCP RST flag resets the connection. It indicates that the receiver should delete the connection. The receiver deletes the connection based on the sequence number and header information. If a connection doesn't exist on the receiver RST is set, and it can come at any time during the TCP connection lifecycle due to abnormal behavior. Let's take one example: a RST packet is sent after receiving SYN/ACK, as shown in the next image.

RST after SYN-ACK

In this example we will see why RST has been set after SYN-ACK instead of ACK:



Open the RST-01.pcap file in the Wireshark:

The image shows a Wireshark packet capture. The top part is a packet list table with three entries:

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.107	10.0.0.221	TCP	1500-9999 [SYN] Seq=100 Win=8192 Len=0
2	0.000020	10.0.0.221	10.0.0.107	TCP	9999-1500 [SYN, ACK] Seq=1404263211 Ack=101 Win=26883
3	0.000325	10.0.0.107	10.0.0.221	TCP	1500-9999 [RST] Seq=101 Win=0 Len=0

Red arrows point to the SYN and SYN/ACK packets with the text "first two handshake happen". Another red arrow points to the RST packet with the text "Connection reseted during final handshake process".

The details pane for the RST packet (No. 3) is expanded, showing the following fields:

- Source Port: 1500 (1500)
- Destination Port: 9999 (9999)
- [Stream index: 0]
- [TCP Segment Len: 0]
- Sequence number: 101
- Acknowledgment number: 0
- Header Length: 20 bytes
- ... 0000 0000 0100 = Flags: 0x004 (RST) ← RST flag Set
- Window size value: 0
- [Calculated window size: 0]
- [Window size scaling factor: -2 (no window scaling used)]

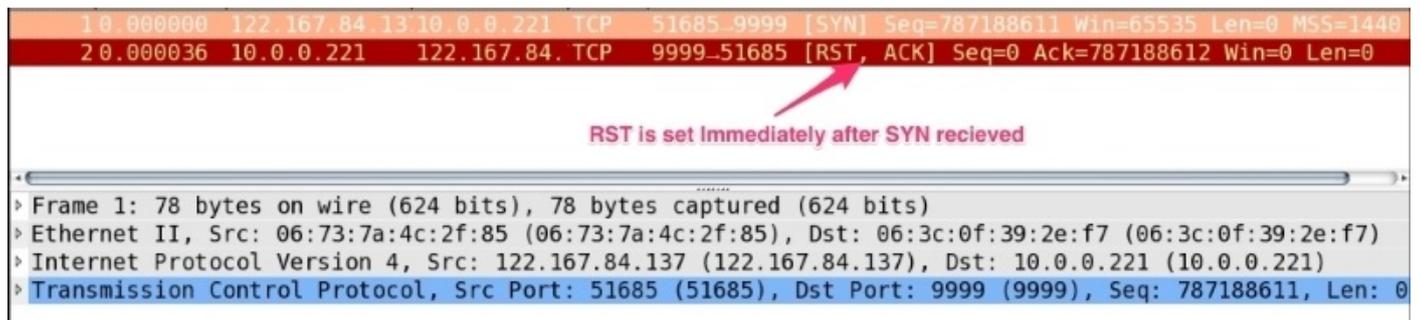
As you can see in the preceding figure:

- The TCP RST packet should not be seen normally
- The TCP RST is set after the first two handshakes are complete. A possible explanation could be one of the following:

- The client connection never existed; a RAW packet was send over the TCP server
- The client aborted its connection
- The sequence number got changed/forged

RST after SYN

This is the most common use case. Open the RST-02-ServerSocket-CLOSED.pcap file in Wireshark. In this example the server was not started, the client attempted to make a connection, and the connection refused an RST packet:



Lab exercise

The steps to generate the RST flag in a generic scenario, when the server is not in the listening state, are as follows:

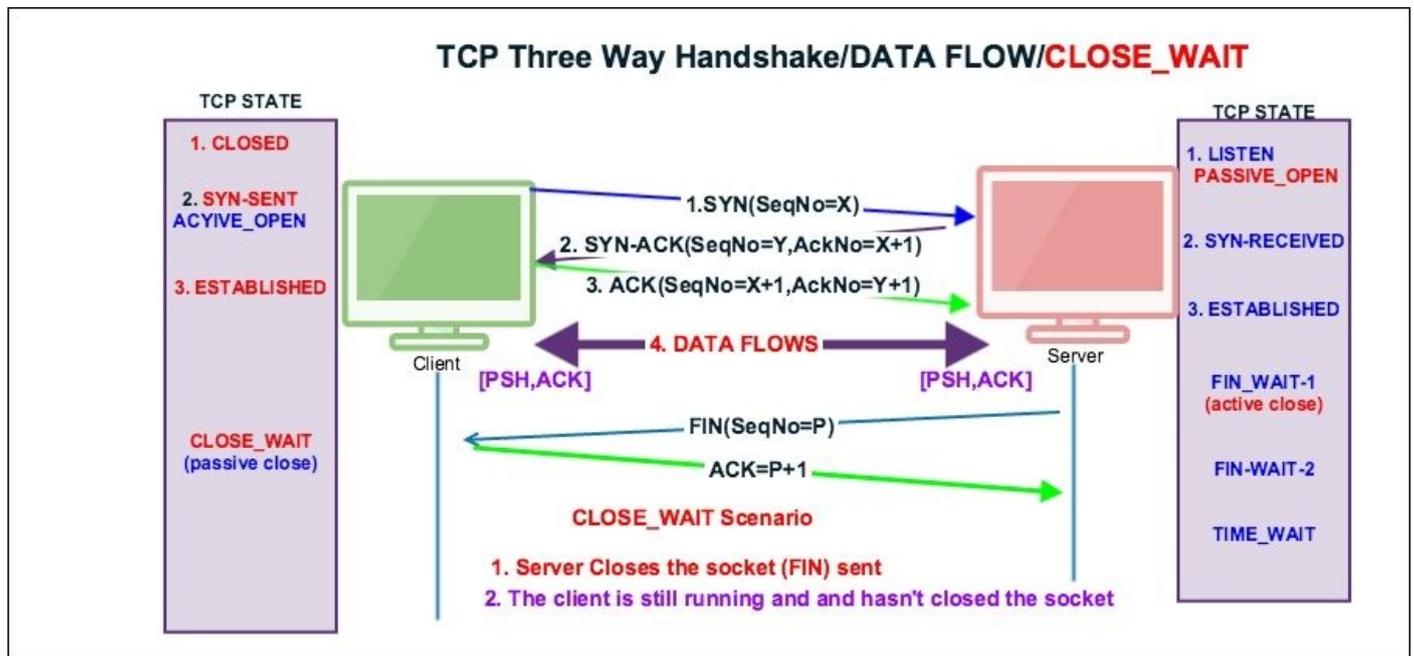
1. Open Wireshark, start capturing the packets, and choose display filter `tcp.port==8082`.
2. Compile the client program `Client0301.java`:

```
bash$ ~ javac Client0301.java
```
3. Run the client program:

```
bash$ ~ java Client0301
```
4. View and analyze the RST packet in Wireshark.

TCP CLOSE_WAIT

Often a connection is stuck in the CLOSE_WAIT state. This scenario typically occurs when the receiver is waiting for a connection termination request from the peer.



Tip

To find a socket in the CLOSE_WAIT state, use the following commands:

```
bash:~ $ netstat -an | grep CLOSE_WAIT
tcp4      0      0 122.167.127.21.56294    10.0.0.21.9999        CLOSE_WAIT
```

To demonstrate the CLOSE_WAIT state, open the close_wait.pcap file in Wireshark:

No.	Time	Source	Destination	Protocol	Sequence number	Acknowledgment number	Info
1	0.000000	122.167.127.21	10.0.0.221	TCP	275472515	0	56294-9999 [SYN] Seq=2754
2	0.000024	10.0.0.221	122.167.127	TCP	2131384030	275472516	9999-56294 [SYN, ACK] Seq
3	0.045830	122.167.127.21	10.0.0.221	TCP	275472516	2131384031	56294-9999 [ACK] Seq=2754
4	0.046540	10.0.0.221	122.167.127	TCP	2131384031	275472516	9999-56294 [ACK] Seq=2754
5	0.046699	10.0.0.221	122.167.127	TCP	2131384057	275472516	9999-56294 [FIN, ACK] Seq
6	0.091496	122.167.127.21	10.0.0.221	TCP	275472516	2131384057	56294-9999 [ACK] Seq=2754
7	0.091503	122.167.127.21	10.0.0.221	TCP	275472516	2131384058	56294-9999 [ACK] Seq=2754

Internet Protocol Version 4, Src: 10.0.0.221 (10.0.0.221), Dst: 122.167.127.21 (122.167.127.21)

Transmission Control Protocol, Src Port: 9999 (9999), Dst Port: 56294 (56294), Seq: 2131384057, Ack: 275472516

Source Port: 9999 (9999)

Destination Port: 56294 (56294)

[Stream index: 0]

[TCP Segment Len: 0]

Sequence number: 2131384057 ← ACK received in Packet#7

Acknowledgment number: 275472516

Header Length: 32 bytes

... 0000 0001 0001 = Flags: 0x011 (FIN, ACK) ← Server Close the socket

Window size value: 210

As you can see in the preceding screenshot:

1. The server closed socket packet#5, set `tcp.flags.fin == 1`, and set `tcp.seq == 2131384057`.
2. The client responded with the ACK packet `tcp.ack == 2131384058` in packet#7 and didn't close its socket, which remains in the `CLOSE_WAIT` state.

`CLOSE_WAIT` means there is something wrong with the application code, and in the high-traffic environment if `CLOSE_WAIT` keeps increasing, it can make your application process slow and can crash it.

Lab exercise

The steps to reproduce `CLOSE_WAIT` are as follows:

1. Open Wireshark, start capturing the packets, and choose display filter `tcp.port==9999`.
2. Compile the Java programs `Server0302.java` and `Client0302.java` using the `javac` command:

```
bash$ ~ javac Server0302.java Client0302.java
```

3. Run `Server0302` using the `java` command:

```
bash$ ~ java TCPServer01
```

4. Verify the server is listening on port 9999:

```
bash $ netstat -an | grep 999
tcp4        0      0  *.9999          *.* LISTEN
```

5. Run the client program:

```
bash$ ~ java Client0302
```

6. Check the state of the TCP socket; it will be in the `CLOSE_WAIT` state:

```
bash $ netstat -an | grep CLOSE_WAIT
tcp4        0      0  127.0.0.1.56960  127.0.0.1.9999
CLOSE_WAIT
```

7. Analyze the packet in Wireshark.

How to resolve TCP CLOSE_STATE

The steps are as follows:

1. To remove `CLOSE_WAIT`, a restart is required for the process.
2. Establishing the `FIN` packet from both the client and server is required to solve the `CLOSE_WAIT` problem. Close the client socket and server socket when done with processing the record:

```
socket.close(); à Initiates the FIN flow
```

3. Open the `Client0302.java` file and close the socket:

```
Socket socket = new Socket(InetAddress.getByName("localhost"), 9999);  
...  
socket.close();  
...  
Thread.sleep(Integer.MAX_VALUE);
```

4. Compile and re-run the Java program. CLOSE_WAIT will not be visible.

TCP TIME_WAIT

The main purpose of the TIME_WAIT state is to close a connection gracefully, when one of ends sits in LAST_ACK or CLOSING retransmitting FIN and one or more of our ACK are lost.

RFC 1122: *“When a connection is closed actively, it MUST linger in TIME-WAIT state for a time 2xMSL (Maximum Segment Lifetime). However, it MAY accept a new SYN from the remote TCP to reopen the connection directly from TIME-WAIT state, if...”*

We ignore the conditions because we are in the TIME_WAIT state anyway.

TCP latency issues

Until now we have been troubleshooting connection-related issues. In this section, we will check the latency part. Latency can be on the network, or in application processing on the part of the client or server.

Cause of latency

Identifying the source of latency also plays an important role in TCP troubleshooting. Let's see what the common causes of latency are:

- Network slow wire latency can be measured with the ping utility
- Too many running processes eat memory. Check the memory management, work with free, top command to identify CPU and memory use
- Application not started with sufficient memory or cannot serve more requests
- Bad TCP tuning; verify the /etc/sysctl.conf file
- Network jitter; verify your network and check with the network administrator
- Poor coding; benchmark your code by performing a load test over the network
- Gateway wrongly set; check the gateway, verify the routing table, and verify the gateway
- Higher hop counts; do a traceroute and check the number of hops (the higher the hop count, the more latency increases)
- Slow NIC interface, the interface goes down; check the NIC card and verify its speed

Identifying latency

Various network utility tools are available to measure the latency between networks—for example traceroute, tcpping, and ping.

- ping: This utility can be used to measure the **round trip time (RTT)**:

```
bash$ ping -c4 google.com
PING google.com (216.58.196.110): 56 data bytes
64 bytes from 216.58.196.110: icmp_seq=0 ttl=55 time=226.034 ms
64 bytes from 216.58.196.110: icmp_seq=1 ttl=55 time=207.748 ms
64 bytes from 216.58.196.110: icmp_seq=2 ttl=55 time=222.995 ms
64 bytes from 216.58.196.110: icmp_seq=3 ttl=55 time=162.507 ms

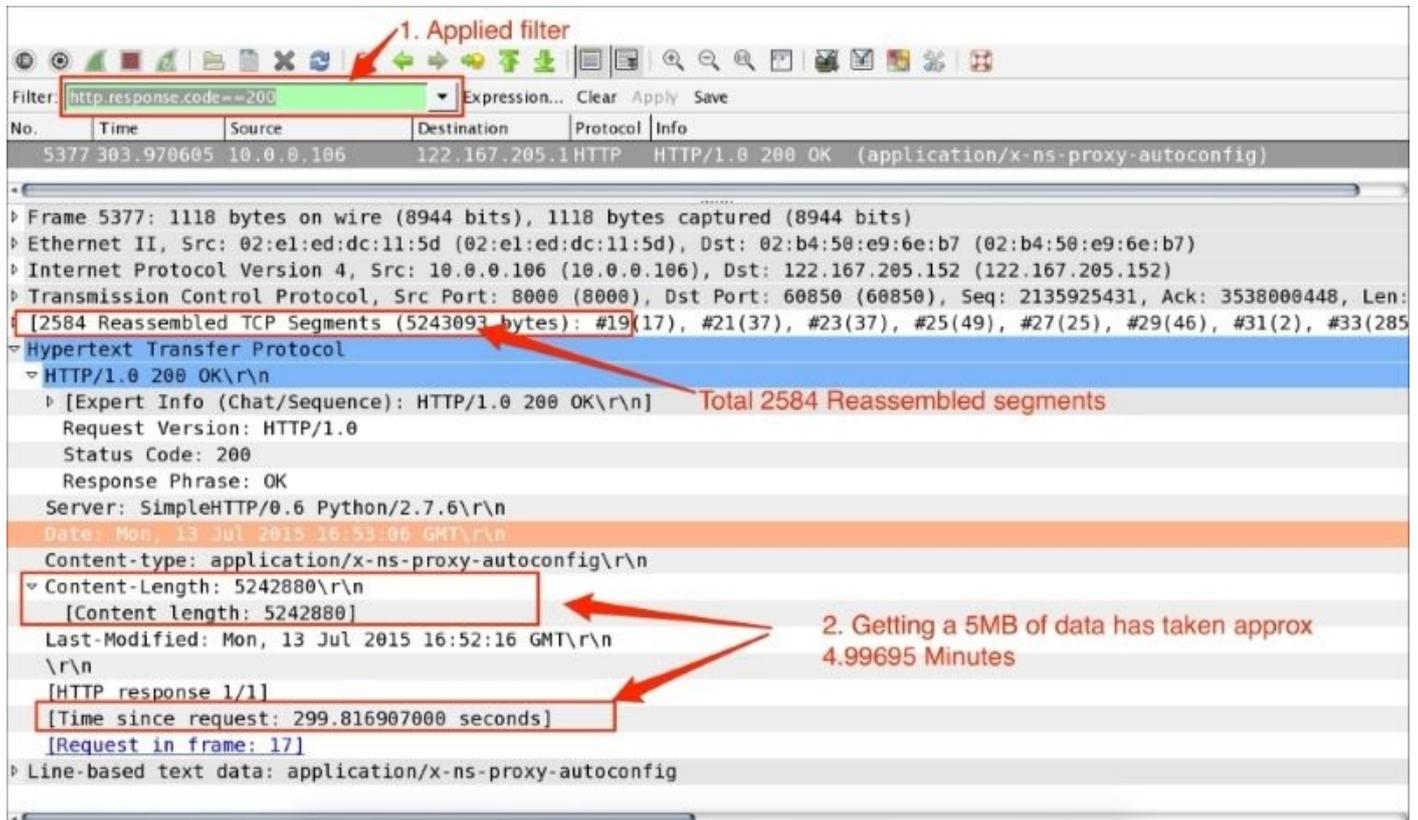
--- google.com ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 162.507/204.821/226.034/25.394 ms
```

- traceroute: This is used to identify the number of HOPS it has taken to reach the destination—the fewer the hops, the lower the latency

Server latency example

Wireshark can be used effectively to identify whether the network is slow or the application is slow. Open the `slow_download.pcap` file in Wireshark, and investigate the root cause of why the download is slow.

In this example, 5 MB of data is requested from the HTTP server, and it has taken approx. 4.99 minutes to download, as shown:



The steps to diagnose this issue are as follows:

1. Go to **Edit | Preferences | Protocols | HTTP** and then enable all HTTP reassemble options.
2. Apply the filter `http.response.code==200`.
3. Go to **HTTP** and set the `http.time == 299.816907000` to approximately 4.99 minutes.
4. Check the size of the file by navigating to `http.content_length_header == "5242880"`; this is the size of the content.
5. Check how many TCP segments have been sent— `tcp.segment.count == 2584`— and ask yourself whether so many are needed and whether the number can be reduced.
6. Verify `window_size` for the client and server to check what was advertised by the client and what got used.
7. Add `tcp.window_size_value` in the **Wireshark** column and sort in ascending order. Note that the entire packet flow from the server (`10.0.0.16`) to the client (`122.167.205.152`) has a window size of 100.

8. Verify the `sysctl.conf` file in UNIX-flavored systems and check the TCP tuning parameters such as `net.core.rmem_max`, `net.core.wmem_max`, `net.ipv4.tcp_rmem`, and `net.ipv4.tcp_wmem`.

Tip

Make sure `tcp.window_size` stays large enough to avoid slowing down the sender. The window size can tell you if a system is too slow when processing incoming data; `tcp_window_size` indicates that the system is slow, not the network.

In this scenario, `tcp.window_size` was reduced in the `sysctl.conf` file to demonstrate the `slow_download` behavior and to give an insight into troubleshooting. After fixing `window_size`, the same download is reduced from 299.816907000 to 2.84 seconds. Open the `fast_download.pcap` file as shown in the following screenshot; the download time is reduced:

The screenshot shows a Wireshark capture of an HTTP response. The filter bar at the top is set to `http.response.code==200` and is labeled "Filter applied". The packet list pane shows a packet of type "HTTP" with a status of "200 OK". The packet details pane is expanded to show the "Hypertext Transfer Protocol" section. Several fields are highlighted with red boxes: "Content-Length: 5242880", "[Time since request: 2.842783000 seconds]", and "[Request in frame: 4]". A red arrow points from the text "Lesser Number of Segments" to the "Hypertext Transfer Protocol" section. Another red arrow points from the text "Getting 5MB of data has taken approx 2.84 Seconds" to the "Time since request" field.

No.	Time	Source	Destination	Protocol	Window size value	Info
2625	2.889089	10.0.0.106	122.167.205.1	HTTP	235	HTTP/1.0 200 OK (application/x-ns-proxy-autoconfig)

Filter: `http.response.code==200` Expression... Clear Apply Save

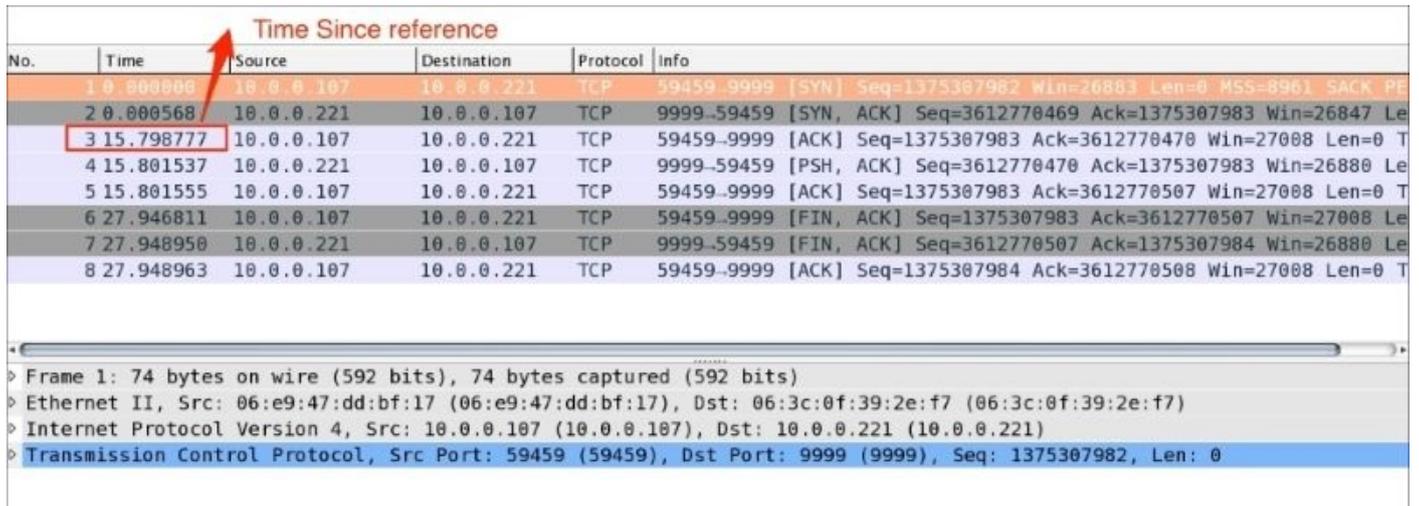
Filter applied

Lesser Number of Segments

Getting 5MB of data has taken approx 2.84 Seconds

Wire latency

In this example, the TCP handshake process will be used to identify wire latency. Open the `slow_client_ack.pcap` file as shown in the following screenshot:



No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.107	10.0.0.221	TCP	59459-9999 [SYN] Seq=1375307982 Win=26883 Len=0 MSS=8961 SACK PE
2	0.000568	10.0.0.221	10.0.0.107	TCP	9999-59459 [SYN, ACK] Seq=3612770469 Ack=1375307983 Win=26847 Le
3	15.798777	10.0.0.107	10.0.0.221	TCP	59459-9999 [ACK] Seq=1375307983 Ack=3612770470 Win=27008 Len=0 T
4	15.801537	10.0.0.221	10.0.0.107	TCP	9999-59459 [PSH, ACK] Seq=3612770470 Ack=1375307983 Win=26880 Le
5	15.801555	10.0.0.107	10.0.0.221	TCP	59459-9999 [ACK] Seq=1375307983 Ack=3612770507 Win=27008 Len=0 T
6	27.946811	10.0.0.107	10.0.0.221	TCP	59459-9999 [FIN, ACK] Seq=1375307983 Ack=3612770507 Win=27008 Le
7	27.948950	10.0.0.221	10.0.0.107	TCP	9999-59459 [FIN, ACK] Seq=3612770507 Ack=1375307984 Win=26880 Le
8	27.948963	10.0.0.107	10.0.0.221	TCP	59459-9999 [ACK] Seq=1375307984 Ack=3612770508 Win=27008 Len=0 T

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
Ethernet II, Src: 06:e9:47:dd:bf:17 (06:e9:47:dd:bf:17), Dst: 06:3c:0f:39:2e:f7 (06:3c:0f:39:2e:f7)
Internet Protocol Version 4, Src: 10.0.0.107 (10.0.0.107), Dst: 10.0.0.221 (10.0.0.221)
Transmission Control Protocol, Src Port: 59459 (59459), Dst Port: 9999 (9999), Seq: 1375307982, Len: 0

As you can see in the preceding screenshot:

- The first two handshake messages (SYN, SYN-ACK) sent by the client/server over the wire are exchanged in less time
- In the last handshake message, ACK sent by the client has taken `frame.time_relative == 15.798777000` seconds and shows an increase in **Time Since Reference**. This is higher than the first two handshake messages, which confirms a wire latency on this packet
- Once the handshake is completed, the operation resumes normally; the Time Since reference for all packets shows a consistent timing

Wireshark TCP sequence analysis

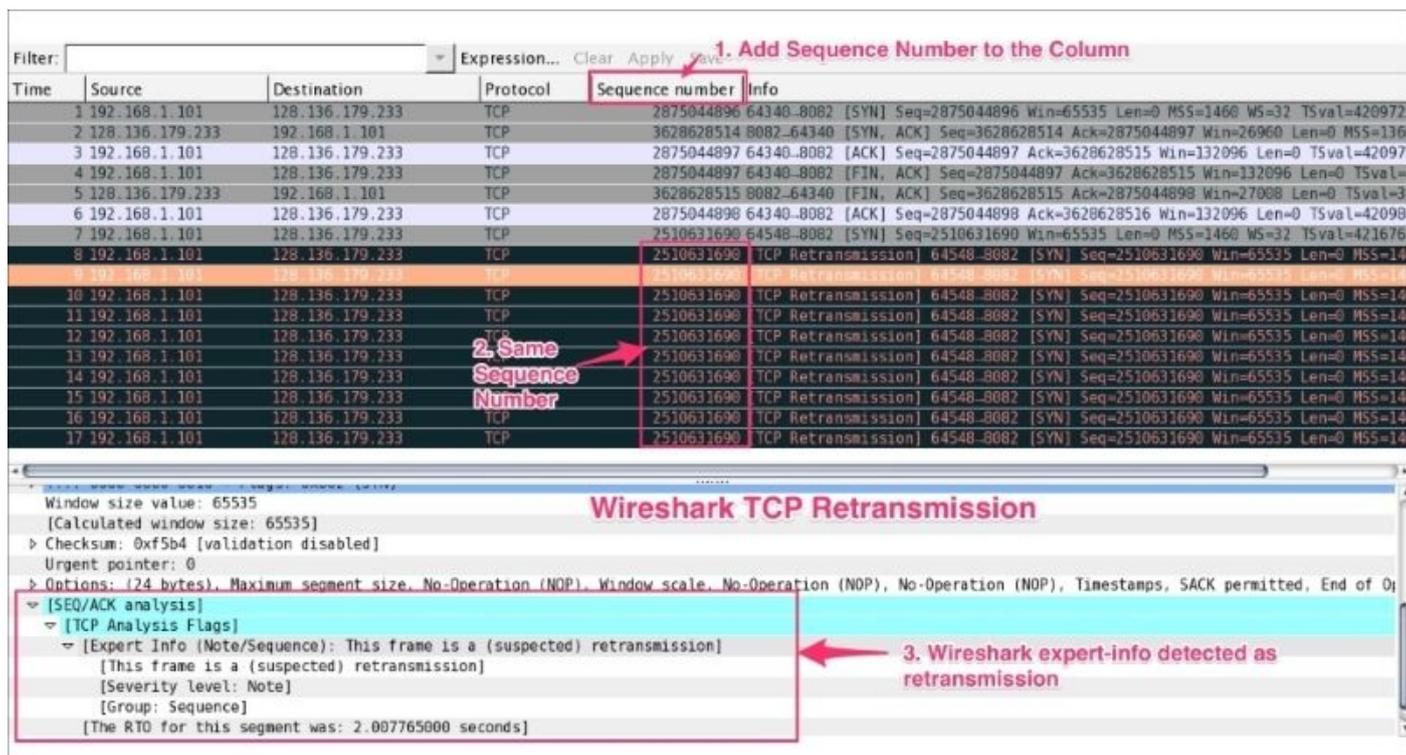
Wireshark has a built-in filter, `tcp.analysis.flags`, that will show you packets that have some kind of expert message from Wireshark; `tcp.analysis.flags` is shown in the **TCP** section of the **Packet Details** pane. Under that, expand **SEQ/ACK analysis** then expand **TCP Analysis Flags**. This will tell you exactly what triggered `tcp.analysis.flags`. A few examples include:

- TCP Retransmission
- TCP Fast Retransmission
- TCP DupACK
- TCP ZeroWindow
- TCP ZeroWindowProbe

TCP retransmission

TCP makes the transmission of segments reliable via sequence number and acknowledgement. When TCP transmits a segment containing data, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that data is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted. During TCP retransmission, the sequence number is not changed until the retransmission timeout happens.

Open the example `tcp-retransmission.pcapng` in Wireshark and add a **Sequence number** column, as shown in the following screenshot:



As you can see in the preceding screenshot:

- After sending `tcp.seq == 1870089183` a lot of TCP retransmission occurs
- A lot of TCP Retransmission can result in operation timeouts

For another example, open the file `syn_sent_timeout_SSH.pcapng` in Wireshark, and observe the TCP retransmission flow.

Tip

KeepAlive is not a retransmission.

Lab exercise

The steps to reproduce the TCP retransmission are as follows (this lab is performed in CentOS6 using the `telnet` and `nc` command utilities):

1. Set up two machines: HOST-A (Server) and HOST-B (client).

2. On HOST-A start the server and configure the firewall rule as shown:

```
[bash ~]# iptables -A OUTPUT -p tcp --dport 8082 -j DROP
[bash ~]# iptables save
[bash ~]# nc -l 8082
```

3. On the HOST-B machine open Wireshark, start capturing the packets, and choose display filter `tcp.port==8082`.

4. On the HOST-B machine run the telnet command; change the IP information to your actual server location:

```
[bash ~]telnet 128.136.179.233 8082
```

5. Verify the TCP state on the HOST-B machine:

```
bash$ netstat -an | grep 8082
tcp4      0      0 192.168.1.101.64658    128.136.179.233.8082
SYN_SENT
```

6. In Wireshark, view and analyze the captured packet using the previous step.

In order to solve operation timeouts, verify the ACL configuration; it allows the incoming packet from the source IP.

TCP ZeroWindow

Open the tcp_zero_window.pcapng file in Wireshark and add tcp.window_size_value to the column.

The TCP window size represents how much data a device can handle from its peer at one time before it is passed to the application process.

No.	Time	Source	Destination	Protocol	Window size value	Info
1	0.000000000	192.168.1.101	54.169.134.196	TCP	65535	52638-8000 [SYN] Seq=322304689
2	0.066711000	54.169.134.196	192.168.1.101	TCP	100	8000-52638 [SYN, ACK] Seq=634335905
3	0.066791000	192.168.1.101	54.169.134.196	TCP	4105	52638-8000 [ACK] Seq=322304689
4	0.066953000	192.168.1.101	54.169.134.196	TCP	4105	[TCP Window Full] [TCP segment
5	0.112259000	54.169.134.196	192.168.1.101	TCP	0	[TCP ZeroWindow] 8000-52638 [A
6	0.112262000	54.169.134.196	192.168.1.101	TCP	100	[TCP Window Update] 8000-52638
7	0.112355000	192.168.1.101	54.169.134.196	TCP	4105	[TCP Window Full] [TCP segment
8	0.167624000	54.169.134.196	192.168.1.101	TCP	100	8000-52638 [ACK] Seq=634335905

Source Port: 8000 (8000)
Destination Port: 52638 (52638)
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 634335905
Acknowledgment number: 3223046992
Header Length: 32 bytes
... 0000 0001 0000 = Flags: 0x010 (ACK)
Window size value: 0
[Calculated window size: 0]
[Window size scaling factor: 1]
Checksum: 0xc785 [validation disabled]
Urgent pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps

As shown in the preceding screenshot:

- Add window_size to the Wireshark column and look for the packet where tcp.window_size=0.
- TCP headers with a window size of zero indicate that the receiver's buffers are full. This condition arrives more rapidly for writes than reads; in this condition tcp.window_size_value is set to 0 and tcp.window_size == 0.
- The segment is exactly 1 byte.

Tip

SYN/RST/FIN flags are never set on TCP ZeroWindow.

SYN/RST/FIN flags are never set on TCP Window Full.

Troubleshoot the ZeroWindow condition:

- Check the application has sufficient memory to start with
- Tune the TCP parameters to obtain a larger window size; check the sysctl.conf file with these parameters:
 - net.core.rmem_max

- `net.core.wmem_max`
- `net.ipv4.tcp_rmem`
- `net.ipv4.tcp_wmem`

- Check the receiver is not running too many processes

TCP Window Update

Wireshark marks a packet as Window Update when the window size has changed. A Window Update is an ACK packet, and only expands the window; this is normal TCP behavior.

Open the `tcp_window_update.pcap` file in Wireshark and observe that a TCP Window Update event is set, as shown:

The image shows a Wireshark packet capture of a TCP Window Update. The packet list pane shows the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	68	54106-9999 [SYN] Seq=4271183518 Win=65535 Len=0 MSS=16344 WS=32 TSval=471
2	0.000091	127.0.0.1	127.0.0.1	TCP	68	9999-54106 [SYN, ACK] Seq=208317014 Ack=4271183519 Win=65535 Len=0 MSS=16
3	0.000106	127.0.0.1	127.0.0.1	TCP	56	54106-9999 [ACK] Seq=4271183519 Ack=208317015 Win=408288 Len=0 TSval=4717
4	0.000118	127.0.0.1	127.0.0.1	TCP	56	54106-9999 [ACK] Seq=4271183519 Ack=208317015 Win=408288 Len=0 TSval=4717
5	0.000947	127.0.0.1	127.0.0.1	TCP	56	9999-54106 [FIN, ACK] Seq=208317015 Ack=4271183519 Win=408288 Len=0 TSval=4717
6	0.000984	127.0.0.1	127.0.0.1	TCP	56	54106-9999 [ACK] Seq=4271183519 Ack=208317016 Win=408288 Len=0 TSval=4717
7	0.001000	127.0.0.1	127.0.0.1	TCP	56	[TCP Dup ACK 5#1] 9999-54106 [ACK] Seq=208317016 Ack=4271183519 Win=408288
8	0.001017	127.0.0.1	127.0.0.1	TCP	56	54106-9999 [FIN, ACK] Seq=4271183519 Ack=208317016 Win=408288 Len=0 TSval=4717
9	0.001047	127.0.0.1	127.0.0.1	TCP	56	9999-54106 [ACK] Seq=208317016 Ack=4271183520 Win=408288 Len=0 TSval=4717

The packet details pane for packet 4 shows the following information:

- Flags: 0x010 (ACK)
- Window size value: 12759
- [Calculated window size: 408288]
- [Window size scaling factor: 32]
- Checksum: 0xfe28 [validation disabled]
- Urgent pointer: 0
- Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
- SEQ/ACK analysis
 - [iRTT: 0.000106000 seconds]
 - TCP Analysis Flags
 - [Expert Info (Chat/Sequence): TCP window update]
 - [TCP window update]
 - [Severity level: Chat]
 - [Group: Sequence]

Red arrows in the image point to the packet 4 in the list and the 'tcp.seq analysis flag' in the details pane.

Note

A Window Update is a 0-byte segment with the same SEQ/ACK numbers as the previously seen segment and with a new window value.

TCP Dup-ACK

Duplicate ACKs are sent when there is fast retransmission. In this scenario the same segment will be seen often. Open `duplicate_ack.pcapng` and apply the `tcp.analysis.duplicate_ack` filter, as shown:

Filter: `tcp.analysis.duplicate_ack` Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Window size value	Info
367	9.609008000	192.168.1.101	54.169.134.196	TCP	4096	[TCP Dup ACK 365#1] 52638-8000
1359	36.562874000	192.168.1.101	54.169.134.196	TCP	4096	[TCP Dup ACK 1357#1] 52638-8000
2051	55.205293000	192.168.1.101	54.169.134.196	TCP	4096	[TCP Dup ACK 2049#1] 52638-8000
2404	64.726134000	192.168.1.101	54.169.134.196	TCP	4096	[TCP Dup ACK 2402#1] 52638-8000

Frame# 2404

[Window size scaling factor: 32]
Checksum: 0xe72a [validation disabled]
Urgent pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps

SEQ/ACK analysis
[This is an ACK to the segment in frame: 2403]
[The RTT to ACK the segment was: 0.000071000 seconds]
[iRTT: 0.066791000 seconds]
TCP Analysis Flags
[This is a TCP duplicate ack]
[Duplicate ACK #: 1]
[Duplicate to the ACK in frame: 2402]
Expert Info (Note/Sequence): Duplicate ACK (#1)
[Duplicate ACK (#1)]
[Severity level: Note]
[Group: Sequence]

Duplicate Detection Wireshark SEQ/ACK Analysis

As you can see in the previous screenshot:

- Duplicate ACKs occur when the Window/SEQ/ACK is the same as the previous segment and if the segment length is 0
- Duplicate ACKs can occur when there is a packet loss, in which case a retransmission can be seen

References

The following references will be useful while working with TCP/IP not limited:

- RFC675 TCP/IP first specification: <https://tools.ietf.org/html/RFC675>
- RFC793 TCP v4: <https://tools.ietf.org/html/RFC793>
- TCP Wiki: https://en.wikipedia.org/wiki/Transmission_Control_Protocol
- The TCP/IP guide at: <http://www.tcpipguide.com/>
- Ask Wireshark for all Wireshark-related queries at: <https://ask.wireshark.org/>
- Display filter references for TCP at: <https://www.wireshark.org/docs/dfref/t/tcp.html>
- TCP analyze sequence numbers at:
https://wiki.wireshark.org/TCP_Analyze_Sequence_Numbers
- Helpful clips at: <https://goo.gl/lVaEc9>

Summary

In this chapter you have learnt how the TCP opens and closes its connection, and how TCP states are maintained during this period. This chapter also covered error patterns seen on networks and how to troubleshoot those scenarios.

In the next chapter we will implement deep-packet inspections of the SSL protocol.

Chapter 4. Analyzing SSL/TLS

In this chapter we will learn what SSL/TLS is used for, how the entire handshake process happens, and about the common areas where the SSL/TLS handshake fails, by covering the following topics:

- An introduction to SSL/TLS
- The SSL/TLS Handshake Protocol with Wireshark
- SSL/TLS—decrypting communication with Wireshark
- SSL/TLS—debugging handshake issues

An introduction to SSL/TLS

Transport Layer Security (TLS) is the new name for **Secure Socket Layer (SSL)**. It provides a secure transport connection between applications with the following benefits:

- SSL/TLS works on Layer 7 (the Application Layer) on behalf of the higher-level protocols
- SSL/TLS provides confidentiality and integrity by encrypting communications
- SSL/TLS allows client-side validation (optional) for closed use cases

SSL/TLS versions

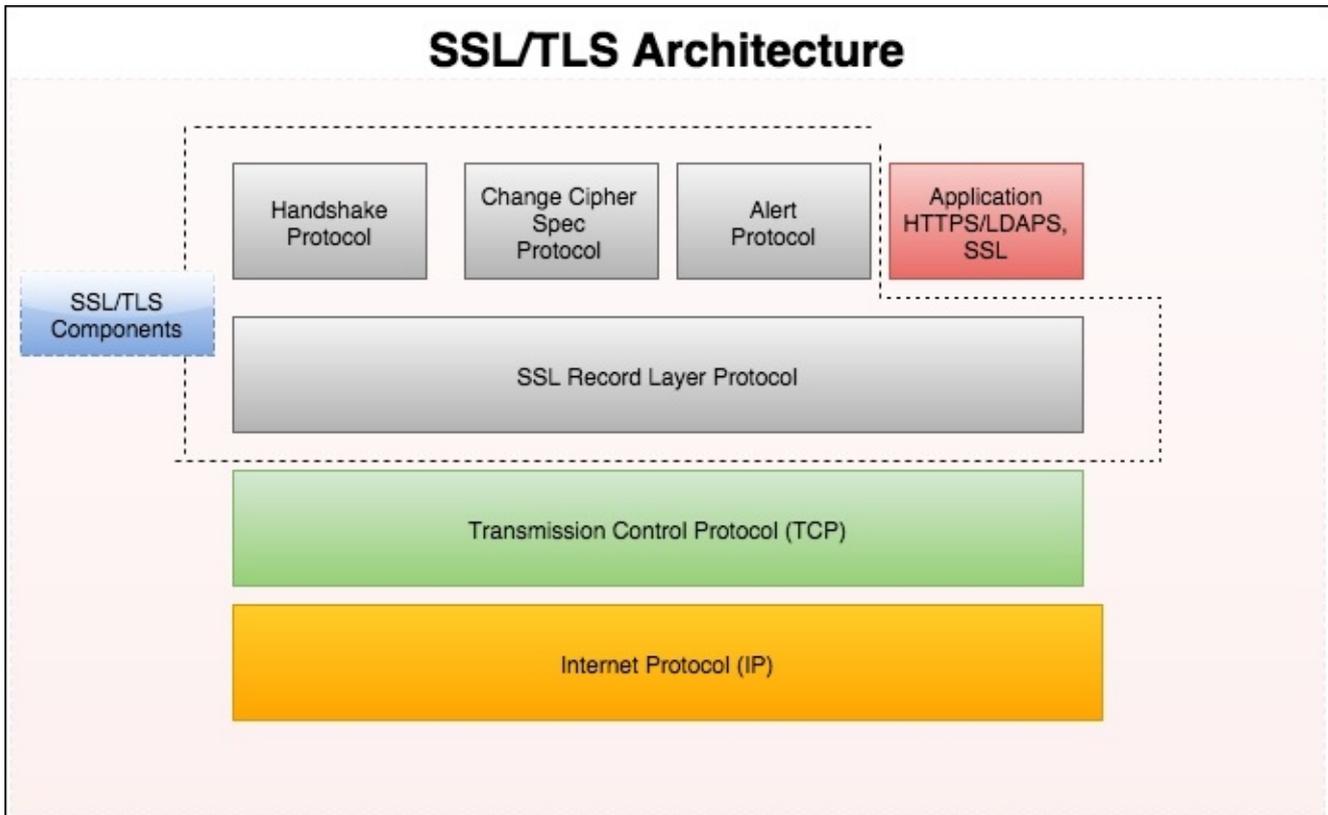
Knowing the versions is extremely important while debugging handshake issues, as most handshake failures happen in this process.

Netscape developed the original SSL versions and other versions; their RFC numbers are shown in the following table:

Protocol	Year	RFC	Deprecated
SSL 1.0	N/A	N/A	N/A
SSL 2.0	1995	NA	Y RFC 6176
SSL 3.0	1996	RFC 6101	Y RFC 7568
TLS 1.0	1999	RFC 2246	N
TLS 1.1	2006	RFC 4346	N
TLS 1.2	2008	RFC 5246	N
TLS 1.3	TBD	DRAFT	N

The SSL/TLS component

SSL/TLS is split into four major components, as shown in the following screenshot, and this chapter will cover all components in detail, one by one:



The SSL/TLS handshake

The TLS Handshake Protocol is responsible for the authentication and key exchange necessary to establish or resume a secure session. Handshake Protocol manages the following:

- Client and server will agree on cipher suite negotiation, random value exchange, and session creation/resumption
- Client and server will arrive at the pre-master secret
- Client and server will exchange their certificate to verify themselves with the client (optional)
- Generating the master secret from the pre-master secret and exchanging it

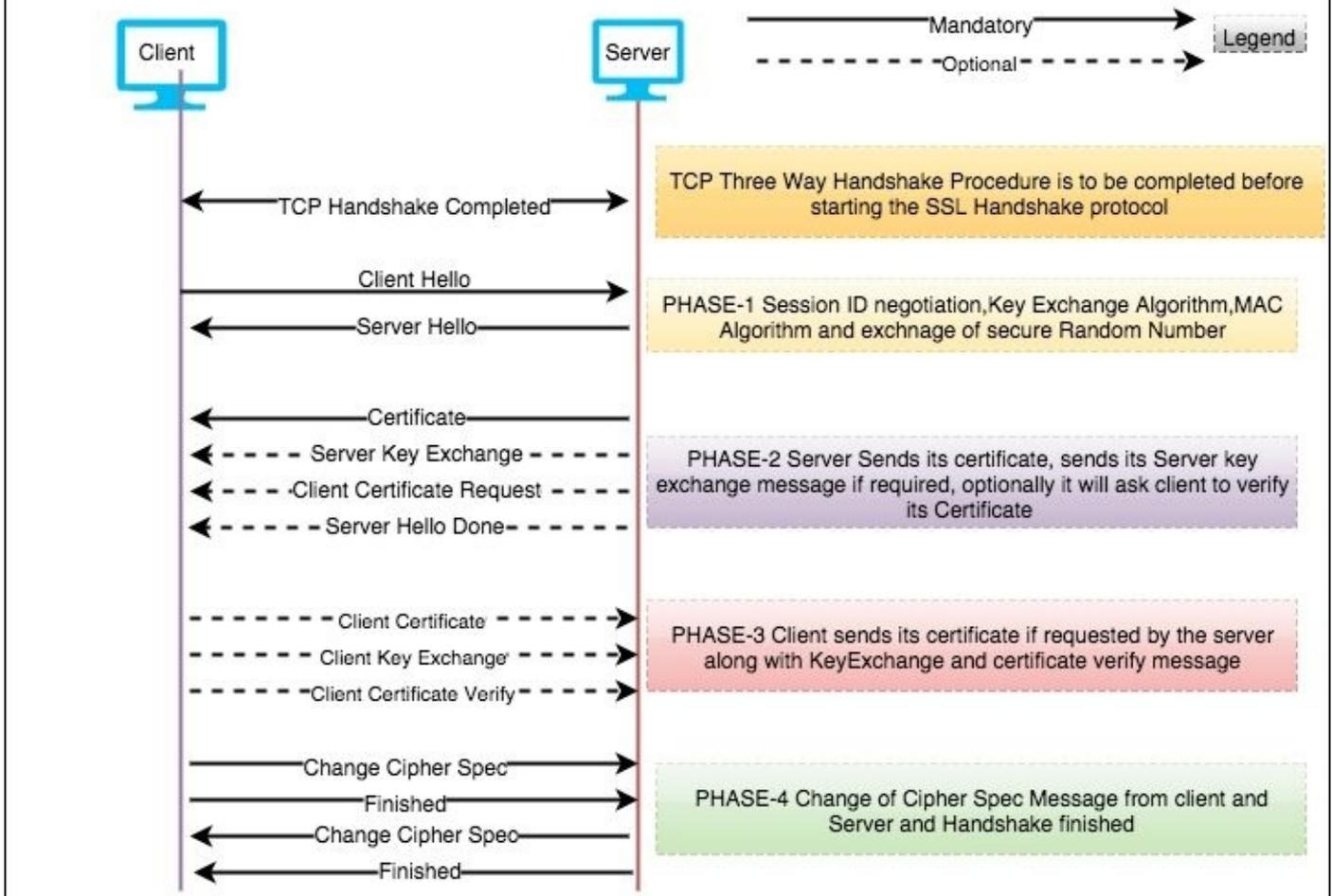
Types of handshake message

There are ten types of message, as shown in the following table, and their corresponding Wireshark filters. This is a one-byte field in the Handshake Protocol:

Type	Protocol	Message	Wireshark content type	Wireshark filter
0	Handshake	Hello request	ssl.record.content_type == 22	ssl.handshake.type == 0
1		Client Hello		ssl.handshake.type == 1
2		Server Hello		ssl.handshake.type == 2
11		Certificate		ssl.handshake.type == 11
12		ServerKeyExchange		ssl.handshake.type == 12
13		CertificateRequest		ssl.handshake.type == 13
14		ServerHelloDone		ssl.handshake.type == 14
15		Certificate Verify		ssl.handshake.type == 15
16		Client Key Exchange		ssl.handshake.type == 16
20		Finished		ssl.handshake.type == 20
	ChangeCipherSpec		ssl.record.content_type == 20	
	Application Data		ssl.record.content_type == 23	
	Alert Protocol		ssl.record.content_type == 21	

The TLS Handshake Protocol involves the following steps in four phases; the prerequisite is that a TCP connection should be established:

SSL/TLS Handshake Protocol Sequence



Open the file two-way-handshake.pcap, which is an example demonstrating a SSL mutual authentication procedure:

No.	Time	Source	Destination	Protocol	Info
4	2.139638	10.0.0.31	10.0.0.106	TLSv1.2	Client Hello
6	2.139709	10.0.0.106	10.0.0.31	TLSv1.2	Server Hello, Certificate
7	2.139721	10.0.0.106	10.0.0.31	TLSv1.2	Server Key Exchange
10	2.142678	10.0.0.31	10.0.0.106	TLSv1.2	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
11	2.143987	10.0.0.106	10.0.0.31	TLSv1.2	Change Cipher Spec, Encrypted Handshake Message
12	2.145766	10.0.0.31	10.0.0.106	TLSv1.2	Application Data
13	2.146385	10.0.0.106	10.0.0.31	TLSv1.2	Application Data
14	2.148431	10.0.0.31	10.0.0.106	TLSv1.2	Encrypted Alert

Client IP 10.0.0.31
 Server IP 10.0.0.106

SSL Mutual Authentication Example

Client Hello

The TLS handshake starts with the Client Hello message (`ssl.handshake.type == 1`), as shown in the following screenshot:

Filter: **ssl** Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Info
4	2.136636	10.0.0.31	10.0.0.106	TLSv1.2	Client Hello
6	2.139709	10.0.0.106	10.0.0.31	TLSv1.2	Server Hello, Certificate
7	2.139721	10.0.0.106	10.0.0.31	TLSv1.2	Server Key Exchange

▶ Frame 4: 339 bytes on wire (2712 bits), 339 bytes captured (2712 bits)
 ▶ Ethernet II, Src: 02:fa:c9:9c:0c:7f (02:fa:c9:9c:0c:7f), Dst: 02:e1:ed:dc:11:5d
 ▶ Internet Protocol Version 4, Src: 10.0.0.31 (10.0.0.31), Dst: 10.0.0.106 (10.0.0.106)
 ▶ Transmission Control Protocol, Src Port: 52792 (52792), Dst Port: 443 (443), Seq: 1603160316
 ▼ Secure Sockets Layer
 ▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello

Content Type: Handshake (22)
 Version: TLS 1.0 (0x0301)
 Length: 268

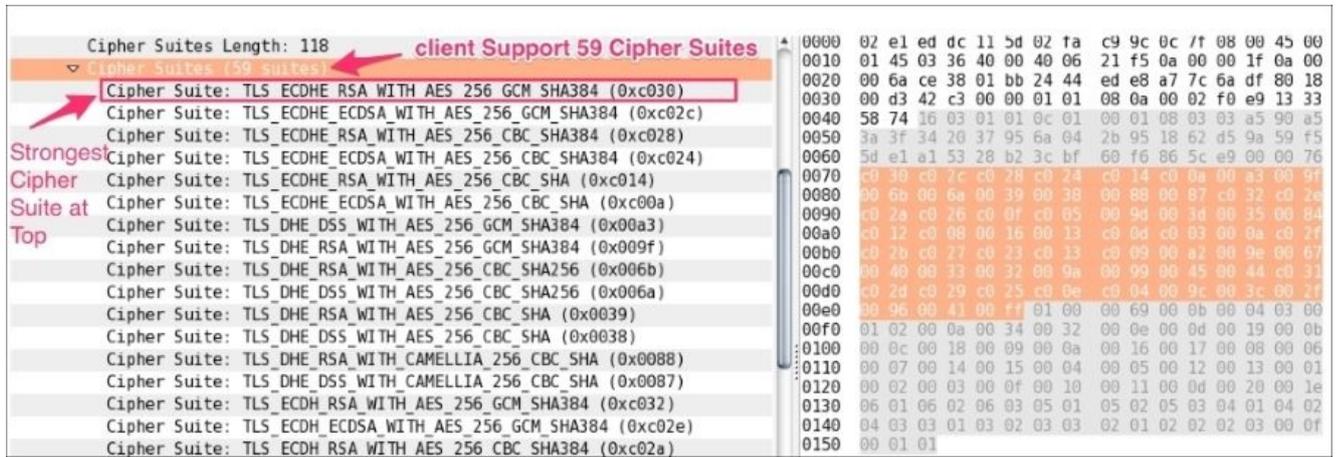
▼ Handshake Protocol: Client Hello
 Handshake Type: Client Hello (1) ← client hello Message
 Length: 264
 Version: TLS 1.2 (0x0303) ← Use TLS1.2 for SSL communication
 Random
 GMT Unix Time: Jan 8, 2058 18:25:22.000000000 IST
 Random Bytes: 3f342037956a042b951862d59a59f55de1a15328b23c9f60...
 Session ID Length: 0 ← 0 indicates no session available
 Cipher Suites Length: 118
 Cipher Suites (59 suites) ← supports total 59 Cipher Suites
 Compression Methods Length: 1
 Compression Methods (1 method)
 Extensions Length: 105
 Extension: ec_point_formats
 Extension: elliptic_curves
 Extension: signature_algorithms ← request extended functional from server
 Extension: Heartbeat

0000 02 e1 ed dc 11 5d 02 fa c9 9c 0c 7f 08 00 45 00
 0010 01 45 03 36 40 00 40 06 21 f5 0a 00 00 1f 0a 00
 0020 00 6a ce 38 01 bb 24 44 ed e8 a7 7c 6a df 80 18
 0030 00 d3 42 c3 00 00 01 01 08 0a 00 02 f0 e9 13 33
 0040 58 74 16 03 01 01 0c 01 00 01 08 03 03 a5 90 a3
 0050 3a 3f 34 20 37 95 6a 04 2b 95 18 62 d5 9a 59 f5
 0060 5d e1 a1 53 78 b2 3c bf 60 76 86 5c e9 00 00 76
 0070 c0 30 c0 2c c0 28 c0 24 c0 14 c0 0a 00 a3 00 9f
 0080 00 6b 00 6a 00 39 00 38 00 88 00 87 c0 32 c0 2e
 0090 c0 2a c0 26 c0 0f c0 05 00 9d 00 38 00 35 00 84
 00a0 c0 12 c0 08 00 16 00 13 c0 0d c0 03 00 0a c0 2f
 00b0 c0 2b c0 27 c0 23 c0 13 c0 09 00 a2 00 9e 00 67
 00c0 00 40 00 33 00 32 00 9a 00 99 00 45 00 44 c0 3f
 00d0 c0 2d c0 29 c0 25 c0 0e c0 04 00 9c 00 3c 00 2f
 00e0 00 96 00 41 00 ff 01 08 00 69 00 0b 00 04 03 00
 00f0 01 02 00 0a 00 34 00 32 00 0e 00 0d 00 19 00 0b
 0100 00 0c 00 18 00 09 00 0a 00 16 00 17 00 08 00 06
 0110 00 07 00 14 00 15 00 04 00 05 00 12 00 13 00 01
 0120 00 02 00 03 00 0f 00 18 00 11 00 0d 00 20 00 1e
 0130 06 01 06 02 06 03 05 01 05 02 05 03 04 01 04 02
 0140 04 03 03 01 03 02 03 03 02 01 02 02 02 03 00 0f
 0150 00 01 01

Client Hello Message

Handshake records are identified as hex byte 0x16=22. The structure of the Client Hello message is as follows:

- **Message:** The Client Hello message 0x01.
- **Version:** The hex byte 0x0303 means it's TLS 1.2; note 0x300 =SSL3.0.
- **Random:**
 - gmt_unix_time: The current time and date in standard UNIX 32-bit format
 - Random bytes: 28 bytes generated by the secure random number
- **Session ID:** The hex byte 0x00 shows the session ID as empty; this means no session is available and generates new security parameters.
- **Cipher suites:** The client will provide a list of supported cipher suites to the server; the first cipher suite in the list is the client-preferred (the strongest) one. The server will pick the cipher suites based on its preferences, the only condition being that the server must have client-offered cipher suites otherwise the server will raise an alert/fatal message and close the connection:



- **Compression methods:** The client will list the compression methods it supports.
- **Extensions:** The client makes use of the extension to request extended functionality from the server; in this case the client has requested four extensions, as shown in the following table:

Value	Extension name	Reference
0	elliptic_curve	RFC4492
1	ec_point_formats	RFC4492
3	signature_algorithms	RFC 5246
5	heartbeat	RFC 6520

Note

For a complete list of TLS extensions, visit: <http://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml>.

Server Hello

The server will send the Server Hello message (`ssl.handshake.type == 2`) in response to the Client Hello, as shown in the following screenshot. The message structure of the Client Hello and Server Hello message is the same, with one difference—the server can select only one cipher suite:

Filter: ssl Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Info
4	2.136636	10.0.0.31	10.0.0.106	TLSv1.2	Client Hello
6	2.139769	10.0.0.106	10.0.0.31	TLSv1.2	Server Hello, Certificate
7	2.139721	10.0.0.106	10.0.0.31	TLSv1.2	Server Key Exchange

Frame 6: 2962 bytes on wire (23696 bits), 2962 bytes captured (23696 bits)

Ethernet II, Src: 02:e1:ed:dc:11:5d (02:e1:ed:dc:11:5d), Dst: 02:fa:c9:9c:0c:7f

Internet Protocol Version 4, Src: 10.0.0.106 (10.0.0.106), Dst: 10.0.0.31 (10.0.0.31)

Transmission Control Protocol, Src Port: 443 (443), Dst Port: 52792 (52792), Seq: 3025644144, Win: 65535, Len: 2962

Secure Sockets Layer

TLSv1.2 Record Layer: Handshake Protocol: Server Hello

Content Type: Handshake (22)

Version: TLS 1.2 (0x0303)

Length: 94

Handshake Protocol: Server Hello

Handshake Type: Server Hello (2)

Length: 90

Version: TLS 1.2 (0x0303)

Random

GMT Unix Time: Jan 16, 2049 05:51:24.000000000 IST

Random Bytes: 8b9c47d8bad463dc61c9992a53a5961ecc1aeca2f0876d4c...

Session ID Length: 32

Session ID: 52392797ed9d80aa6c32937ec09c2849f6d051f041d68f73...

Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)

Compression Method: null (0)

Extensions Length: 18

Extension: renegotiation_info

Extension: ec_point_formats

Extension: Heartbeat

Handshake records are identified as hex byte 0x16=22. The structure of the Server Hello message is:

- **Handshake Type:** The hex byte 0x02=2 shows the Server Hello message
- **Version:** The hex byte 0x0303 shows TLS 1.2 has been accepted by the server

Server/client	SSLv2	SSLv3	SSLv23	TLSv1	TLSv1.1	TLSv1.2
SSLv2	Y	N	Y	N	N	N
SSLv3	N	Y	Y	N	N	N
SSLv23	N	Y	Y	Y	Y	Y
TLSv1	N	N	Y	Y	N	N
TLSv1.1	N	N	Y	N	Y	N
TLSv1.2	N	N	Y	N	N	Y

The following table shows which SSL version of the client can connect to which SSL version of the server:

- **Session ID:** A 32-byte session ID is created for reconnection purposes without a handshake
- **Cipher suite:** The server has picked Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030), which means use **Elliptic curve Diffie-Hellman (ECDHE)** key exchange, RSA for authentication, Block cipher Galois/Counter Mode (GCM), AES-256 for encryption, and SHA-384 for

digests

- **Extensions:** A response with extension info is requested in the Client Hello message

Server certificate

After the Server Hello message is sent, the server should send a X.509 server certificate (ssl.handshake.type == 11). The certificate configured on the server are signed by the CA or intermediate CA, or can be self-signed based on your deployment:

No.	Time	Source	Destination	Protocol	Info
4	2.136636	10.0.0.31	10.0.0.106	TLsv1.2	Client Hello
6	2.139709	10.0.0.106	10.0.0.31	TLsv1.2	Server Hello, Certificate
7	2.139721	10.0.0.106	10.0.0.31	TLsv1.2	Server Key Exchange
10	2.142678	10.0.0.31	10.0.0.106	TLsv1.2	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted
11	2.143987	10.0.0.106	10.0.0.31	TLsv1.2	Change Cipher Spec, Encrypted Handshake Message
12	2.145766	10.0.0.31	10.0.0.106	TLsv1.2	Application Data
13	2.146385	10.0.0.106	10.0.0.31	TLsv1.2	Application Data

Handshake Protocol: Certificate
Content Type: Handshake (22)
Version: TLS 1.2 (0x0303)
Length: 2665

Handshake type: Certificate (11)
Length: 2661
Certificates Length: 2658

Certificates (2658 bytes)
Certificate Length: 1151

Certificate (pkcs-9-at-emailAddress=zarigatongy@gmail.com, id-a)
signedCertificate
serialNumber: 1
signature (sha256WithRSAEncryption)
issuer: rdnSequence (0)
validity
subject: rdnSequence (0)
subjectPublicKeyInfo
algorithmIdentifier (sha256WithRSAEncryption)
Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
padding: 0
encrypted: 48fe9be58d30cc24dec0ff1de2b34e029db5effd80a7ae1d.
Certificate Length: 1501

Certificate (pkcs-9-at-emailAddress=zarigatongy@gmail.com, id-at

If a SSL/TLS server is configured with the certificate chain then the entire chain will be presented to the client along with the server certificate. The client (a browser or any other SSL/TLS client) can then check the highest certificate in the chain with stored CA certificates; typically, modern Web browsers have the root CA installed from the trusted CA provider.

The given certificate is signed with the relevant signature (sha256WithRSAEncryption); in this case, the hash value itself is concatenated into the OID (Algorithm Id:

1.2.840.113549.1.1.11) representing the signing algorithm. The certificate follows the DER encoding format and when encrypted becomes PKCS#7, the Cryptographic Message Syntax Standard (refer to RFC 2315).

Server Key Exchange

From RFC #5246, the server sends the Server Key Exchange message (ssl.handshake.type == 12) only when the Server Certificate message (if sent) does not contain enough data to allow the client to exchange a premaster secret:

The screenshot displays a Wireshark capture of a TLS handshake. The packet list shows the following messages:

No.	Time	Source	Destination	Protocol	Info
4	2.136636	10.0.0.31	10.0.0.106	TLSv1.2	Client Hello
6	2.139709	10.0.0.106	10.0.0.31	TLSv1.2	Server Hello, Certificate
7	2.139721	10.0.0.106	10.0.0.31	TLSv1.2	Server Key Exchange
10	2.142678	10.0.0.31	10.0.0.106	TLSv1.2	Certificate, Client Key Exchange, Certificate Verifv, Change Cipher Sock

The packet details pane shows the following structure for the Server Key Exchange message (packet 7):

- Secure Sockets Layer
 - TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
 - Content Type: Handshake (22)
 - Version: TLS 1.2 (0x0303)
 - Length: 333
 - Handshake Protocol: Server Key Exchange
 - Handshake Type: Server Key Exchange (12)
 - Length: 329
 - EC Diffie-Hellman Server Params (ECDH)

The packet details pane also shows the Multiple Handshake Messages (packet 10):

- Secure Sockets Layer
 - TLSv1.2 Record Layer: Handshake Protocol: Multiple Handshake Messages
 - Content Type: Handshake (22)
 - Version: TLS 1.2 (0x0303)
 - Length: 181
 - Handshake Protocol: Certificate Request
 - Handshake Type: Certificate Request (13)
 - Length: 173
 - Certificate types count: 3
 - Certificate types (3 types)
 - Signature Hash Algorithms Length: 30
 - Signature Hash Algorithms (15 algorithms)
 - Distinguished Names Length: 135
 - Distinguished Names (135 bytes)
 - Handshake Protocol: Server Hello Done
 - Handshake Type: Server Hello Done (14)
 - Length: 0

The packet bytes pane shows the raw hex data for the Server Key Exchange message (packet 7):

```

0000 02 fa c9 9c 0c 7f 02 e1 ed dc 11 5d 08 00 45 00
0010 01 c1 32 ae 40 00 40 06 f2 00 0a 00 00 6a 0a 00
0020 00 1f 01 bb ce 38 a7 7c 76 2f 24 44 ee f9 80 18
0030 00 eb 16 3c 00 00 01 01 08 0a 13 33 5a 8a 00 02
0040 f0 e9 02 94 1d ba 47 81 cb 30 30 9b 21 d6 04 20
0050 21 18 5e 63 20 07 81 58 09 ef a1 41 cf e4 ce df
0060 e3 00 44 cf 48 f5 83 1c f4 46 fe 9a 79 80 d9 39
0070 fa 34 c3 6a 02 8c 97 e9 e2 43 70 7c d4 2f 84 d3
0080 ef 37 e1 98 f2 d1 73 5d a4 0e fe f5 54 c4 7d 7a
0090 f6 67 d6 96 80 74 1f 79 23 09 46 8a 42 92 16 fd
00a0 a5 5b 36 a8 75 68 39 c2 2c b1 d7 7d 41 2d 44 eb
00b0 70 31 08 b6 d0 3d 40 59 3d ab 62 d6 20 94 31 4c
00c0 55 3c ec 55 cc 30 92 5c 79 9c f6 70 bd c2 98 7e
00d0 00 ac
00e0 12 fe c4 68 e4 3e 28 2d 7b a9 b3 c0 79 96 be 57
00f0 ea 4a 64 ee 06 4b fa ba 70 86 cb 85 84 be 91 d8
0100 bf 57 88 8d 9d 30 1e 70 b7 9e f0 33 b2 c0 60 90
0110 f3 a7 47 3b 6d 18 03 03 08 05 09 00 00 48 03 01
0120 01 40 04 1e 06 01 06 02 06 03 05 01 05 02 05 03
0130 04 01 04 02 04 03 03 01 03 02 03 03 02 01 02 02
0140 02 03 00 07 00 85 30 81 82 31 0b 30 09 06 03 55
0150 04 06 13 02 41 55 11 13 30 11 06 03 55 04 08 0c
0160 0a 53 6f 60 65 2d 53 74 61 74 65 31 21 30 1f 9e
0170 03 55 04 0a 0c 18 49 6e 74 65 72 6e 65 74 20 57
0180 09 64 67 69 74 73 20 50 74 79 20 4c 74 64 31 16
0190 30 14 06 03 55 04 03 0c 00 69 70 2d 31 30 2d 30
01a0 2d 30 2d 31 30 36 31 23 30 21 06 09 2a 06 48 86
01b0 f7 0d 01 09 01 16 14 7a 61 72 69 67 61 74 6f 6e
01c0 07 40 67 6e 61 69 6c 7e 63 6f 6e 0e 00 00 00
  
```

As you can see in the preceding screenshot:

- Cipher suites contains key exchange algorithms
- The Server Key Exchange message will be sent for the following key exchange methods: DHE_DSS, DHE_RSA, DH_anon
- In line with RFC#5246, the use of Server Key Exchange is not legal for these key exchange methods: RSA, DH_DSS, DH_RSA

Client certificate request

The server can optionally ask client to verify its certificate. To support mutual authentication, the server will send the certificate request message (`ssl.handshake.type == 13`) to the client and the client must provide its certificate information to the server. If the client fails to provide it, an Alert protocol will be generated and the connection will terminate.

Server Hello Done

The Server Hello Done message means that the server is done sending messages to support the key exchange, and the client can proceed with its phase of the key exchange:

Filter: ssl Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Info
4	2.136636	10.0.0.31	10.0.0.106	TLSv1.2	Client Hello
6	2.139709	10.0.0.106	10.0.0.31	TLSv1.2	Server Hello, Certificate
7	2.139721	10.0.0.106	10.0.0.31	TLSv1.2	Server Key Exchange
10	2.142678	10.0.0.31	10.0.0.106	TLSv1.2	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
11	2.143987	10.0.0.106	10.0.0.31	TLSv1.2	Change Cipher Spec, Encrypted Handshake Message
12	2.145766	10.0.0.31	10.0.0.106	TLSv1.2	Application Data
13	2.146385	10.0.0.106	10.0.0.31	TLSv1.2	Application Data
14	2.148431	10.0.0.31	10.0.0.106	TLSv1.2	Encrypted Alert

Client-Key Exchange

Frame 10: 1327 bytes on wire (10616 bits), 1327 bytes captured (10616 bits)

Ethernet II, Src: 02:fa:c9:9c:0c:7f (02:fa:c9:9c:0c:7f), Dst: 02:e1:ed:dc:11:5d

Internet Protocol Version 4, Src: 10.0.0.31 (10.0.0.31), Dst: 10.0.0.106 (10.0.0.106)

Transmission Control Protocol, Src Port: 52792 (52792), Dst Port: 443 (443), Seq: 304414400, Win: 65535, Len: 1327

Secure Sockets Layer

TLSv1.2 Record Layer: Handshake Protocol: Certificate

TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange

Content Type: Handshake (22) pre-master-secret set by exchange of ECDH param

Version: TLS 1.2 (0x0303)

Length: 70

Handshake Protocol: Client Key Exchange

Handshake Type: Client Key Exchange (16)

Length: 66

EC Diffie-Hellman Client Params

Pubkey Length: 65

Pubkey: 04d564b6417a18e1a38d0bec8f7d8bb6b862d12d7fd4d041...

sending public key to server to arrive at pre-master secret

TLSv1.2 Record Layer: Handshake Protocol: Certificate Verify

TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec

TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message

Client Certificate Verify

The Client Certificate Verify message will be sent after the Client Key Exchange message (ssl.handshake.type == 16) using master_secret generated by pre_master_secret.

Change Cipher Spec

The Change Cipher Spec record type (ssl.record.content_type == 20) is different from the handshake record type (ssl.record.content_type == 22) and it's a part of the Change Cipher Spec protocol. The Change Cipher Spec message is sent by both the client and server only when key_exchange is completed and it indicates to the receiving party that subsequent records will be protected under the newly negotiated Change Cipher Spec and keys (master_secret):

Filter: Expression... Clear Apply Save

Time	Source	Destination	Protocol	Info
7	10.0.0.106	10.0.0.31	TLSv1.2	Continuation Data
8	10.0.0.31	10.0.0.106	TCP	52792->443 [ACK] Seq=608497401 Ack=2809950343 Win=29824 Len=0 TSval=192746 TSecr=192746
9	10.0.0.31	10.0.0.106	TCP	52792->443 [ACK] Seq=608497401 Ack=2809952188 Win=33536 Len=0 TSval=192746 TSecr=192746
10	10.0.0.31	10.0.0.106	TLSv1.2	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
11	10.0.0.106	10.0.0.31	TLSv1.2	Change Cipher Spec, Encrypted Handshake Message
12	10.0.0.31	10.0.0.106	TLSv1.2	Application Data
13	10.0.0.106	10.0.0.31	TLSv1.2	Application Data
14	10.0.0.31	10.0.0.106	TLSv1.2	Encrypted Alert
15	10.0.0.31	10.0.0.106	TCP	52792->443 [FIN, ACK] Seq=608498796 Ack=2809952614 Win=36480 Len=0 TSval=192746 TSecr=192746

Length: 66
 EC Diffie-Hellman Client Params
 TLSv1.2 Record Layer: Handshake Protocol: Certificate Verify
 Content Type: Handshake (22)
 Version: TLS 1.2 (0x0303)
 Length: 136
 Handshake Protocol: Certificate Verify
 Handshake Type: Certificate Verify (15)
 Length: 132
 Signature with client's private key
 TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
 Content Type: Change Cipher Spec (20)
 Version: TLS 1.2 (0x0303)
 Length: 1
 Change Cipher Spec Message
 TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
 Content Type: Handshake (22)
 Version: TLS 1.2 (0x0303)
 Length: 40
 Handshake Protocol: Encrypted Handshake Message

Change Cipher Spec Protocol

Finished

The Finished (`ssl.record.content_type == 22`) message is encrypted so it will be an **encrypted handshake message** in Wireshark. This message is sent immediately after a Change Cipher Spec message from both the client and server to verify that the key exchange and authentication processes were successful. This message contains the MD5 hash + SHA hash. When both the client and server have sent the Finished message, the TLS handshake is considered to have finished successfully and now sending and receiving application data over the secure channel can begin:

SSL Filter

Filter: Expression... Clear Apply Save

Time	Source	Destination	Protocol	Info
4	10.0.0.31	10.0.0.106	TLSv1.2	Client Hello
6	10.0.0.106	10.0.0.31	TLSv1.2	Server Hello, Certificate
7	10.0.0.106	10.0.0.31	TLSv1.2	Continuation Data
10	10.0.0.31	10.0.0.106	TLSv1.2	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
11	10.0.0.106	10.0.0.31	TLSv1.2	Change Cipher Spec, Encrypted Handshake Message
12	10.0.0.31	10.0.0.106	TLSv1.2	Application Data
13	10.0.0.106	10.0.0.31	TLSv1.2	Application Data
14	10.0.0.31	10.0.0.106	TLSv1.2	Encrypted Alert

Frame 10: 1327 bytes on wire (10616 bits), 1327 bytes captured (10616 bits) on interface 0
 Ethernet II, Src: 02:fa:c9:9c:0c:7f (02:fa:c9:9c:0c:7f), Dst: 02:el:ed:dc:11:5d (02:el:ed:dc:11:5d)
 Internet Protocol Version 4, Src: 10.0.0.31 (10.0.0.31), Dst: 10.0.0.106 (10.0.0.106)
 Transmission Control Protocol, Src Port: 52792 (52792), Dst Port: 443 (443), Seq: 608497401, Ack: 2809952188, Len: 1261
 Secure Sockets Layer
 TLSv1.2 Record Layer: Handshake Protocol: Certificate
 TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
 TLSv1.2 Record Layer: Handshake Protocol: Certificate Verify
 TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
 TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
 Content Type: Handshake (22)
 Version: TLS 1.2 (0x0303)
 Length: 40
 Handshake Protocol: Encrypted Handshake Message

Finished Encrypted Handshake Message

Finished will send after the change Cipher Spec by both Client and Server

Application Data

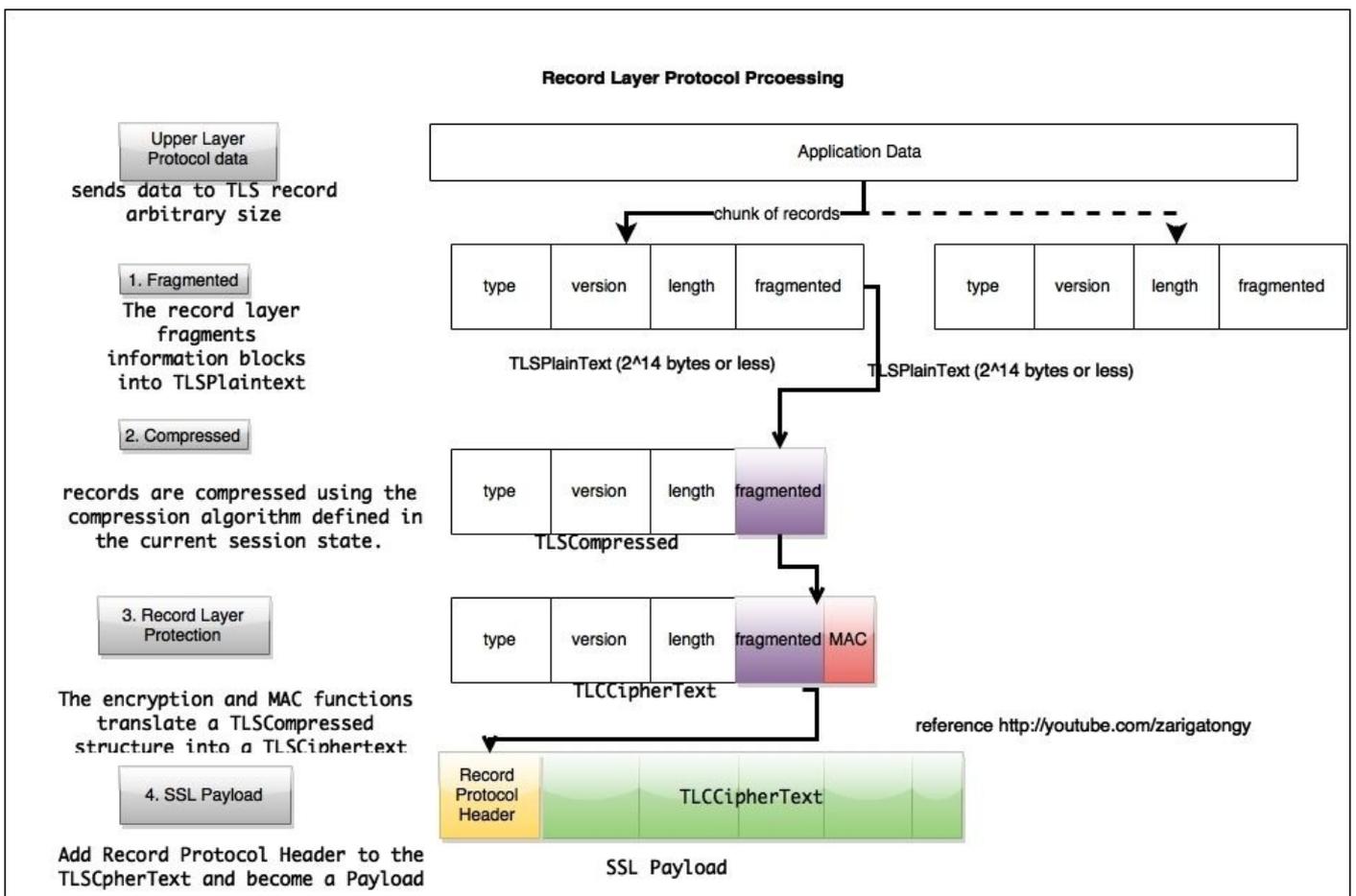
The Application Data message (ssl.record.content_type == 23) is carried by the record layer and fragmented, compressed, and encrypted:

The screenshot shows a Wireshark capture of an SSL/TLS session. The top pane displays a list of packets, with two 'Application Data' packets highlighted in red. The bottom pane shows the details of a selected packet, specifically the 'Record Layer' section. The record layer structure is as follows:

- Content Type: Application Data (23)
- Version: TLS 1.2 (0x0303)
- Length: 98
- Encrypted Application Data: a8a45892242852db16a7553864537ffbd38d3c99896817c...

The hex dump below shows the raw bytes of the record, with a red arrow pointing to the 'data exchange' field.

Record layer processing involves the mentioned step as shown in the following screenshot:



Alert Protocol

The Alert Protocol (`ssl.record.content_type == 21`) describes the severity of the message and the alert. Alert messages are encrypted and compressed and support two alert levels: warning and fatal. In the case of fatal alerts, the connection will be terminated.

Alert descriptions are shown in the following table:

Alert name	Alert type	Description
<code>close_notify(0)</code>	Closure alert	Sender will not send any more messages on this connection
<code>unexpected_message(10)</code>	Fatal	An inappropriate message was received
<code>bad_record_mac(20)</code>	Fatal	Incorrect MAC received
<code>decryption_failed(21)</code>	Fatal	TLS Cipher text decrypted in an invalid way
<code>record_overflow(22)</code>	Fatal	Message size is more than $2^{14}+2048$ bytes
<code>decompression_failure(30)</code>	Fatal	Invalid input received
<code>handshake_failure(40)</code>	Fatal	Sender unable to finalize the handshake
<code>bad_certificate(42)</code>	Fatal	Received corrupted certificate; bad ASN sequence
<code>unsupported_certificate(43)</code>	Fatal	Certificate type is not supported
<code>certificate_revoked(44)</code>	Warning	Signer has revoked the certificate
<code>certificate_expired(45)</code>	Warning	The certificate is not valid
<code>certificate_unknown(46)</code>	Warning	Certificate unknown
<code>illegal_parameter(47)</code>	Fatal	TLV contain invalid parameters
<code>unknown_ca(48)</code>	Fatal	CA chain couldn't be located
<code>access_denied(49)</code>	Fatal	Certificate is valid, the server denied the negotiation
<code>decode_error(50)</code>	Fatal	The TLV received does not have a valid form
<code>decrypt_error(51)</code>	Fatal	Decryption cipher invalid
<code>export_restriction(60)</code>	Fatal	A negotiation not in compliance with export restrictions was detected
<code>protocol_version(70)</code>	Fatal	The selected protocol version is not supported by the server
<code>insufficient_security(71)</code>	Fatal	Strong cipher suite needed
<code>internal_error(80)</code>	Fatal	Server-related issue
<code>user_canceled(90)</code>	Fatal	Client cancelled the operation
<code>no_renegotiation(100)</code>	Fatal	Server is not able to negotiate the handshake

As shown in the following screenshot, the Alert Protocol is generated by the server:

The screenshot displays a Wireshark network traffic capture with a filter set to 'ssl'. The main packet list table shows the following entries:

Time	Source	Destination	Protocol	Info
4	10.0.0.31	10.0.0.106	TLSv1.2	Client Hello
6	10.0.0.106	10.0.0.31	TLSv1.2	Server Hello, Certificate
7	10.0.0.106	10.0.0.31	TLSv1.2	Continuation Data
10	10.0.0.31	10.0.0.106	TLSv1.2	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
11	10.0.0.106	10.0.0.31	TLSv1.2	Change Cipher Spec, Encrypted Handshake Message
12	10.0.0.31	10.0.0.106	TLSv1.2	Application Data
13	10.0.0.106	10.0.0.31	TLSv1.2	Application Data
14	10.0.0.31	10.0.0.106	TLSv1.2	Encrypted Alert

Packet 14 is highlighted in red and labeled 'Alert Protocol'. Below the main list, the packet details pane shows the following structure:

- Frame 14: 97 bytes on wire (776 bits), 97 bytes captured (776 bits)
- Ethernet II, Src: 02:fa:c9:9c:0c:7f (02:fa:c9:9c:0c:7f), Dst: 02:e1:ed:dc:11:5d (02:e1:ed:dc:11:5d)
- Internet Protocol Version 4, Src: 10.0.0.31 (10.0.0.31), Dst: 10.0.0.106 (10.0.0.106)
- Transmission Control Protocol, Src Port: 52792 (52792), Dst Port: 443 (443), Seq: 608498765, Ack: 2809952614, Len: 31
- Secure Sockets Layer
 - TLSv1.2 Record Layer: Encrypted Alert
 - Content Type: Alert (21)
 - Version: TLS 1.2 (0x0303)
 - Length: 26
 - Alert Message: Encrypted Alert

The 'Alert Protocol detail Message' section is highlighted in red and labeled 'Alert Protocol detail Message'.

Key exchange

In the next section, we will talk about how the SSL/TLS channel can be decrypted; before that, we need to understand what the different keys exchange methods are and what their cipher suites look like. These are the following key exchange methods.

The Diffie-Hellman key exchange

This protocol allows two users to exchange a secret key over an insecure medium without any prior secrets; in this scheme, the example cipher suites will have a naming convention such as:

- SSL_DH_RSA_WITH_DES_CBC_SHA
- SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA

Cipher suites will have “DH” in their name, not “DHE” or “DH_anon”.

Note

You can learn more about Diffie-Hellman at: https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange.

Elliptic curve Diffie-Hellman key exchange

Elliptic curve Diffie-Hellman is a modified Diffie-Hellman exchange that uses elliptic curve cryptography instead of the traditional RSA-style large primes. **Elliptic curve cryptography (ECC)** is a public-key cryptosystem just like RSA, Rabin, and El Gamal. Some important points with this algorithm are:

- Every user has a public and a private key
- The public key is used for encryption/signature verification
- The private key is used for decryption/signature generation

Note

You can learn more about Elliptic Curve Diffie-Hellman at:

https://en.wikipedia.org/wiki/Elliptic_curve_Diffie-Hellman.

Note that the Client Hello message exchange process in the Extension elliptic_curves key exchange was offered. The example cipher suites will follow a naming convention such as:

- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA

Cipher suites will have “DHE” in their name, not “DH” or “DH_anon”.

RSA

The server's public key is made available to the client during the Server Key Exchange handshake. The `pre_master_secret` key is encrypted with the server public RSA key. The example cipher suites in this case will be:

- `SSL_RSA_WITH_RC4_128_SHA`
- `SSL_RSA_WITH_DES_CBC_SHA`
- `TLS_RSA_WITH_AES_128_CBC_SHA`

Cipher suites will have “RSA” in their name, not “DH” or “DH_anon” or “DHE”.

Decrypting SSL/TLS

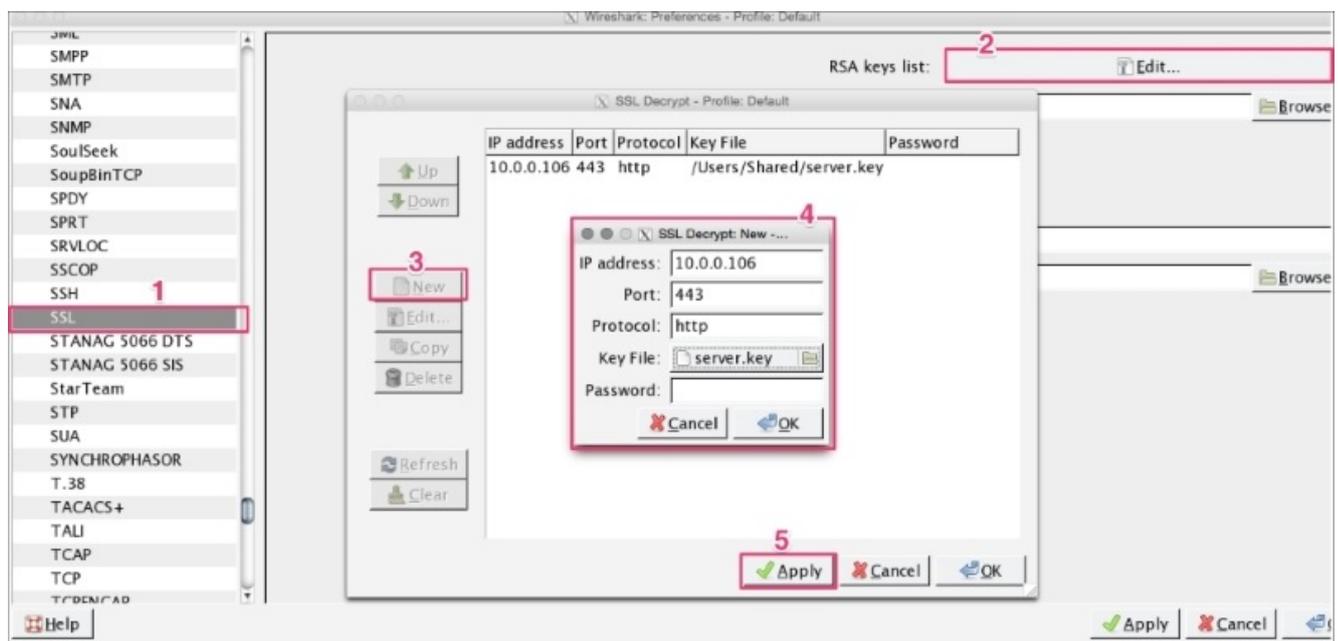
So far we have learned how the SSL/TLS protocol encrypts traffic and maintains confidentiality. In the next section, we will cover how Wireshark helps to decrypt SSL/TLS traffic.

Decrypting RSA traffic

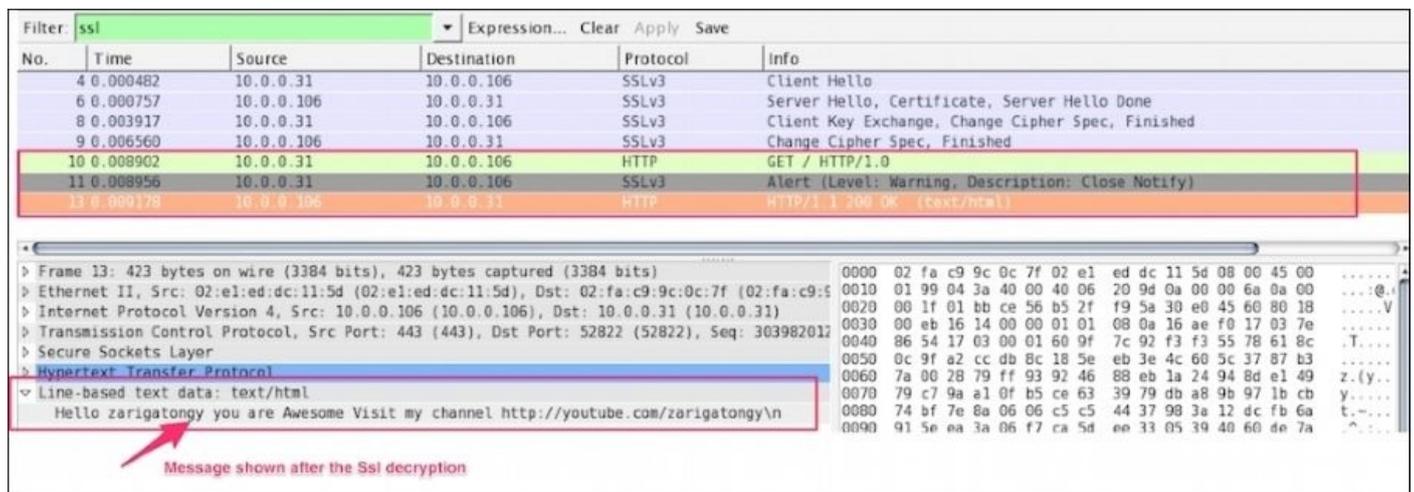
Decryption of TLS traffic depends upon which cipher suite was chosen by the server in the Server Hello message. Open the file `decrypt-ssl-01.pcap` and look for the cipher selected by the server. In this case the `TLS_RSA_WITH_AES_256_CBC_SHA` cipher suite was used; since this is RSA, we can decrypt the packet using our private key.

Now go to **Edit | Preferences | Protocol | SSL**, add the new RSA key, and configure the following properties of the RSA key dialog box:

1. The Private key file (here, `server.key`, which is used by the server).
2. The IP address of the server.
3. The port of the SSL/TLS server (443).
4. The decoding protocol—use `http` in this case.



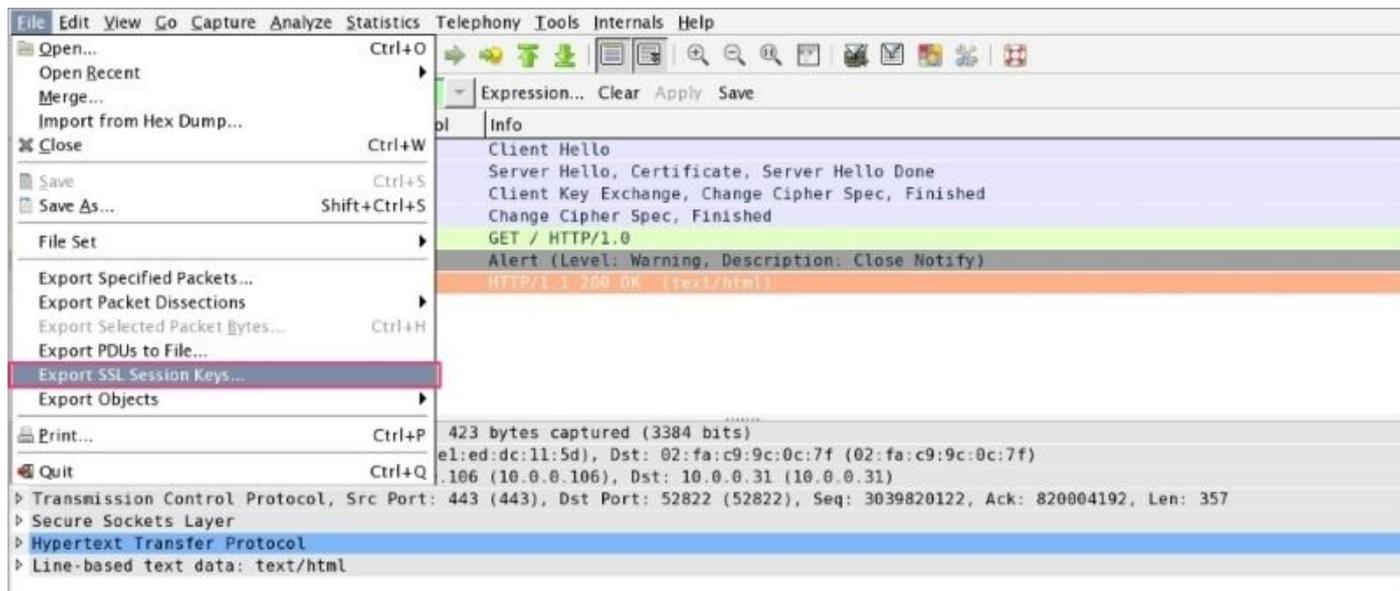
After applying these settings, the SSL traffic will be decoded into HTTP traffic for that IP, as shown in the following screenshot:



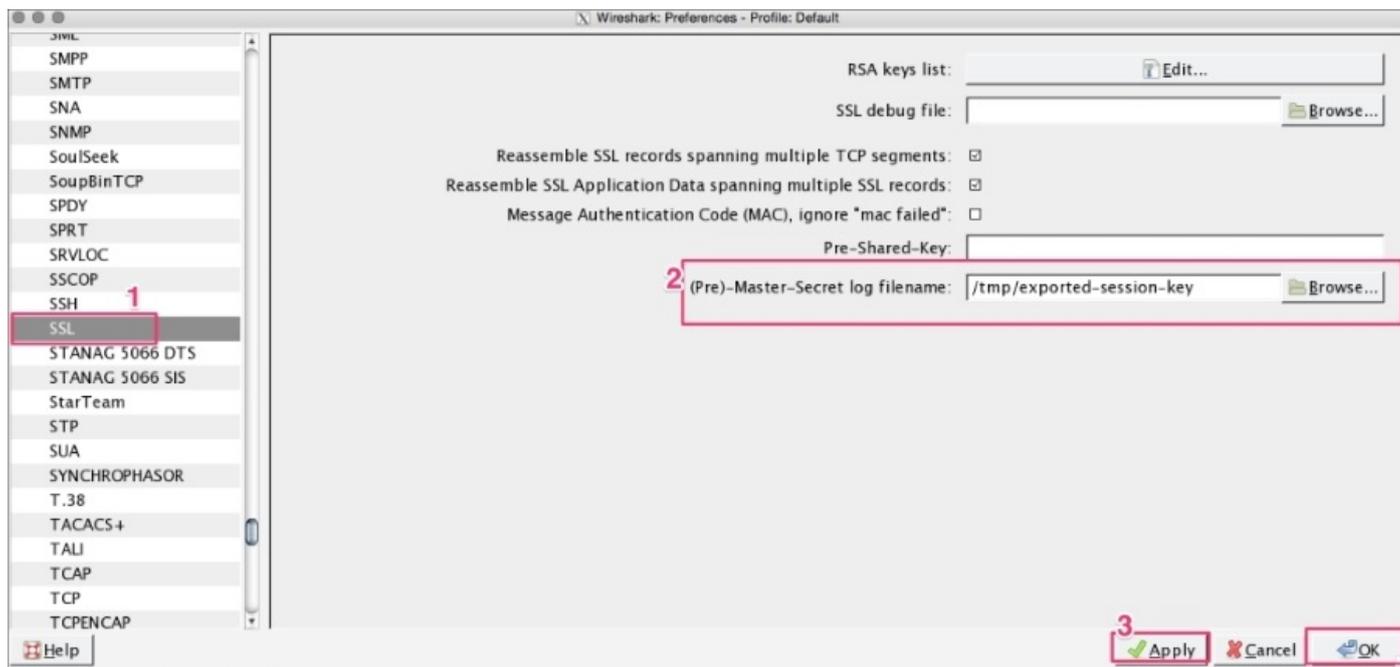
Once the packet is decrypted, the SSL session can be exported by clicking on **File | Export SSL Session Keys**. A dialog box will open; save this session key in the file (exported-session-keys). The content of the file looks like this:

RSA Session-

ID: af458c9c61675238b74f40b2a9547a0a2a394ada458a1b648e0495ed279d5e2e Master-Key: 6c970211a77548811267646a759d0d03bbc532d9b6336f2b656cb0c6bbef8f3a262d845b9abed87d26583a9c4bb9b230



Once the exported-session-keys file is created, use this file to decrypt the SSL/TLS traffic. To do so, go to **Edit | Preferences | Protocol | SSL** and configure the (Pre)-master-secret log file with the path of the SSL Session Keys. This approach is helpful when the user wants to share the packet without sharing the private keys and still needs to provide the decryption step:



Decrypting DHE/ECDHE traffic

DHE/ECDHE can't be decrypted using this approach even if we have private keys as they are designed to support forward secrecy.

Forward secrecy

Forward secrecy is supported in the **Diffie-Hellman (DHE)** and **Elliptic curve cryptography Diffie-Hellman (ECDHE)** key exchange algorithms. Take the previous scenario; the SSL/TLS communication can be decrypted by knowing the server's private key. If the private key is compromised by poor system hardening or (an internal threat agent), the SSL/TLS communication can be broken. In forward secrecy, the SSL/TLS communication is secure even if we have access to the server's private key.

If the cipher suite's name contains "ECDHE" or "DHE", it means it supports forward secrecy. For example, note this cipher suite name:

TLS_ECDHE_RSA_WITH_RC4_128_SHA.

Note

Some useful references for this are as follows:

- <http://security.stackexchange.com/questions/35639/decrypting-tls-in-wireshark-when-using-dhe-rsa-ciphersuites/42350#42350>
- <https://wiki.wireshark.org/SSL>
- <https://weakdh.org/>
- <https://www.openssl.org/docs/apps/ciphers.html>
- <https://goo.gl/9YU0HC>

Debugging issues

In the section, we will learn how to debug common SSL-related issues:

- Know your SSL/TLS server. It's very important how the server is configured, which TLS version is used, and which cipher suites it supports. To do this, use the nmap utility as shown:

```
root@bash :/home/ubuntu# nmap --script ssl-cert,ssl-enum-ciphers -p 443 10.0.0.106
Starting Nmap 6.40 ( http://nmap.org ) at 2015-08-03 16:49 UTC
Nmap scan report for ip-10-0-0-106.ap-southeast-1.compute.internal (10.0.0.106)
Host is up (0.000067s latency).
PORT      STATE SERVICE
443/tcp   open  https
| ssl-cert: Subject: commonName=ip-10-0-0-106/organizationName=Internet Widgits Pty Ltd/stateOrProvinceName=Some-State/countryName=AU
| Issuer: commonName=ip-10-0-0-106/organizationName=Internet Widgits Pty Ltd/stateOrProvinceName=Some-State/countryName=AU
| Public Key type: rsa
| Public Key bits: 2048
| Not valid before: 2015-07-28T14:43:45+00:00
| Not valid after: 2016-07-27T14:43:45+00:00
| MD5: 9ba5 0ea9 14b2 0793 7fe6 9329 08ce fab3
|_SHA-1: 1604 27b6 4f1c a838 9a9d db67 3136 88de effb f881
| ssl-enum-ciphers:
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA - strong
|     compressors:
|       NULL
|_ least strength: strong
```

- The nmap output shows the server supports TLSv1.2 and one cipher suite. If the client connects with other SSL protocols or cipher suites the server doesn't support, the server will return with handshake failure. For example, connecting the same server with TLSv1.1 will return an error:

```
rootbash # curl -k --tlsv1.1 https://10.0.0.106
curl: (35) Unknown SSL protocol error in connection to 10.0.0.106:443
```

- Connecting with ciphers the server doesn't support will return a handshake error as shown:

```
root@bash # curl -k --ciphers EXP-RC2-CBC-MD5 https://10.0.0.106
curl: (35) error:14077410:SSL routines:SSL23_GET_SERVER_HELLO:sslv3 alert handshake failure
```

- Receiving the unknown_ca error check the following find the hash value from the certificate, private key and CSR file use the following commands:

```
bash $ openssl x509 -noout -modulus -in server.crt | openssl md5
f637e8d51413ff7fa8d609e21cb27244
```

```
bash $ openssl rsa -noout -modulus -in server.key | openssl md5
f637e8d51413ff7fa8d609e21cb27244
bash $ openssl req -noout -modulus -in server.csr | openssl
f637e8d51413ff7fa8d609e21cb27244
```

The md5 hash value of csr, cer, and the private key will be the same, if csr is generated with the client private key, though the certificate is generated by using the CA (Intermediate CA) private key.

If the md5 file is the same, then verify that the certificate issued by the CA matches its path:

```
bash $ openssl verify -verbose -CAfile cacert.pem server.crt
bash $ openssl verify -verbose -CAfile cacert.pem client.crt
```

Note

Useful reference for SSL testing:

- <https://www.ssllabs.com/ssltest/>
- <https://github.com/rbsec/sslscan>
- <https://testssl.sh/openssl-rfc.mapping.html>

Summary

In this chapter, we have learned how the SSL/TLS Handshake Protocol works and how to analyze it using Wireshark. We have examined sample debugging issues related to handshakes, and learned how to solve them. In the next chapter, we will continue analyzing other application layer protocols with the help of Wireshark.

Chapter 5. Analyzing Application Layer Protocols

In the previous chapter, we covered the SSL/TLS application layer protocol in detail. In this chapter, we will continue with other application layer protocols (their basic flows and some generic use cases) and learn how to generate these types of traffic:

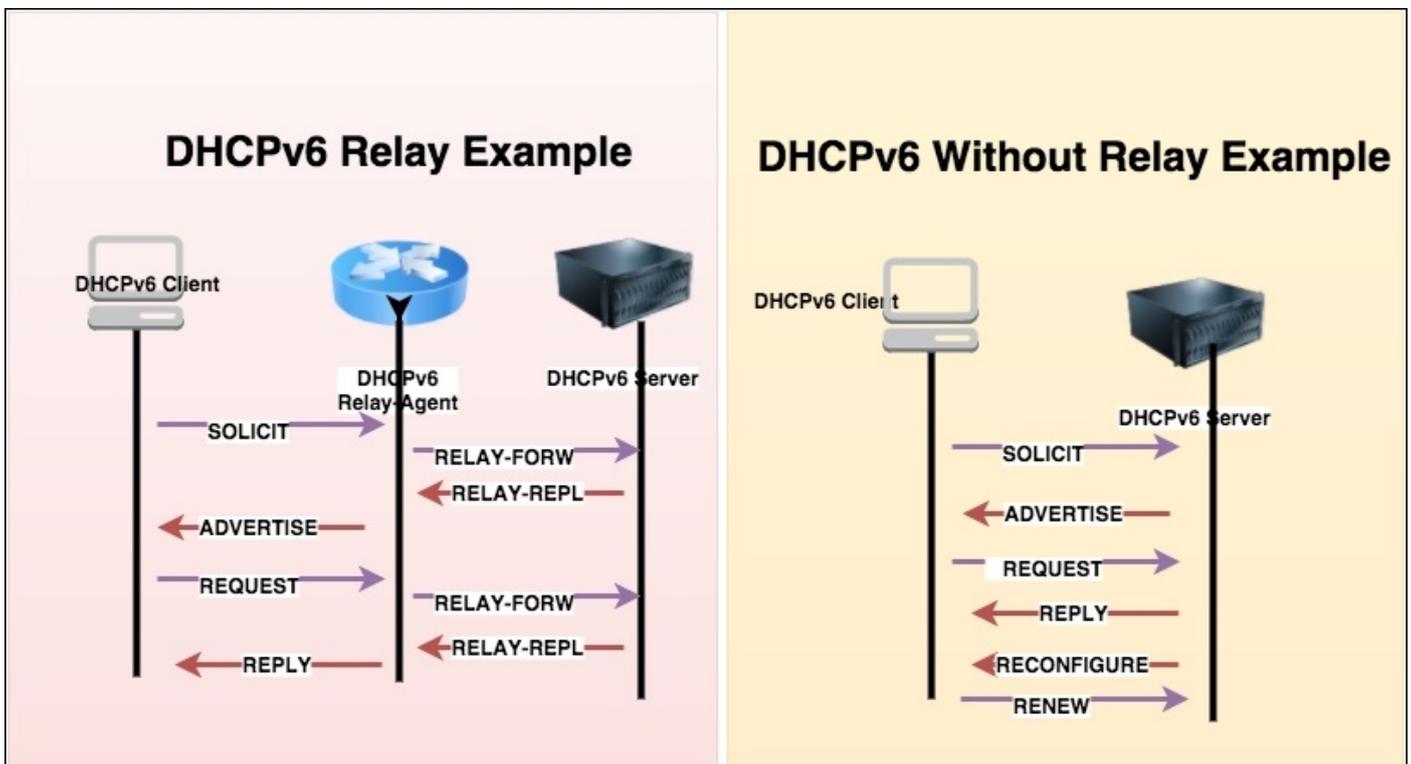
- DHCPv6
- DHCv4
- DNS
- HTTP

DHCPv6

The **Dynamic Host Configuration Protocol for IPv6 (DHCPv6)** is an application layer protocol that provides a DHCPv6 client with IPv6 an address, and other configuration information, that is carried in the DHCPv6 options.

DHCPv6 is both a Stateful Address Autoconfiguration protocol and a Stateless Address Configuration protocol.

The client and server exchange DHCPv6 message over UDP; the client uses a link-local address, DHCPv6 receives message over the link-scoped multicast address. If the DHCPv6 server is not attached to the same link, then a DHCPv6 relay agent on the client's link will relay messages between the DHCPv6 client and DHCPv6 server, as shown in the following screenshot:



DHCPv6 Wireshark filter

Use the dhcpv6 display filter to show DHCPv6 traffic. For the capturing filter, use UDP port 547.

Multicast addresses

Multicast addresses are used by the DHCPv6 client to send datagrams to a group of DHCPv6 servers:

- For all DHCP relay agents and servers, the address is FF02::1:2 (link local)
- For all DHCPv6 servers, the address is FF05::1:3 (site local)

The UDP port information

Servers and relay agents listen for DHCPv6 messages on UDP port 547; clients listen for DHCPv6 messages on UDP port 546. To find the port information, the `netstat` command can be used:

```
[root@bash ~]# netstat -an | grep 547
udp        0          0 :::547           :::*
```

DHCPv6 message types

DHCPv6 messages are exchanged over UDP port 546 and 547 and the messages are described in the following table:

DHCPv6 message	Description	DHCPv6 Wireshark filter	Equivalent DHCP for IPv4 message
SOLICIT	This message is sent by the client to a group of DHCPv6 servers	dhcpv6.msgtype == 1	DHCPDISCOVER
ADVERTISE	This message is sent by the server, and reveals the server availability for the DHCPv6 service, in response to the SOLICIT message	dhcpv6.msgtype == 2	DHCP OFFER
REQUEST	This message will be sent by the client and contains the IPv6 address or configuration parameter	dhcpv6.msgtype == 3	DHCPREQUEST
CONFIRM	This message will be sent by the client to confirm whether the IPv6 address is still valid for this link or not	dhcpv6.msgtype == 4	DHCPREQUEST
RENEW	This message will be sent by the client to update its lifetime or other configuration parameter	dhcpv6.msgtype == 5	DHCPREQUEST
REBIND	This message will be sent by the client if the RENEW message was not received, and it will update its IPv6 address and other configuration parameters	dhcpv6.msgtype == 6	DHCPREQUEST
REPLY	For every message sent by the client a REPLY message will be received from the server	dhcpv6.msgtype == 7	DHCPACK
RELEASE	This message will be sent by the client to release the IPv6 address and other configuration parameters	dhcpv6.msgtype == 8	DHCPRELEASE
DECLINE	This message will be sent by the client if it found that the IPv6 address is already assigned and in use	dhcpv6.msgtype == 9	DHCPDECLINE
RECONFIGURE	This message will be sent by the server to indicate that configuration parameters are updated or changed; the client will send a RENEW/REPLY or INFORMATION-REQUEST/REPLY to get the updated configuration	dhcpv6.msgtype == 10	N/A
INFORMATION-REQUEST	This message will be sent by the client for the configuration request no IPv6 address assignment	dhcpv6.msgtype == 11	DHCPINFORM
RELAY-FORWARD	This message will be sent by a relay agent to forward a message to a server. RELAY-FORWARD contains a client message encapsulated as the DHCPv6 RELAY message option	dhcpv6.msgtype == 12	N/A
RELAY-REPLY	This message will be sent by a server to send a message to a client through a relay agent. RELAY-REPLY contains a server message encapsulated as the DHCPv6 RELAY message option	dhcpv6.msgtype == 13	N/A

Message exchanges

DHCPv6 message exchanges happen in order to obtain the IPv6 addresses, configuration (NTP server, DNS server), or RENEW/RELEASE/DECLINE of the IPv6 address, and these message exchanges are categorized in two parts:

- Client-server with a four-message exchange
- Client-server with a two-message exchange

The four-message exchange

The acronym for a four-message exchange is **SARR**, and it is used to request the assignment of one or more IPv6 addresses. The message flow is as follows:

- SOLICIT
- ADVERTISE
- REQUEST
- REPLY

Open the DHCPv6-Flow-SOLICIT.pcap file in Wireshark, and examine the IP assignment flow as shown:

1. Wireshark filter: `dhcpv6`

2. Client always Communicate on: SOLICIT, ADVERTISE, REQUEST, REPLY

No.	Time	Source	Destination	Protocol	Info
3	0.581260	fe80::f816:3eff:fe1d:e848	ff02::1:2	DHCPv6	Solicit
4	1.595232	fe80::f816:3eff:fe1d:e848	ff02::1:2	DHCPv6	Advertise
5	1.595660	fe80::f816:3eff:fe1d:e848	ff02::1:2	DHCPv6	Request X

3. SOLICIT DHCPv6 Message Type

4. Client Request OPTION, the Advertise will have Name server Information

5. The client uses IA_NA options to request the assignment of non-temporary addresses and uses IA_TA options to request the assignment of temporary addresses

The preceding screenshot shows a SARR flow packet being captured. IPv6 is assigned to the DHCPv6 client, and the message exchanges in detail are:

- SOLICIT: The client (fe80::f816:3eff:fe1d:e848) sends a SOLICIT message to locate the servers. Note the destination is multicast ff02::1:2 not the server

(destination) IPv6 address:

- The client includes its client-identifier option `dhcpv6.option.type == 1`.
 - The client sends it ORO option (`dhcpv6.option.type == 6`) to the server that is interested in receiving. In this case, the client has requested the name server information.
 - In this example, the client uses the IA_NA options to request the assignment of non-temporary addresses (`dhcpv6.option.type == 3`) and uses IA_TA options to request the assignment of temporary addresses.
 - The client IA address option is used to specify IPv6 addresses associated with IA_NA or IA_TA. In this example, it's associated with IA_NA.
- **ADVERTISE:** The server (`fe80::f816:3eff:fe1d:e848`) sends the ADVERTISE (`dhcpv6.msgtype == 2`) message to the client (`fe80::f816:3eff:fe1d:e848`). There can be multiple servers that will respond to the client SOLICIT message; the client will choose the DHCPv6 server based on its preference:
 - The server updates the IA_NA (`dhcpv6.option.type == 3`) value based on its preferences.
 - The server includes its server identifier (`dhcpv6.option.type == 2`) information. The **Server Identifier** option is used to carry DUID. The **DUID** is the **DHCP Unique Identifier**, the host identifier in IPv6. (In the case of DHCPv4, the host identifier is the MAC address.)
 - The server includes the name server (`dhcpv6.option.type == 23`) information as requested in the SOLICIT message.
 - The server transaction ID `0x10eafe` in this case must match with the client SOLICIT transaction ID.
- **REQUEST:** In this message the client chooses one of the servers and sends a REQUEST message to the server asking for confirmed assignment of addresses and other configuration information:
 - The client (`fe80::f816:3eff:fe1d:e848`) constructs the REQUEST packet and sends it to multicast `ff02::1:2`
 - The client includes a new transaction ID: `0x3ec03e`. (random)
 - The client include server identifier information in the REQUEST packet

No.	Time	Source	Destination	Protocol	Info
2	0.581260	fe80::f816:3eff:fe1d:e848	ff02::1:2	DHCPv6	Solicit XID: 0x10eafe CID: 000100011d578a81fa163e1de848
3	0.582000	fe80::f816:3eff:fe1d:e848	fe80::f816:3eff:fe1d:e848	DHCPv6	Advertise XID: 0x10eafe IAA: 2001:ed8:77b5::b8d1:f180 CID: 000100011d578a81fa163e1de848
4	1.595232	fe80::f816:3eff:fe1d:e848	ff02::1:2	DHCPv6	Request XID: 0x3ec03e CID: 000100011d578a81fa163e1de848
5	1.595660	fe80::f816:3eff:fe1d:e848	fe80::f816:3eff:fe1d:e848	DHCPv6	Reply XID: 0x3ec03e IAA: 2001:ed8:77b5::b8d1:f180 CID: 000100011d578a81fa163e1de848

Frame 4: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits)

Linux cooked capture

Internet Protocol Version 6, Src: fe80::f816:3eff:fe1d:e848 (fe80::f816:3eff:fe1d:e848), Dst: ff02::1:2 (ff02::1:2)

User Datagram Protocol, Src Port: 546 (546), Dst Port: 547 (547)

DHCPv6

Message type: Request (3)

Transaction ID: 0x3ec03e

Client Identifier

Server Identifier

Option: Server Identifier (2)

Length: 14

Value: 000100011d5789cdfa163e1de848

DUID: 000100011d5789cdfa163e1de848

DUID Type: link-layer address plus time (1)

Hardware type: Ethernet (1)

DUID Time: Aug 7, 2015 20:52:53.000000000 IST

Link-layer address: fa:16:3e:1d:e8:48

Option Request

Elapsed time

Identity Association for Non-temporary Address

- **REPLY:** In the case of a valid REQUEST message, the server creates the bindings for that client according to the server's policy and configuration information, records the IAs and other information requested by the client, and sends a REPLY message by setting `dhcpv6.msgtype == 7`:
 - The server transaction ID `0x3ec03e` will be the same as client DHCPv6 REQUEST message transaction ID
 - The server will include the server identifier and the client identifier
 - The REPLY message will be part of a two-message exchange and a four-message exchange

The two-message exchange

The two-message exchange will be performed between client and server when IP address assignment is not required or when the DHCPv6 client wants to obtain configuration information such as a list of available DNS servers or NTP servers—for example CONFIRM-REPLY and RELEASE-REPLY. Open the sample DHCPv6-Flow-CONFIRM-RELEASE.pcap file in Wireshark, which shows that a two-message exchange was performed:

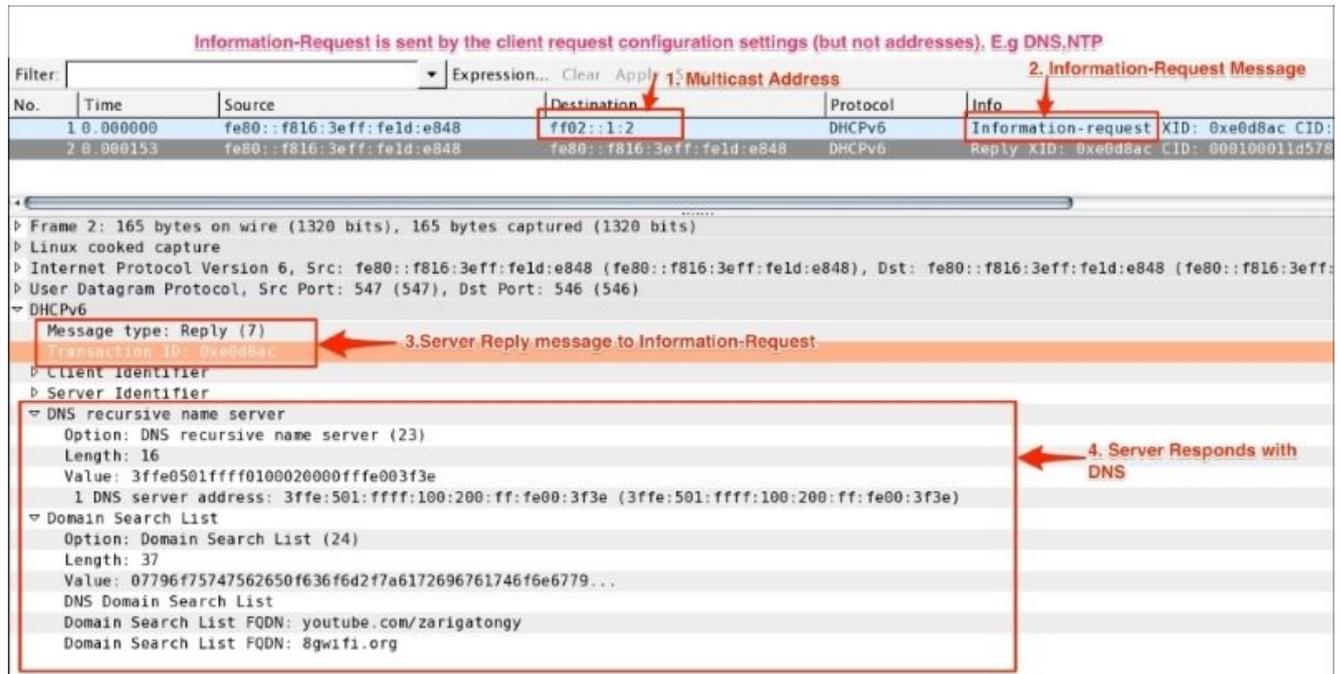
1. DHCPv6 messages CONFIRM-REPLY and RELEASE-REPLY:

No.	Time	Source	Destination	Protocol	Info
2	0.360034	fe80::f816:3eff:fe1d:e848	ff02::1:2	DHCPv6	Confirm XID: 0x38d82 CID: 000100011d578a81fa163e1de848
3	0.360471	fe80::f816:3eff:fe1d:e848	fe80::f816:3eff:fe1d:e848	DHCPv6	Reply XID: 0x38d82 CID: 000100011d578a81fa163e1de848
8	15.342561	fe80::f816:3eff:fe1d:e848	ff02::1:2	DHCPv6	Confirm XID: 0x360963 CID: 000100011d578a81fa163e1de848
9	15.342738	fe80::f816:3eff:fe1d:e848	fe80::f816:3eff:fe1d:e848	DHCPv6	Reply XID: 0x360963 CID: 000100011d578a81fa163e1de848
14	37.858625	fe80::f816:3eff:fe1d:e848	ff02::1:2	DHCPv6	Release XID: 0xd7972e CID: 000100011d578a81fa163e1de848
15	37.859183	fe80::f816:3eff:fe1d:e848	fe80::f816:3eff:fe1d:e848	DHCPv6	Reply XID: 0xd7972e CID: 000100011d578a81fa163e1de848

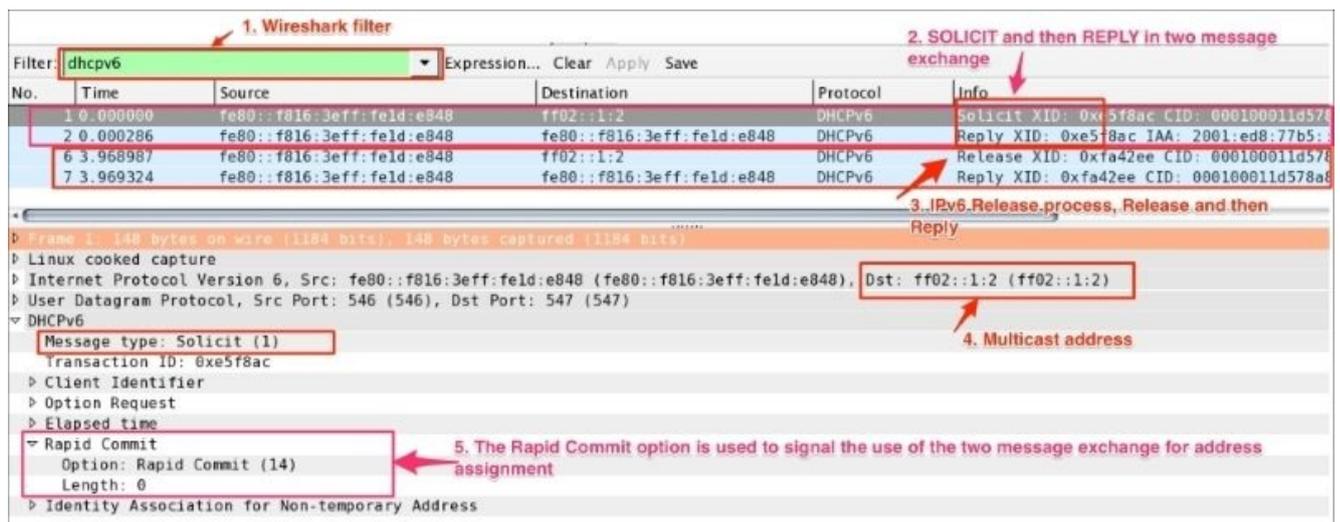
2. DHCPv6 messages INFORMATION-REQUEST: The client sends the INFORMATION-

REQUEST when the client requests configuration settings (but not addresses)—for example, DNS, NTP. As shown in the following screenshot, open the DHCPv6-Information_request.pcap file in Wireshark:

- Client will set dhcpv6.msgtype == 11:



- The rapid commit option is used to obtain the IPv6 address assignment in the two-message exchange, as shown in the following screenshot example, DHCPv6-Rapid-Commit.pcap. Note that rapid commit is not a separate DHCPv6 message and is part of the SOLICIT option:



- If a client that supports the rapid commit option intends to use the rapid commit capability, it includes a rapid commit option in the SOLICIT messages that it sends.
- If the client receives a REPLY message with a rapid commit option, it *should*

process the **REPLY** immediately (without waiting for additional **ADVERTISE** or **REPLY** messages) and use the address and configuration information contained therein.

- If the server doesn't support the rapid commit option, then it will follow with a four-message exchange (**SOLICIT**, **ADVERTISE**, **REQUEST**, and **REPLY** known as **SARR**).

DHCPv6 traffic capture

Use `dhclient` to simulate DHCPv6 traffic over the network. For this, do the following:

1. Make sure a DHCPv6 server is set up. This example is performed over an ISC **Dynamic Host Configuration Server (dhcpd)** server.
2. Run the `tcpdump` utility to capture IPv6 traffic:

```
bash$ tcpdump -i any ip6 -vv -w DHCPv6-FLOW.pcap -s0 &
```

Make sure the DHCPv6 server is running in your network.

3. To capture a DHCPv6 four-message exchange (SARR):

```
bash$ dhclient -6 eth0
```

4. To capture the DHCPv6 RELEASE message:

```
bash$ dhclient -6 -r eth0
```

5. To capture the DHCPv6 CONFIRM message:

```
bash$ dhclient -6 eth0
```

6. To capture the DHCPv6 INFORMATION request:

```
bash$ dhclient -S -6 eth0
```


BOOTP/DHCP

DHCP is an extension of the BOOTP mechanism. In other words, DHCP uses BOOTP as its transport protocol. This behavior allows existing BOOTP clients to interoperate with DHCP servers without requiring any change to the clients' initialization software; the following table shows basic comparisons between these two protocols:

BOOTP/DHCP	BOOTP	DHCP (Dynamic Host Configuration Protocol)
Meaning	Bootstrap Protocol	Dynamic Host Configuration Protocol extension of BOOTP
Year	1985	1993
UDP Server Port	67	
UDP Client port	68	
Services	<ul style="list-style-type: none"> • IPv4 address assignment • Obtaining IPv4 configuration parameter • Limited number of client configuration parameters called vendor extensions 	<ul style="list-style-type: none"> • IP address assignment • Leases • Support legacy BOOTP functionality • DHCP supports a larger and extensible set of client configuration parameters called options
RFC	RFC951	RFC 2131
Existence	Superseded by the Dynamic Host Configuration Protocol (DHCP)	ACTIVE; RFCs keep coming to add more features and support different technical requirements

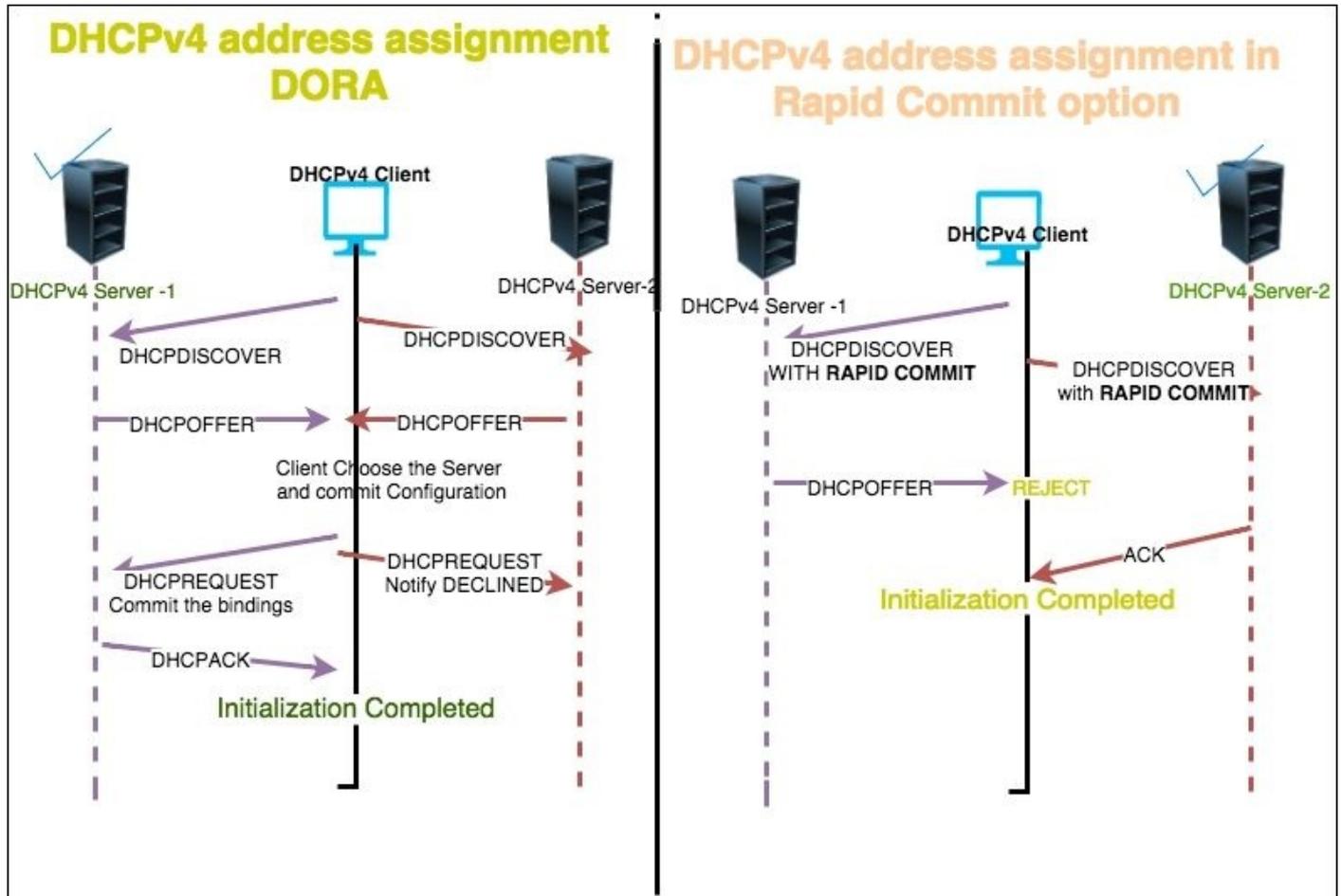
BOOTP/DHCP Wireshark filter

Use the bootp filter to display BOOTP/DHCP traffic and use UDP port 67 to capture the BOOT/DHCP traffic only.

Address assignment

DISCOVER, OFFER, REQUEST, ACK protocol exchanges happen between clients and servers during network address assignment, as shown in the following screenshot. As a mnemonic, refer to this as **DORA**.

The address assignment can also be done using the Rapid Commit option for DHCPv4. Modeled on DHCPv6, it uses two-message exchanges to quickly configure the DHCPv4 client.



To demonstrate four-message exchange open the DHCPv4.pcap file in the Wireshark, as shown in the following screenshot:

Wireshark bootp filter

Filter: **bootp** Expression... Clear Apply Save

D: Discover
O: Offer
R: Request
A: ACK

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.106	10.0.0.1	DHCP	DHCP Release - Transaction ID 0xd0faa63a
2	9.275367	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xdc7b1b3b
3	9.275746	10.0.0.1	10.0.0.106	DHCP	DHCP Offer - Transaction ID 0xdc7b1b3b
4	9.276074	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xdc7b1b3b
5	9.276297	10.0.0.1	10.0.0.106	DHCP	DHCP ACK - Transaction ID 0xdc7b1b3b

Frame 2: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits)

Ethernet II, Src: 02:e1:ed:dc:11:5d (02:e1:ed:dc:11:5d), Dst: Broadcast (ff:ff:ff:ff:ff:ff)

Internet Protocol Version 4, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)

User Datagram Protocol, Src Port: 68 (68), Dst Port: 67 (67) ← **Default server port is 67**

Bootstrap Protocol (Discover)

Message type: Boot Request (1)
Hardware type: Ethernet (0x01)
Hardware address length: 6
Hops: 0
Transaction ID: 0xdc7b1b3b
Seconds elapsed: 0

Bootp flags: 0x0000 (Unicast)

Client IP address: 0.0.0.0 (0.0.0.0)
Your (client) IP address: 0.0.0.0 (0.0.0.0)
Next server IP address: 0.0.0.0 (0.0.0.0)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: 02:e1:ed:dc:11:5d (02:e1:ed:dc:11:5d)
Client hardware address padding: 00000000000000000000
Server host name not given
Boot file name not given
Magic cookie: DHCP

Option: (53) DHCP Message Type (Discover) ← **DISCOVER Message**

Option: (50) Requested IP Address
Option: (12) Host Name
Option: (55) Parameter Request List

DHCPv4

The preceding figure shows a message exchange happening between the DHCPv4 client and DHCPv4 server. This is summarized as follows:

- DISCOVER (bootp.option.dhcp == 1):
 - Expand Bootstrap protocol to view BOOTP details
 - The client broadcasts (255.255.255.255), a DHCPDISCOVER message, on its local physical subnet and may include the option: (55 that is bootp.option.type) parameter request list; during this time the “yiaddr” field will be (bootp.ip.your == 0.0.0.0)
- OFFER (bootp.option.dhcp == 2):
 - Expand Bootstrap protocol to view BOOTP details
 - The DHCP server may respond with a DHCPOFFER message that includes an available network address in the “yiaddr” (bootp.ip.your == 10.0.0.106) field
 - The DHCP server will send its option 54: DHCP server identifier and may include the other configuration parameter as requested in option 55 the DISCOVER phase
- DHCPREQUEST (bootp.option.dhcp == 3):
 - Expand Bootstrap protocol to view BOOTP details

- The client broadcasts (255.255.255.255) a DHCPREQUEST message that *must* include the option 54 DHCP server identifier to indicate which server it has selected, and may include other options specifying the desired configuration values
- The DHCP server selected in the DHCPREQUEST message commits the binding for the client to the db storage and responds with an ACK
- ACK (bootp.option.dhcp == 5):
 - Expand Bootstrap protocol to view BOOTP details
 - The server will send the ACK to the client with the configuration parameter; during this time the IPv4 address will be “yiaddr” (bootp.ip.your == 10.0.0.106)
 - The client will verify the obtained configuration and check the IPv4 address again using the ARP protocol; if the address is in use by other dhcp clients, the client will send a DECLINE message to the server and restart the configuration process

Capture DHCPv4 traffic

The commands to capture DHCPv4 traffic are as follows:

- On a Windows machine:
 1. Start a Wireshark capture.
 2. Open the Command Prompt.
 3. Type `ipconfig /renew` and press *Enter*.
 4. Type `ipconfig /release` and press *Enter*.
 5. Stop the Wireshark capture.

- On a Linux machine:
 1. Start a Wireshark capture.
 2. Open the Command Prompt.
 3. Bring down the network interface:

```
bash# ifdown eth0:0
```
 4. Bring up the network interface:

```
bash$ ifup eth0:0
```
 5. Stop the Wireshark capture.

- Using dhclient:
 1. Start a Wireshark capture.
 2. Open the Command Prompt.
 3. To capture a DORA packet use:

```
bash$dhclient -4 eth0
```
 4. Stop the capture.

DNS

DNS stands for **Domain Name System**. DNS is used by all machines to translate hostnames into IP addresses. This mechanism is used to translate names to attributes such as addresses (IPv4/IPv6) based on the query type.

DNS has three major components:

- A name space
- Servers making that name space available
- Resolvers (clients) that query the servers about the name space

This topic will focus on the resolver perspective, where the client sends a query to the server and the server answers the query. There can be multiple answers to the same query.

DNS Wireshark filter

Wireshark's dns filter is used to display only DNS traffic, and UDP port 53 is used to capture DNS traffic.

Port

The default DNS port is 53, and it uses the UDP protocol. Some DNS systems use the TCP protocol also. TCP is used when the response data size exceeds 512 bytes, or for tasks such as zone transfers.

Resource records

The following format is used by the DNS system:

Field	Description	Length	Wireshark filter
NAME	The owner name	variable	dns.qry.name == "google.com"
TYPE	Type of Resource Record (RR) in numeric form	2	dns.qry.type == 1 (A Record Type) dns.qry.type == 255 (ANY Record Type) dns.qry.type == 2 (NS name server) dns.qry.type == 15(MX mail exchange) dns.qry.type == 28 (AAAA quad A, Ipv6 record Type)
CLASS	Class code	2	dns.qry.class == 0x0001 (IN set to internet)
TTL	Time to live	4	
RDLENGTH	Length in octets of the RDATA field	2	
RDATA	Additional RRspecific data	Variable	

DNS traffic

In this chapter, the `dig` and `nslookup` network commands are used to query the DNS server. Open the sample DNS-Packet .pcap file, set the display filter to `dns.qry.type==28`, and examine the query.

In this example, client (192.168.1.101) is asking the name server (8.8.4.4) to resolve `ipv6.google.com` by setting these parameters in the query section:

- The client sets the record type AAAA record
- The client sets the hostname (`ipv6.google.com`)
- Client set the class (that is, IN (Internet))
- The name server (8.8.4.4) responds to the client with multiple answers
- `ipv6.google.com` is the canonical name that equals `ipv6.l.google.com`
- `ipv6.l.google.com` has the AAAA address `2404:6800:4007:805::200e`

The image shows a Wireshark packet capture of a DNS transaction. The filter is set to `dns.qry.type == 28`. The packet list shows two packets: a Standard query (No. 28) from 192.168.1.101 to 8.8.4.4, and a Standard query response (No. 29) from 8.8.4.4 to 192.168.1.101. The packet details pane shows the response packet with the following structure:

- Domain Name System (response)
- Transaction ID: 0x90d7
- Flags: 0x8180 Standard query response, No error
- Questions: 1
- Answer RRs: 2
- Authority RRs: 0
- Additional RRs: 0
- Queries
 - ipv6.google.com: type AAAA, class IN
 - Name: ipv6.google.com
 - [Name Length: 15]
 - [Label Count: 3]
 - Type: AAAA (IPv6 Address) (28)
 - Class: IN (0x0001)
- Answers
 - ipv6.google.com: type CNAME, class IN, cname ipv6.l.google.com
 - Name: ipv6.google.com
 - Type: CNAME (Canonical NAME for an alias) (5)
 - Class: IN (0x0001)
 - Time to live: 21599
 - Data length: 9
 - CNAME: ipv6.l.google.com
 - ipv6.l.google.com: type AAAA, class IN, addr 2404:6800:4007:805::200e
 - Name: ipv6.l.google.com
 - Type: AAAA (IPv6 Address) (28)

User can use the popular `dig` or `nslookup` network utility commands to query different DNS record types. Use a network capture in the background and observe the query and answer section for each command:

- Query a record type used to show the IPv4 address of the given hostname:

```
bash# nslookup google.com
bash# dig google.com
bash# dig A +noadditional +noquestion +nocomments +nocmd +nostats
google.com. @8.8.4.4
```

- Query the AXFR record type; AXFR is used to transfer zone files from the master to the secondary name server:

```
bash# nslookup -type=axfr google.com 8.8.4.4
bash# dig AXFR +noadditional +noquestion +nocomments +nocmd +nostats
+multiline google.com. @8.8.4.4
```

- Query the CNAME record type. CNAME is used to set up the alias:

```
bash# nslookup -type=cname google.com 8.8.4.4
bash# dig CNAME +noadditional +noquestion +nocomments +nocmd +nostats
google.com. @8.8.4.4
```

- Query the MX record type; MX is the mail exchange record:

```
bash# nslookup -type=mx google.com 8.8.4.4
bash# dig MX +noadditional +noquestion +nocomments +nocmd +nostats
google.com. @8.8.4.4
```

- Query the NS record type; NS is the name server record:

```
bash# nslookup -type=ns google.com 8.8.4.4
bash# dig NS +noadditional +noquestion +nocomments +nocmd +nostats
google.com. @8.8.4.4
```

- Query the PTR record type; PTR is the pointer used for reverse DNS lookups:

```
bash# nslookup -type=ptr google.com 8.8.4.4
bash# dig PTR +noadditional +noquestion +nocomments +nocmd +nostats
google.com. @8.8.4.4
```

- Query the SOA record type. SOA is used to provide authoritative information such as nameserver and e-mail:

```
bash# nslookup -type=soa google.com 8.8.4.4
bash# dig SOA +noadditional +noquestion +nocomments +nocmd +nostats
+multiline google.com. @8.8.4.4
```

- Query the TXT record type; this refers to the text record:

```
bash# nslookup -type=txt google.com 8.8.4.4
bash# dig TXT +noadditional +noquestion +nocomments +nocmd +nostats
google.com. @8.8.4.4
```

- Query AAAA (also referred to as the quad-A record type); this will display the IPv6 address of the given hostname:

```
bash# nslookup -type=aaaa google.com 8.8.4.4
bash# nslookup -type=aaaa ipv6.google.com 8.8.4.4
bash# dig AAAA +noadditional +noquestion +nocomments +nocmd +nostats
ipv6.google.com. @8.8.4.4
```

- Query the ANY record type; this returns all record types:

```
bash# nslookup -type=any google.com 8.8.4.4
bash# dig ANY +noadditional +noquestion +nocomments +nocmd +nostats
google.com. @8.8.4.4
```


HTTP

HTTP is an application layer protocol used in WWW. HTTP enables communications between the HTTP client and HTTP server. Example traffic is shown in the following screenshot. An HTTP GET request is created by the client (browser or cURL), and the HTTP server has responded with the appropriate content type:

1. Wireshark filter

Filter: tcp.stream eq 2

No.	Time	Source
21	19.118828	10.0.0.221
22	19.118918	10.0.0.221

Frame 21: 83 bytes on wire (664 bits)
Ethernet II, Src: 06:3c:0f:39:2e:f7
Internet Protocol Version 4, Src: 10.0.0.221
Transmission Control Protocol, Src Port: 53736
Hypertext Transfer Protocol

http_01.pcap

Stream Content

GET / HTTP/1.1
Host: 52.74.246.190:8000
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.124 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.7.6
Date: Sun, 21 Jun 2015 17:49:36 GMT
Content-type: text/html; charset=UTF-8
Content-Length: 828

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>

Entire conversation (1340 bytes)

Find Save As Print ASCII EBCDIC Hex Dump C Arrays Raw

Help Filter Out This Stream Close

2. HTTP Request

3. HTTP Response

HTTP Wireshark filter

Use `http` to display HTTP packets only. Use TCP port 80 to filter for HTTP traffic only; port 80 is the default HTTP port.

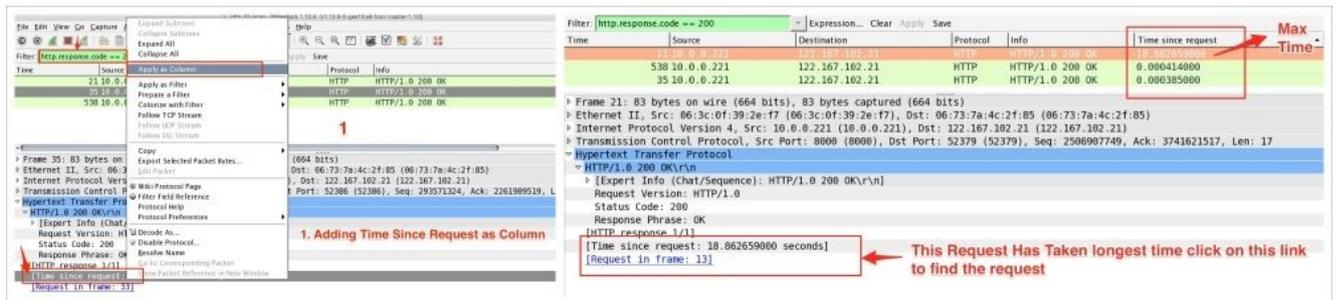
HTTP use cases

The following example shows different use cases where Wireshark can help to analyze HTTP packets.

Finding the top HTTP response time

Open the file `http_01.pcap` in the Wireshark, and find the top HTTP response time for the request HTTP get:

1. Click on **Edit | Preferences | Protocols | TCP**, uncheck **Allow subdissector to reassemble TCP streams**. This will help in knowing how many continuation packets there are to get the actual content and it will help in fine-tuning TCP parameters—for example, setting up the TCP window size to reduce the continuation packet.
2. In the **Filter** bar, apply the `http` filter and add `http.time` as a column from the `http.response.code == 200 HTTP OK` packet.
3. Click on the **Time since request** column and make it in descending order. Find the request frame and click on the link.



Finding packets based on HTTP methods

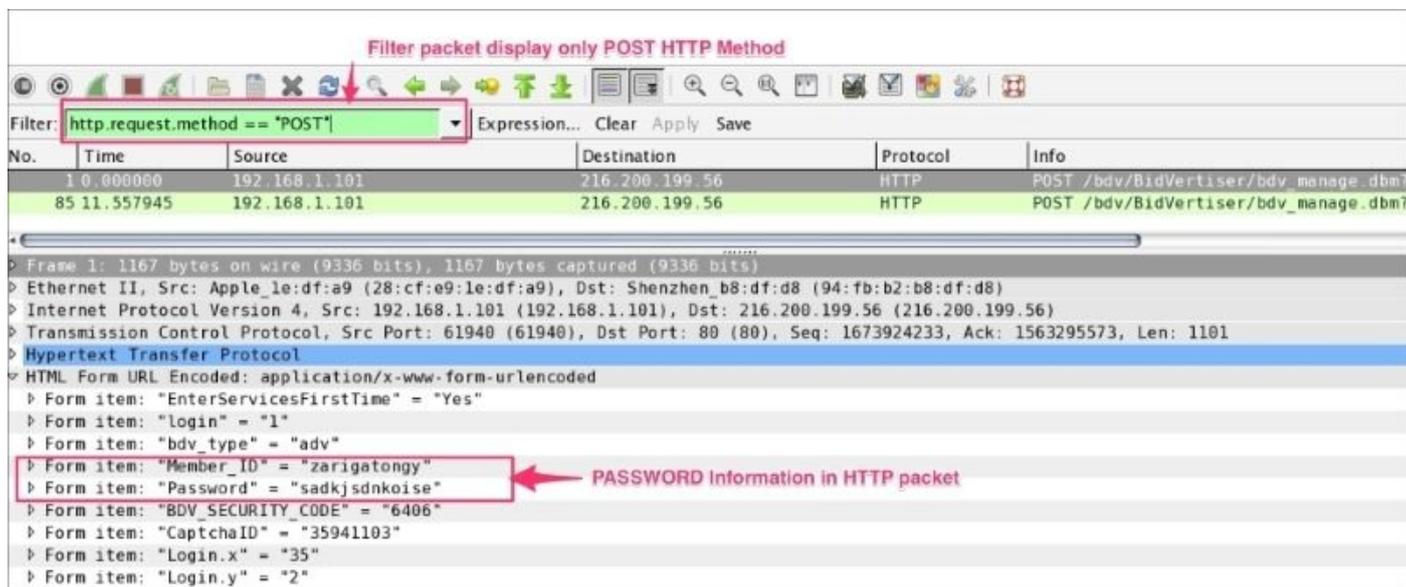
Use Wireshark's `http.request.method` to display packets for analysis. For example, the following table describes how to apply this filter:

HTTP method	Meaning	Wireshark filter
GET	Get a specified resource example: GET <code>http://www.w3.org/pub/WWW/TheProject.html</code> HTTP/1.1	<code>http.request.method=="GET"</code>
POST	Submits data to be processed to a specified resource	<code>http.request.method=="POST"</code>
PUT	Uploads a representation of the specified URI	<code>http.request.method=="PUT"</code>
DELETE	Deletes the specified resource/entity	<code>http.request.method=="DELETE"</code>
OPTIONS	Returns the HTTP methods that the server supports	<code>http.request.method=="OPTIONS"</code>
CONNECT	Converts the request connection to a transparent TCP/IP tunnel	<code>http.request.method=="CONNECT"</code>

Finding sensitive information in a form post

If the form contains sensitive information such as password, Wireshark can easily reveal it as HTTP is an unsecure means of transferring data over the network.

Open the HTTP_FORM_POST.pcap file and filter the traffic to display only the request method POST and locate the password form item, as shown in the following screenshot:



Using HTTP status code

The first line of the HTTP response contains the status code. Use the Wireshark filter `http.response.code`, to display packets based on the status code. This will be helpful when debugging the HTTP client-server interaction:

Type	Code	Meaning	HTTP Wireshark filter
Informational – 1xx	100	Continue	<code>http.response.code == 100</code>
	101	Switching protocol	<code>http.response.code == 101</code>
Successful – 2xx From: 200 To: 206	200	OK	<code>http.response.code == 200</code>
	201	Created	<code>http.response.code == 201</code>
Redirection – 3xx From: 300 To: 307	300	Multiple choices	<code>http.response.code == 300</code>
	301	Moved permanently	<code>http.response.code == 301</code>
Client Error – 4xx From: 400 To: 417	400	Bad Request	<code>http.response.code == 400</code>
	401	Unauthorized	<code>http.response.code == 401</code>
Server Error – 5xx	500	Internal Server Error	<code>http.response.code == 500</code>

From—500			
To— 505	501	Not implemented	http.response.code == 501

References

The HTTP protocol:

- https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- https://wiki.wireshark.org/Hyper_Text_Transfer_Protocol

The DNS protocol:

- https://en.wikipedia.org/wiki/Domain_Name_System#Protocol_transport
- <https://www.ietf.org/rfc/rfc1035.txt>

The DHCP/BOOT protocol:

- <https://tools.ietf.org/html/rfc2131>
- <http://linux.die.net/man/8/dhclient>
- <http://www.iana.org/assignments/bootp-dhcp-parameters/bootp-dhcp-parameters.xhtml>
- <https://goo.gl/snUXkp>

The DHCPv6 protocol:

- <http://www.iana.org/assignments/dhcpv6-parameters/dhcpv6-parameters.xhtml>
- <https://tools.ietf.org/html/rfc3315>
- <https://en.wikipedia.org/wiki/DHCPv6>

Summary

In this chapter, we have learned how Wireshark helps us to analyze application layer protocols such as DHCPv6, DHCP, DNS, and HTTP. We also learned how to simulate these traffic on the wire.

In the next chapter, we will learn more about wireless sniffing.

Chapter 6. WLAN Capturing

So far, we have seen packets captured on Ethernet. In this chapter we will learn how to capture WLAN network traffic, and use effective display filters for all the frames, by covering the following topics:

- WLAN (802.11) capture setup and the monitor mode
- 802.11 capturing with tcpdump
- 802.11 display filters
- Layer-2 datagram frame types and Wireshark display filters
- 802.11 auth process
- 802.1X EAPOL
- 802.11 protocol stack

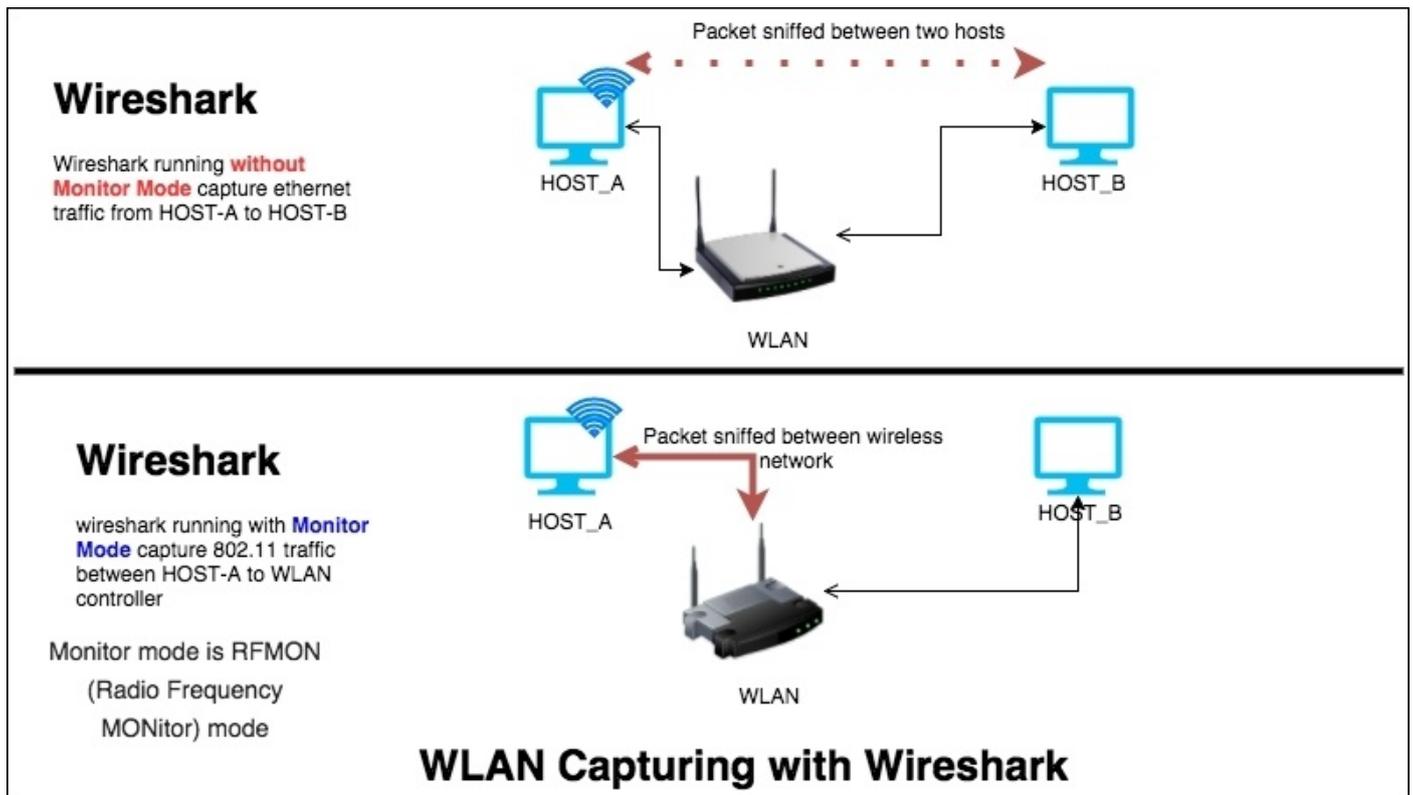
WLAN capture setup

Wireshark depends on the operating system on which it's running (and on the drivers for the wireless adapter) for monitor mode support.

For Linux, the 802.11 wireless toolbar (**View | Wireless Toolbar**) provides excellent options to enable the monitor mode and set the channel for cfg80211 devices. This even supports multiple network interfaces for multi-channel captures; refer to <https://wiki.wireshark.org/CaptureSetup/WLAN> for detailed instructions.

The MAC OS has a wireless adapter, and the monitor mode is supported. On Windows, the monitor mode is not supported; you need a commercial adaptor for this, such as the AirPcap USB adapter.

The WLAN (IEEE 802.11) capturing process is slightly different from capturing Ethernet traffic in Wireshark. By default, when we start capturing traffic in a Wi-Fi network, it captures traffic between two endpoints (HOST-A and HOST-B). To capture the Wi-Fi traffic, Wireshark has to run in the monitor mode—**RFMON (Radio Frequency Monitor)** mode—which allows a computer with a **wireless network interface controller (WNIC)** to monitor all traffic received from the **AP (Access Point)**, as shown in the screenshot:

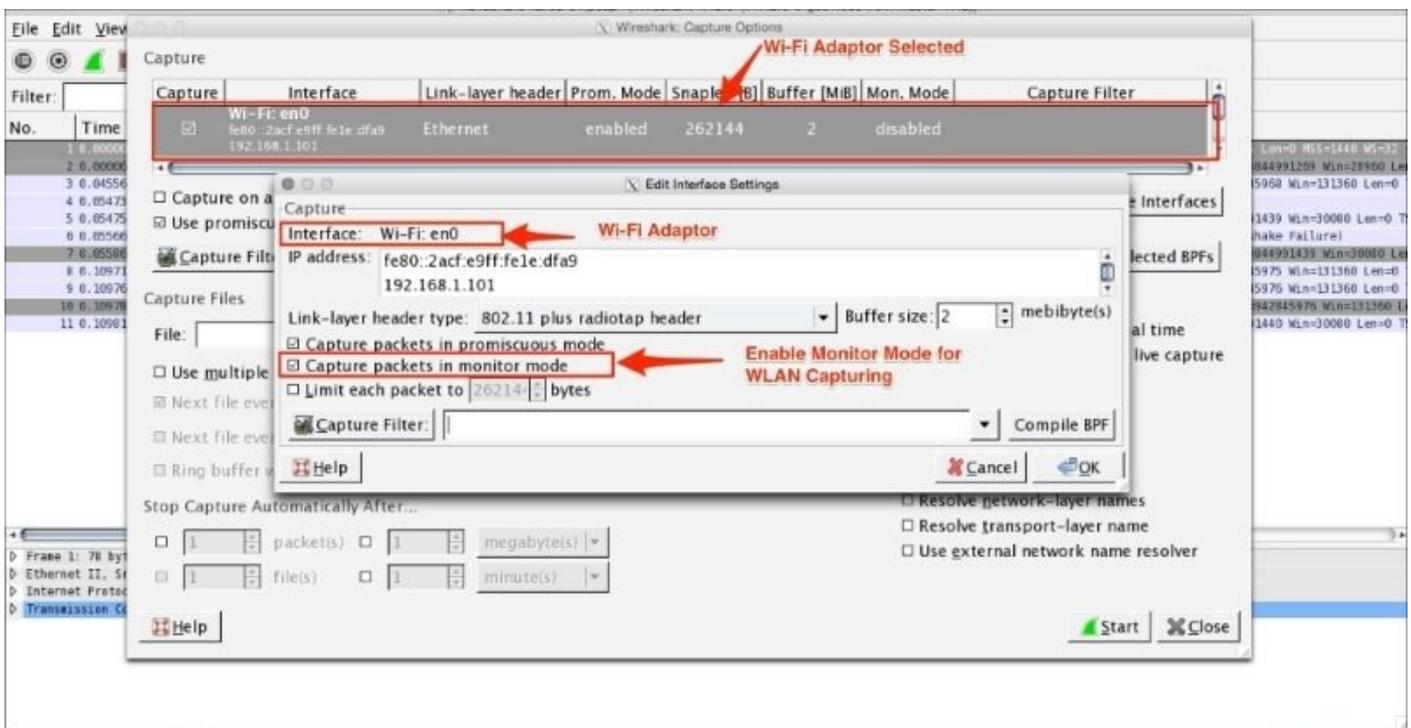


The monitor mode

The monitor mode is supported only on IEEE 802.11 Wi-Fi interfaces, and only on some operating systems. To enable the monitor mode in a Wi-Fi interface, perform these steps in Wireshark:

1. Click on **Capture | Options**.
2. Select the active Wi-Fi adaptor. Double-click on the interface setting; a window will appear.
3. Enable the **Capture packets in Monitor mode** option.
4. Click on **OK**.
5. Start the capture.

You should see the following screen:



When the monitor mode is on, the adapter captures all the packets transmitted on the channel. These include:

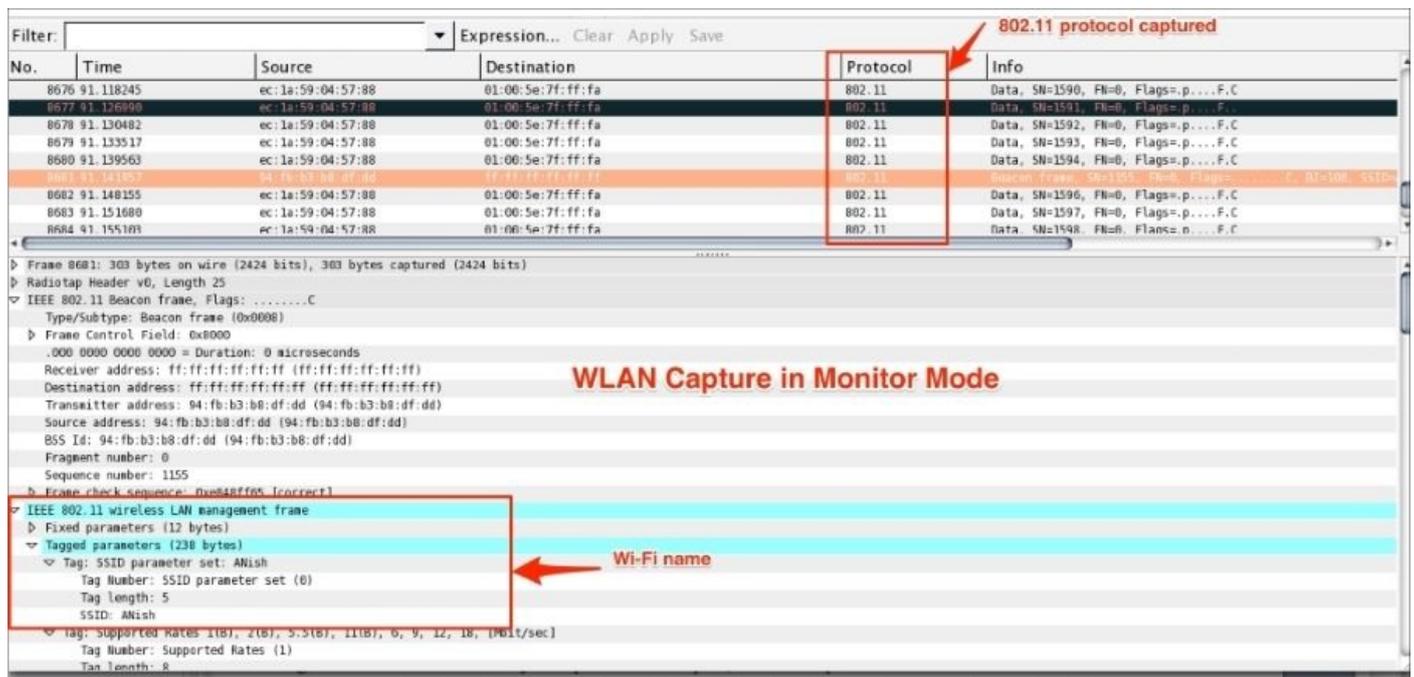
- Unicast packets
- Broadcast packets
- Control and management packets

Tip

Disable name resolution in the monitor mode because Wireshark will try resolving the FQDN, which results in slowness in opening the packet capture file (there is no external network in the monitor mode).

Once the packet capture starts, Wireshark will start displaying the 802.11 protocol packet

exchange between source and destination, as shown in the following screenshot (or open the packet capture 802.11.pcap file in Wireshark). Packet capture in the monitor mode will not be associated with any of the access points and the user can see only 802.11 frames, which include non-data (management and beacon) frames, as shown:



To perform a wireless packet capture using tcpdump, execute the following command. The tcpdump with `-I` option will turn the monitor mode on:

```
bash $ tcpdump -I -P -i en0 -w 802.11.pcap
```

The output obtained is as follows:

```
tcpdump: WARNING: en0: no IPv4 address assigned
tcpdump: listening on en0, link-type IEEE802_11_RADIO (802.11 plus radiotap
header), capture size 65535 bytes
^C52 packets captured
52 packets received by filter
```


Analyzing the Wi-Fi networks

When analyzing a Wi-Fi network, it's important to go through the IEEE standard 802.11 as the source of truth as this is one of the most interesting protocols to gain an expertise on.

Wireless networks are different from a wired LAN: here the addressable unit is a station (STA), and the STA is the message destination not the fixed location when the packet is transferred to the STA.

Within the scope of the book, we are dealing with packets captured between the WNIC controller and the access point. The **access point (AP)** contains one station (STA) and provides access to the distribution. In this book, we will see how Wireshark has provided display filters for analyzing Wi-Fi frames:

- wlan: This displays IEEE 802.11 wireless LAN frame
- wlan_ext: This displays IEEE 802.11 wireless LAN extension frame
- wlan_mgt: This displays IEEE 802.11 wireless LAN management frame
- wlan_aggregate: This displays IEEE 802.11 wireless LAN aggregate frame

Frames

In Layer 2, datagrams are called frames; they show all channel traffic and a count of all the frames received at the measuring STA. There are four types of frame, which are defined in the following table:

Frame type	Value	Wireshark display filter
Management	0x00	wlan.fc.type == 0
Control	0x01	wlan.fc.type == 1
Data	0x02	wlan.fc.type == 2
Extension	0x03	wlan.fc.type == 3

Let's take a detailed look at these frames one by one.

Management frames

Wireshark uses the wlan_mgt display filter to show all the management frames. In line with the IEEE 802.11 standard, the following management frames are defined and their corresponding values, with appropriate Wireshark display filters, are shown in the following table:

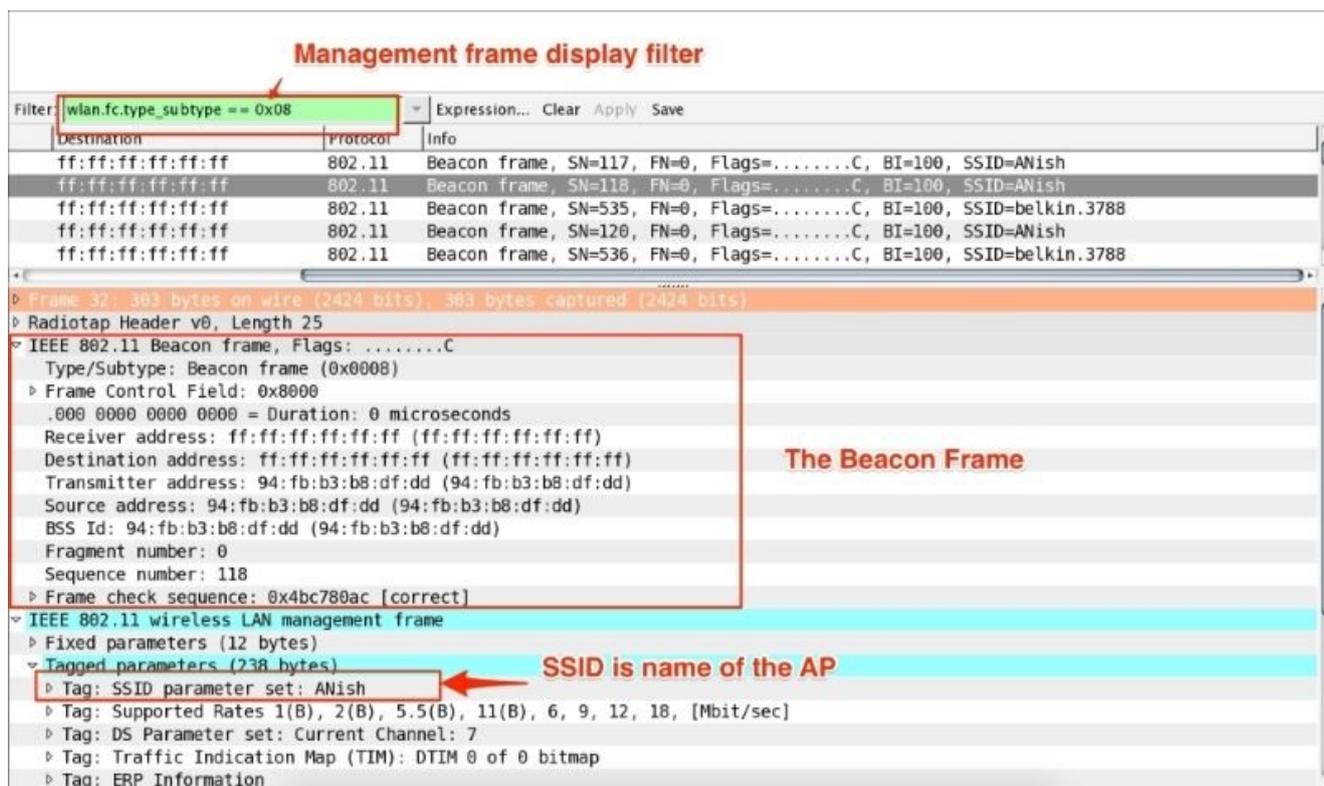
Name	Value	Wireshark display filter
association request	0x00	wlan.fc.type_subtype == 0x00
association response	0x01	wlan.fc.type_subtype == 0x01
reassociation request	0x02	wlan.fc.type_subtype == 0x02
reassociation response	0x03	wlan.fc.type_subtype == 0x03
probe request	0x04	wlan.fc.type_subtype == 0x04
probe response	0x05	wlan.fc.type_subtype == 0x06
measurement pilot	0x06	wlan.fc.type_subtype == 0x06
beacon frame	0x08	wlan.fc.type_subtype == 0x08
atim	0x09	wlan.fc.type_subtype == 0x09
disassociation	0x0a	wlan.fc.type_subtype == 0x0a
authentication	0x0b	wlan.fc.type_subtype == 0x0b
deauthentication	0x0c	wlan.fc.type_subtype == 0x0c
action	0x0d	wlan.fc.type_subtype == 0x0d

```
||action no ack ||0x0e ||wlan.fc.type_subtype == 0x0e||
```

For example, by setting `wlan.fc.type_subtype == 0x08`, in the `802.11.pcap` file, the entire beacon frame will be displayed in Wireshark.

A beacon is a small broadcast data packet that shows the characteristics of the wireless network, and provide information such as data rate (max data rate), capabilities (encryption on or off), Access Point MAC address, SSID (wireless network name), RSN information, vendor specific information, Wi-Fi protected setup, and so on, where:

- SSID is the name of the AP, for example: ANish
- BSSID is the MAC address of the AP, for example is 94:FB:B3:B8:DF:DD



In another example, the `wlan_mgt.ssid == "ANish"` display filter will display all management frames whose SSID matches with ANish.

Data frames

Data frames carry the packets that can contain the payload (such as files, screenshots, and so on). Type values for data frames used in 802.11 and their corresponding Wireshark display filters are shown in the following table:

Name	Value	Wireshark display filter
data	0x20	<code>wlan.fc.type_subtype == 0x20</code>
data + cf-ack	0x21	<code>wlan.fc.type_subtype == 0x21</code>
data + cf-poll	0x22	<code>wlan.fc.type_subtype == 0x22</code>

data + cf-ack + cf-poll	0x23	wlan.fc.type_subtype == 0x23
null function	0x24	wlan.fc.type_subtype == 0x24
no data cf-ack	0x25	wlan.fc.type_subtype == 0x25
no data cf-poll	0x26	wlan.fc.type_subtype == 0x26
no data cf-ack + cf-poll	0x27	wlan.fc.type_subtype == 0x27
qos data	0x28	wlan.fc.type_subtype == 0x28
qos data + cf-ack	0x29	wlan.fc.type_subtype == 0x29
qos data + cf-poll	0x2a	wlan.fc.type_subtype == 0x2a
qos data + cf-ack + cf-poll	0x2b	wlan.fc.type_subtype == 0x2b
qos null	0x2c	wlan.fc.type_subtype == 0x2c
no data qos cf-poll	0x2e	wlan.fc.type_subtype == 0x2e
qos cf-ack + cf-poll	0x2f	wlan.fc.type_subtype == 0x2f

For example, `wlan.fc.type_subtype == 0x2A` will display all the packets that contain QoS Data + CF-Poll in the packet capture file `802.11.pcap`, as shown in the following screenshot:

The screenshot shows the Wireshark interface with a filter applied: `wlan.fc.type_subtype == 0x2A`. The packet list shows a frame at time 3998.000000, source 6c:40:5c:fb:f7:43, destination a6:cd:a2:38:60:04, protocol 802.11, and info QoS Data + CF-Poll, SN=1208, FN=8, Flags=.p..RMFT.

The packet details pane shows the following structure:

- Frame 3998: 1253 bytes on wire (10024 bits), 1253 bytes captured (10024 bits)
- Radiotap Header v0. Length 25
- IEEE 802.11 QoS Data + CF-Poll, Flags: .p..RMFT. (Type/Subtype: QoS Data + CF-Poll (0x002a))
- Frame Control Field: 0xab41
- Duration/ID: 14396 (reserved)
- Receiver address: 6f:43:8a:65:1f:7c (6f:43:8a:65:1f:7c)
- Transmitter address: 93:7d:43:10:16:21 (93:7d:43:10:16:21)
- Destination address: a6:cd:a2:38:60:04 (a6:cd:a2:38:60:04)
- Fragment number: 8
- Sequence number: 1208
- Source address: 6c:40:5c:fb:f7:43 (6c:40:5c:fb:f7:43)
- Frame check sequence: 0xb488955e [incorrect, should be 0x4a5a63fe]
- Qos Control: 0x050e
- TKIP/CCMP parameters
- Data (1184 bytes): 1e04191ccfbcd09e145f3e8784a5ded62b83e6ffbbba20c9e... [Length: 1184]

Annotations in the screenshot point to:

- Data Frame filter Subtype 0x2a**: Points to the filter expression.
- QoS Data + CF-Poll**: Points to the IEEE 802.11 QoS Data + CF-Poll entry in the details pane.
- Encryption Method**: Points to the TKIP/CCMP parameters entry.
- QoS data**: Points to the Data entry in the details pane.

Control frames

Control frames exchange data frames between stations. Control frame ranges are 0x160 -

0x16A for control frame extensions where type = 1 and subtype = 6. Values for control frames and the corresponding Wireshark display filters are shown in the following table:

Name	Value	Wireshark display filter
vht ndp announcement	0x15	wlan.fc.type_subtype == 0x15
poll	0x162	wlan.fc.type_subtype == 0x162
service period request	0x163	wlan.fc.type_subtype == 0x163
grant	0x164	wlan.fc.type_subtype == 0x164
dmg clear to send	0x165	wlan.fc.type_subtype == 0x165
dmg denial to send	0x166	wlan.fc.type_subtype == 0x166
grant acknowledgment	0x167	wlan.fc.type_subtype == 0x167
sector sweep	0x168	wlan.fc.type_subtype == 0x168
sector sweep feedback	0x169	wlan.fc.type_subtype == 0x169
sector sweep acknowledgment	0x16a	wlan.fc.type_subtype == 0x16a
control wrapper	0x17	wlan.fc.type_subtype == 0x17
block ack request	0x18	wlan.fc.type_subtype == 0x18
block ack	0x19	wlan.fc.type_subtype == 0x19
power-save poll	0x1a	wlan.fc.type_subtype == 0x1a
request to send	0x1b	wlan.fc.type_subtype == 0x1b
clear to send	0x1c	wlan.fc.type_subtype == 0x1c
acknowledgement	0x1d	wlan.fc.type_subtype == 0x1d
contention-free period end	0x1e	wlan.fc.type_subtype == 0x1e
contention-free period end/ack	0x1f	wlan.fc.type_subtype == 0x1f

802.11 auth process

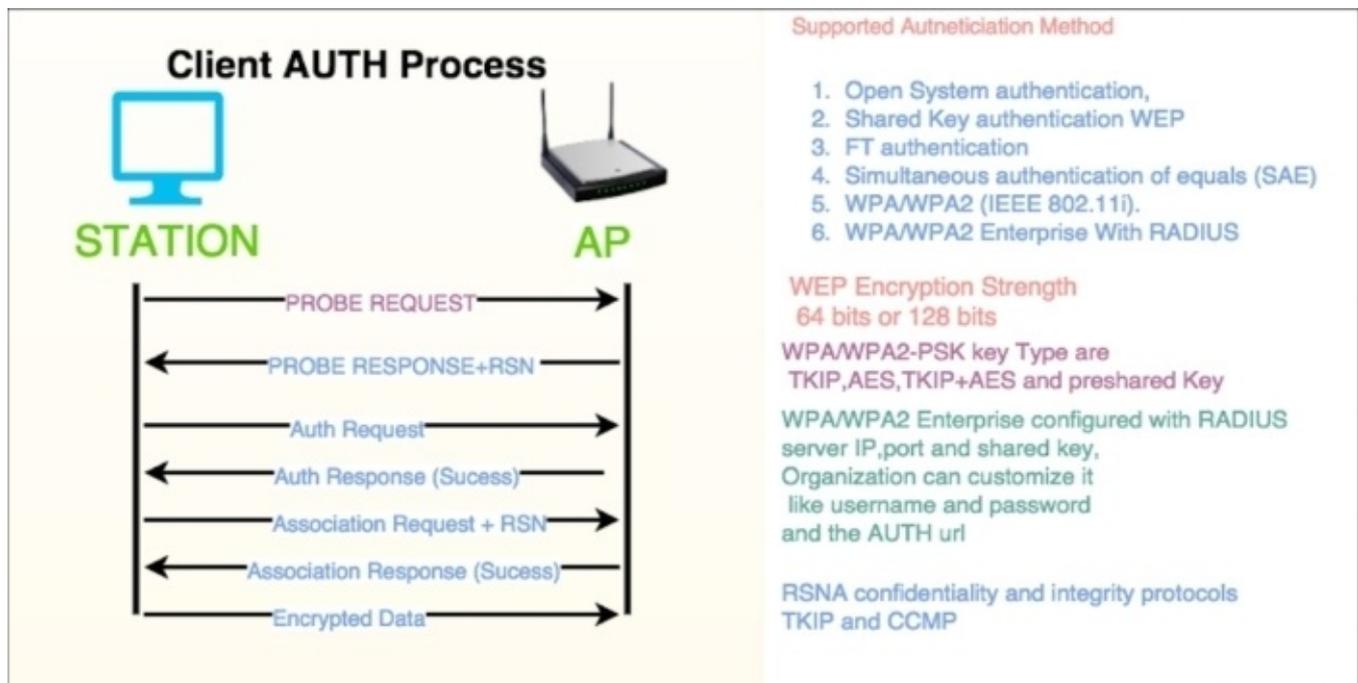
The AP advertises its capabilities in a Beacon frame; the client (STA) broadcasts itself, using its own probe request frame, on every channel—typically (channel 11). By doing this, it determines which access points are within range.

Probe response frames contain capability information, supported data rates and so on, of the AP after it receives a probe request frame.

The STA sends an authentication frame containing its identity to the AP. With open system authentication (the default), the access point responds with an authentication frame as a response, indicating acceptance (or rejection).

Shared key authentication requires WEP (64-bit or 128-bit) keys, and the same WEP keys on the client and AP should be used. The STA requests a shared key authentication, which returns unencrypted challenge text (128 bytes of randomly generated text) from the AP. The STA encrypts the text and returns the data to AP, the AP response indicating acceptance (or rejection).

The STA sends an association request frame to the AP containing the necessary information and then that the AP will send an Association response frame that includes acceptance (or rejection). If this is accepted, the STA can utilize AP to access other networks:



802.1X EAPOL

IEEE802.1x is based on **Extensible Authentication Protocol (EAP)**, which is an extension of **PPP (Point-to-Point Protocol)**, also known as “EAP over LAN” or EAPOL.

The IEEE 802.11 Working Group passed the 802.1x standard in 2001 to improve upon the security specified in the original 802.11 standard (IEEE, 2001).

Open the 802.11-AUTH-enabled.pcap file in Wireshark and use the display filter eapol to display all the eapol messages only, as shown in the following screenshot. In the eapol packets, the session key of the device and the AP are handled.

As shown in the screenshot, all eapol packets are captured as 1 of 4, 2 of 4, 3 of 4, and 4 of 4.

The eapol packets are needed if you are trying to decrypt 802.11 traffic. The Wireshark wiki link <https://wiki.wireshark.org/HowToDecrypt802.11> is an excellent source of information on how to decrypt traffic with the help of Wireshark.

The screenshot shows the Wireshark interface with a display filter of 'eapol'. The packet list pane shows four EAPOL Key messages (1 of 4, 2 of 4, 3 of 4, and 4 of 4). The packet details pane for the selected packet (Frame 412) shows the IEEE 802.11 QoS Data structure, including the Logical-Link Control field. The 802.1X Authentication details are expanded, showing the Key message structure with fields like Version, Type, Length, Key Descriptor Type, Key Information, Key Length, and Replay Counter.

Wireshark eapol filter

Time	Source	Destination	Protocol	Info
412	e8:de:27:59:72:06	98:e7:9a:48:2f:4f	EAPOL	Key (Message 1 of 4)
414	98:e7:9a:48:2f:4f	e8:de:27:59:72:06	EAPOL	Key (Message 2 of 4)
416	e8:de:27:59:72:06	98:e7:9a:48:2f:4f	EAPOL	Key (Message 3 of 4)
424	98:e7:9a:48:2f:4f	e8:de:27:59:72:06	EAPOL	Key (Message 4 of 4)

Link Layer

- 802.1X Authentication
 - Version: 802.1X-2001 (1)
 - Type: Key (3)
 - Length: 95
 - Key Descriptor Type: EAPOL WPA Key (254)
 - Key Information: 0x008a
 - Key Length: 16
 - Replay Counter: 1

The 802.11 protocol stack

The 802.11 standard specifies a common **medium access control (MAC)** layer (the data link layer) that supports the operation of 802.11-based wireless LANs. The 802.11 MAC layer uses an 802.11 **Physical (PHY)** layer, such as 802.11a/b, to perform the tasks of carrier sensing, transmission, and receiving 802.11 frames.

Open the packet capture file `802.11-AUTH-Disabled.pcap` in Wireshark and set the display filter to `wlan.da==e8:de:27:59:72:06` to view how the data is carried using 802.11 as the transport medium.

The 802.11 QoS data frames shows that the LLC header follows IEEE 802.11; this is what is expected in the monitor mode.

The captured 802.11 looks like an Ethernet packet as the 802.11 adapter will often try to transform data packets into fake Ethernet packets and then supply them to the host.

The image shows a Wireshark packet capture analysis. The display filter is `wlan.da==e8:de:27:59:72:06`. The packet list shows several frames, with frame 986 selected. The packet details pane shows the following structure:

- Frame 986: 133 bytes on wire (1064 bits), 133 bytes captured (1064 bits)
- Radiotap Header v0, Length 59
- IEEE 802.11 QoS Data, Flags:T
 - Type/Subtype: QoS Data (0x0028)
 - Frame Control Field: 0x8801
 - .000 0000 0011 0000 = Duration: 48 microseconds
 - Receiver address: e8:de:27:59:72:06 (e8:de:27:59:72:06)
 - BSS Id: e8:de:27:59:72:06 (e8:de:27:59:72:06)
 - Transmitter address: 28:cf:e9:1e:df:a9 (28:cf:e9:1e:df:a9)
 - Source address: 28:cf:e9:1e:df:a9 (28:cf:e9:1e:df:a9)
 - Destination address: e8:de:27:59:72:06 (e8:de:27:59:72:06)
 - Fragment number: 0
 - Sequence number: 542
 - QoS Control: 0x0000
- Logical-Link Control
 - DSAP: SNAP (0xaa)
 - SSAP: SNAP (0xaa)
 - Control field: U, func=UI (0x03)
 - Organization Code: Encapsulated Ethernet (0x000000)
 - Type: IP (0x0800)
- Internet Protocol Version 4, Src: 192.168.1.102 (192.168.1.102), Dst: 192.168.1.1 (192.168.1.1)
- Transmission Control Protocol, Src Port: 65386 (65386), Dst Port: 80 (80), Seq: 2019197508, Ack: 494693892, Len: 0

On the right side of the details pane, a red box highlights the following list:

- TCP Data Carried over the Wifi Network the stack**
- 1. Radiotap Header
- 2. IEEE 802.11 QoS Data
- 3. Logical-Link Control
- 4. Ipv4
- 5. TCP

Wi-Fi sniffing products

There are other commercial (as well as open source) tools that use a form of Wi-Fi sniffing depending on the operating system and uses cases (such as WEP decryption, advance analytics, and geo location). A few of them are listed as follows:

- **Kismet** (<https://www.kismetwireless.net/documentation.shtml>): Kismet can sniff 802.11a/b/g/n Wi-Fi traffic.
- **Riverbed AirPcap** (<http://riverbed.com>): The Riverbed AirPcap adapter is used to capture and analyze 802.11a/b/g/n Wi-Fi traffic and is fully integrated with Wireshark.
- **KisMac** (<http://kismac.en.softonic.com/mac?ex=SWH-1740.2>) for Mac OS X: KisMac offers many of the same features as Kismet and is considered as NetStumbler for Mac. Mac users can find utility tools such as airport ID, airport utility, and Wi-Fi Diagnostics, for sniffing and diagnosing Wi-Fi networks.
- **NetStumbler** (<http://www.netstumbler.com>): This is used for Wi-Fi analysis.

Note

For more information, you can visit the following links:

- <https://wiki.wireshark.org/CaptureSetup/WLAN>
- https://en.wikipedia.org/wiki/IEEE_802.11
- <https://wiki.wireshark.org/HowToDecrypt802.11>
- <https://www.wireshark.org/tools/wpa-psk.html>

Summary

In this chapter, we have covered Wi-Fi capture setup and discussed exactly what the monitor mode is and its pros and cons. We have also learned how the various display filters are used on the Layer 2 datagram (frames). In the next chapter, we will explore network security and its mitigation plans in greater detail.

Chapter 7. Security Analysis

In the previous chapters, we learned more about protocols and their analysis techniques. In this chapter, we will learn how Wireshark helps us perform a security analysis and try to cover the security aspects in these area application and network by covering these topics:

- The Heartbleed bug
- DoS SYN flood/mitigation
- DoS ICMP flood/mitigation
- Scanning the network
- ARP duplicate IP detection (MITM)
- DrDoS introduction
- BitTorrent source identification
- Wireshark endpoints and protocol hierarchy

Heartbleed bug

The Heartbeat protocol (RFC6520) runs on top of the Record layer protocol (the Record layer protocol is defined in SSL).

The Heartbleed bug (CVE-2014-0160) exists in selected OpenSSL versions (1.0.1 to 1.0.1f) that implement the Heartbeat protocol.

This bug is a serious vulnerability that allows attackers to read larger portions of memory (including private keys and passwords) during Heartbeat response.

The Heartbleed Wireshark filter

The Heartbeat protocol runs on top of the Record layer identified as record type (24) in SSL/TLS. In Wireshark, a display filter `ssl.record.content_type == 24` can be used to show the HeartBeat message. Heartbeat messages are Heartbeat Request and HeartBeat Response.

Heartbleed Wireshark analysis

Open the heartbleed.pcap packet capture file in Wireshark and set the display filter to `ssl.record.content_type == 24`.

Wireshark will display only encrypted heartbeat messages. The first one is the Heartbeat Request message. In this message, the length (`ssl.record.length == 112`) of the Heartbeat Request is set to 112 bytes, as shown in the screenshot:

1. Wireshark Heartbeat Protocol Filter

2. Heartbeat Request

3. Heartbeat Response

No.	Time	Source	Destination	Protocol	Info
15	0.102574	52.1.90.117	10.0.0.3	TLSv1.2	Encrypted Heartbeat
16	0.102695	10.0.0.3	52.1.90.117	TLSv1.2	Encrypted Heartbeat

Frame 15: 183 bytes on wire (1464 bits), 183 bytes captured (1464 bits)

Ethernet II, Src: fa:16:3e:23:d3:f1 (fa:16:3e:23:d3:f1), Dst: fa:16:3e:ac:b9:fa (fa:16:3e:ac:b9:fa)

Internet Protocol Version 4, Src: 52.1.90.117 (52.1.90.117), Dst: 10.0.0.3 (10.0.0.3)

Transmission Control Protocol, Src Port: 49578 (49578), Dst Port: 443 (443), Seq: 1136955684, Ack: 663866918, Len: 117

Secure Sockets Layer

TLSv1.2 Record Layer: Encrypted Heartbeat

Content Type: Heartbeat (24)

Version: TLS 1.2 (0x0303)

Length: 112

Encrypted Heartbeat Message

```
0000 fa 16 3e ac b9 fa fa 16 3e 23 d3 f1 08 00 45 00  ..>.....>#...E.
0010 00 a9 54 39 40 00 33 06 5a 9d 34 01 5a 75 0a 00  ..T9@.3. Z.4.Zu..
0020 00 03 c1 aa 01 bb 43 c4 91 24 27 91 ce 26 80 18  ....C. $'..&..
0030 00 e0 bb af 00 00 01 01 08 0a 01 a5 c6 ed 00 15  ....
0040 18 36 18 03 03 30 70 0d 91 a3 74 21 c3 9c 65 a4  .6...g...t!..e.
0050 85 98 51 05 5a ac 01 2d 57 e5 48 6d 33 6b a6 37  ..Q.Z...W.Hm3k.7
0060 b0 ec 01 f5 d5 98 bd 2a 7c 97 21 e1 64 1b 68 d6  ....*|!.d.h.
0070 db c3 b9 10 41 70 e7 72 59 77 5c 95 1a b3 eb a8  ....Ap.r Yw\....
0080 a2 8b c4 d8 2f e9 49 ab 83 03 ea 38 cf 43 df ba  .../I...8.C.
0090 a3 7e be 1d f6 26 07 47 54 db 1a 75 20 ea 7c 19  ~...&.G T..u.].
00a0 1b fc ae fc 29 51 85 ac 02 4e 0f 28 5d 57 d4 48  ...)Q...N.(!W.H
00b0 3d d8 53 bd c5 6a a4  ..S..j
```

Whenever a Heartbeat Request message is sent to the server, the server answers with a corresponding Heartbeat Response message.

In the given packet, the Heartbeat Response length (`ssl.record.length == 144`) is set to 144, which means the server has returned more data (32-bytes more) than expected. This extra information is known as the heartbleed; this bleed may contain sensitive information such as passwords and private keys:

Wireshark filter

Filter: `ssl.record.content_type == 24` Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Info
15	0.102574	52.1.90.117	10.0.0.3	TLSv1.2	Encrypted Heartbeat
16	0.102695	10.0.0.3	52.1.90.117	TLSv1.2	Encrypted Heartbeat

Heartbeat Response

▶ Frame 16: 215 bytes on wire (1720 bits), 215 bytes captured (1720 bits) on interface 0
▶ Ethernet II, Src: fa:16:3e:ac:b9:fa (fa:16:3e:ac:b9:fa), Dst: fa:16:3e:23:d3:f1 (fa:16:3e:23:d3:f1)
▶ Internet Protocol Version 4, Src: 10.0.0.3 (10.0.0.3), Dst: 52.1.90.117 (52.1.90.117)
▶ Transmission Control Protocol, Src Port: 443 (443), Dst Port: 49578 (49578), Seq: 663866918, Ack: 1136955801, Len: 149
▼ Secure Sockets Layer
 ▼ TLSv1.2 Record Layer: Encrypted Heartbeat
 Content Type: Heartbeat (24)
 Version: TLS 1.2 (0x0303)
 Length: 144
 Encrypted Heartbeat Message

Heart bleed happen, as more data is returned from the server

The Heartbleed test

To test the heartbleed, use the following steps:

1. Install OpenSSL version (1.0.1c) from the openssl library:

```
[bash ]# openssl version
OpenSSL 1.0.1c 10 May 2012
```

2. Create a self-signed SSL certificate:

```
[bash #]openssl req -sha256 -new -newkey rsa:2048 -nodes -keyout
./server.key -out ./server.csr -subj "/C=PU/ST=Anish/L=Test/O=Security
Analysis /OU=Heartbleed/CN=myhost.com"
[bash #]openssl x509 -req -days 365 -in server.csr -signkey server.key
-out server.pems
```

3. Start the TLS server using the affected version of OpenSSL:

```
[bash ]# openssl s_server -www -cipher AES256-SHA -key ./server.key -
cert ./server.pem -accept 443
```

4. Start the packet capture:

```
[bash ]# tcpdump port 443 -s0 -w heartbleed.pcap &
```

If the SSL/TLS server is reachable through the public network, online filippo can be used. Other tools (such as Heartbeat Detector, which is a shell script) can also be used for this purpose:

- **Heartbleed Detector:** <https://access.redhat.com/labsinfo/heartbleed>
- **Heartbleed online test:** <https://filippo.io/Heartbleed/>

Heartbleed recommendations

The following are Heartbleed recommendations:

- Apply the patches as recommended in the OpenSSL advisory
- Change the passwords if the vulnerability is addressed.

The DOS attack

This technique is used to attack the host in such a way that the host won't be able to serve any further requests to the user. Finally, the server crashes, resulting in a server unavailable condition.

There are various attack techniques used in this topic. We will cover SYN flood and ICMP flood detection with the help of Wireshark.

SYN flood

We learned about the TCP handshake process in [Chapter 3, Analyzing the TCP Network](#). In this handshake process, a connection is established with SYN, SYN-ACK, and ACK between the client and server.

In the SYN flood attack scenario, what is happening is that:

- The client is sending very fast SYN; it has received the SYN-ACK but doesn't respond with the final ACK
- Alternatively, the client is sending very fast SYN and blocking the SYN-ACK from the server, or the client is sending very fast SYN from a spoofed IP address so the SYN-ACK is sent to an unknown host that virtually doesn't exist

In all these scenarios, the TCP/IP stack file descriptors are consumed, causing the server to slow down and finally crash.

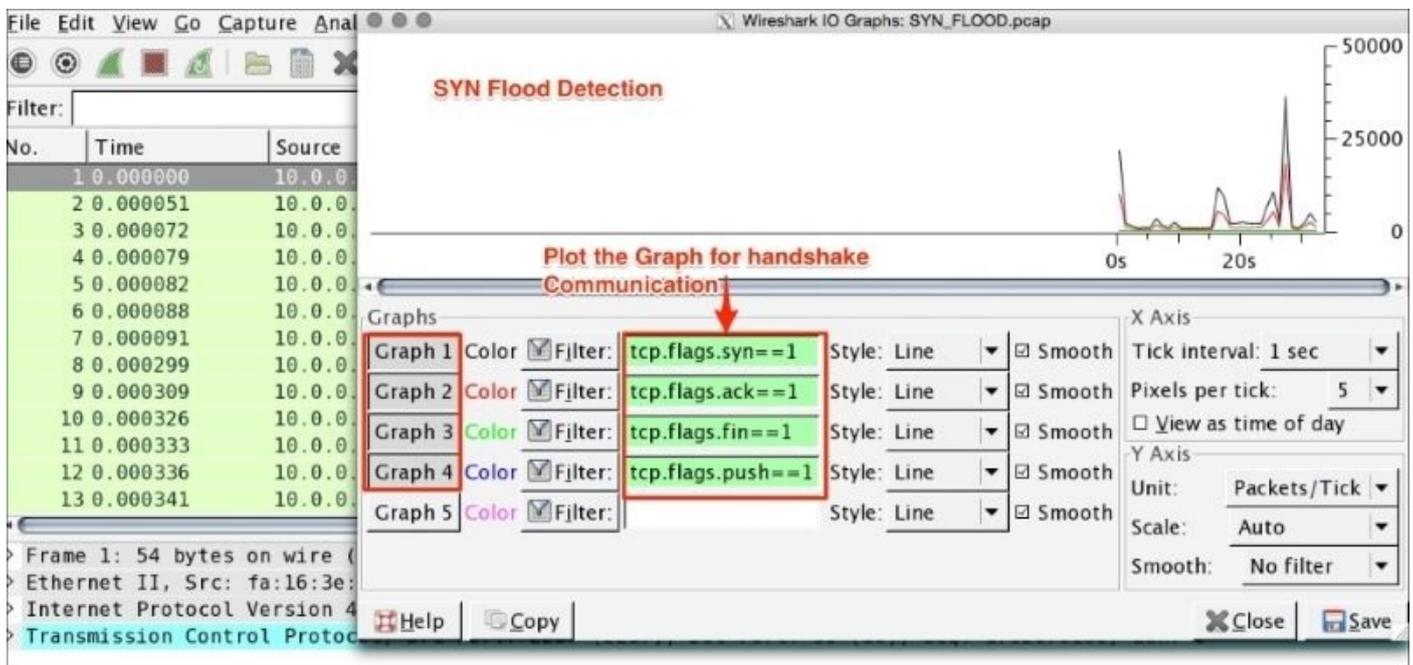
Open the `SYN_FLOOD.pcap` packet capture file in Wireshark and perform the following steps:

1. Click on **Statistics | IO Graph**.
2. The **IO Graph** dialog box will appear.
3. Generate four graphs for the TCP handshake message SYN, ACK, FIN, and PUSH.

The IO graph statistics show the following summary:

- The TCP connection never closes as there is no count for `tcp.flags.fin`
- The TCP connection never exchanges any data as there is no count for `tcp.flags.push`
- The count of SYN packets is very high
- The count of ACK is half of that of the SYN packets

In real scenarios, this data will be mixed up with actual packet flows, but the analysis technique will remain the same. The moment you see an unexpected growth in SYN packets or a spike in SYN packets, it's a SYN flood from DoS or from the multiple-source DDoS.



SYN flood mitigation

SYN attacks can be mitigated. The following are a few mitigation plans:

- **TCP/IP stack hardening:** The operating system decides how many times SYN, SYN-ACK, ACK will be repeated; lowering the SYN,ACK retries will help the server mitigate SYN flood attacks. A SYN cookie is used to resist SYN flood attacks. To perform all these on Linux systems, edit the `/etc/sysctl.conf` file and make changes to these entries:

```
#Prevent SYN attack, enable SYNcookies (they will kick-in when the
max_syn_backlog reached)
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_syn_retries = 2
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_max_syn_backlog = 4096
# Increase the tcp-time-wait buckets pool size to prevent simple DOS
attacks
net.ipv4.tcp_max_tw_buckets = 1440000
```

- Restart `sysctl` to apply the changes:

```
bash#sysctl -p
```

- IPtables firewalls can be set to deny the IPs that are causing the problem. To generate the firewall rules, use the Wireshark feature generating Firewall rules to *drop* the traffic that is causing DoS.
- For example, blocking the traffic causing the DoS:

```
# Netfilter (iptables)
iptables -A INPUT -i eth0 -d 10.0.0.3/32 -j DROP
! Cisco IOS (standard)
access-list NUMBER deny host 10.0.0.3
# IPFirewall (ipfw)
```

```
add deny ip from 10.0.0.3 to any in
# Windows Firewall (netsh)
add portopening tcp 443 Wireshark DISABLE 10.0.0.3
```

- Ports opened to the external world should be audited.
- Monitoring by creating alerts on the spikes that show unhealthy trends on the network which can result in the DoS scenario; generate the firewall rule dynamically and apply it on the targeted VM.
- Network ACLs block the traffic at the router level; introduce the IDS/IPS system to the network.
- Use the loadbalancer as the connection off-loader. In this case, if an attack happens, it will happen on the loadbalancer. The VM will remain protected. Most of the commercially available loadbalancers have the ability to defend themselves from this type of attack.
- Rate-limiting the SYN per second per IP.
- Put DoS/DDoS protection on the data center edge router (L2).
- Apply multiple levels of detection and knowing the signatures and attributes of suspected traffic locations.
- Prepare mitigation plans.

ICMP flood

Internet Control Message Protocol (ICMP) flood is also categorized as a Layer 3 DoS attack or a DDoS attack. It works as follows: an attacker is trying to flood the echo request (ping) packet with a spoofed IP address or the server is flooded with echo requests (ping packets) and not able to process the echo response for each ICMP echo request, resulting in host slowness and denial of service.

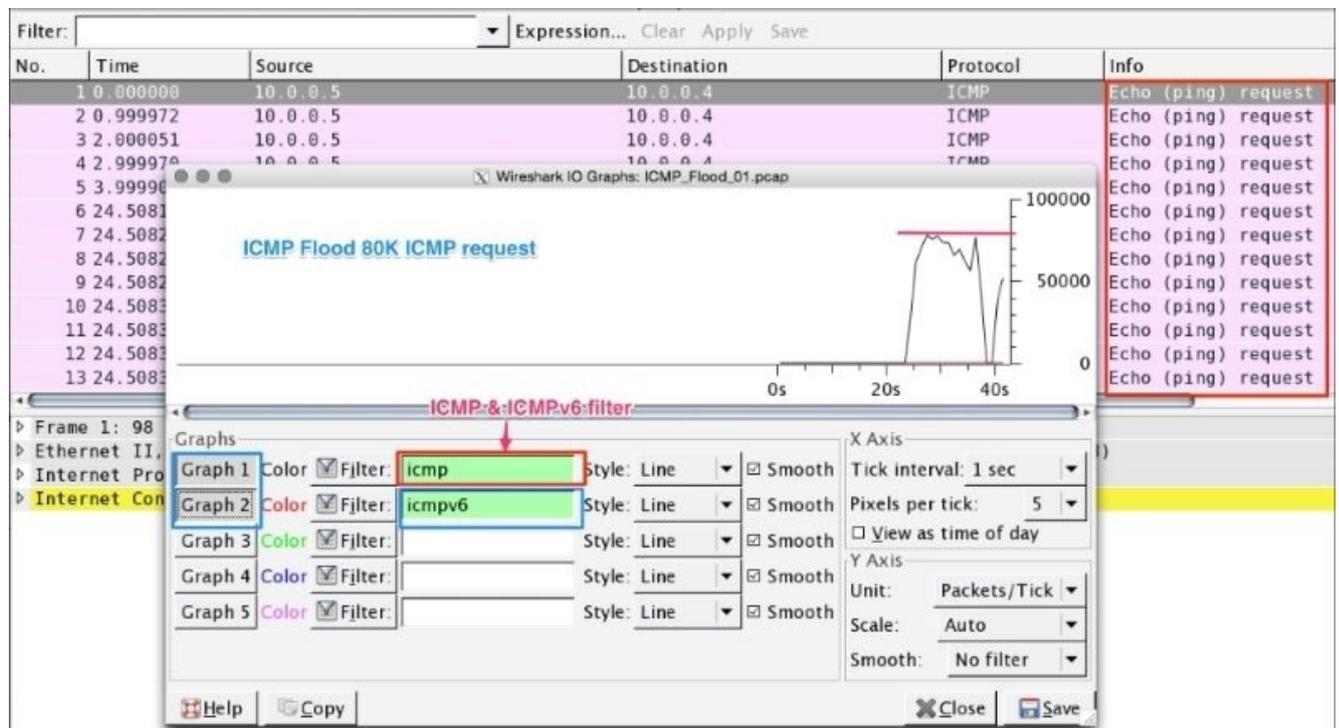
Open the ICMP_Flood_01.pcap packet capture file in Wireshark and perform the following steps:

1. Click on **Statistics | IO Graph**.
2. The **IO Graph** dialog box will appear.
3. Generate graphs for ICMP and ICMPv6.

As shown in the screenshot, ICMP flood has the following characteristics:

- The IO graph shows a large number of ICMP packets: nearly 80K ping requests in a short period of time
- The packet capture doesn't have the echo reply message

This is sample data; in real environment it may vary as attackers are also learning and finding new ways to perform ICMP DoS.



ICMP flood mitigation

The following are a few mitigation plans for the ICMP flood attack:

- **OS hardening:** On the host machine (production environment) disable the ICMP and ICMPv6 protocol through the iptables firewall:

```
bash# iptables -I INPUT -p icmp --icmp-type 8 -j DROP
bash# iptables -A OUTPUT -p icmp -o eth0 -j ACCEPT
bash# iptables -A INPUT -p icmp --icmp-type echo-reply -s 0/0 -i eth0 -
j ACCEPT
bash# iptables -A INPUT -p icmp --icmp-type destination-unreachable -s
0/0 -i eth0 -j ACCEPT
bash# iptables -A INPUT -p icmp --icmp-type time-exceeded -s 0/0 -i
eth0
-j ACCEPT
bash# iptables -A INPUT -p icmp -i eth0 -j DROP
bash# ip6tables -I INPUT -p icmpv6 -icmpv6-type 8 -j DROP
bash# ip6tables -I INPUT -p icmpv6 -i eth0 -j DROP
```

- TCP/IP stack hardening: by editing the `sysctl.conf` file and adding the following entry in this file:

```
net.ipv4.icmp_echo_ignore_all = 1
```

- Restart `sysctl` to apply the changes:

```
bash#sysctl -p
```

- Rate-limiting on the Router level if ICMP/ICMPv6 traffic is allowed
- The firewall should block the ICMP/ICPMv6 traffic on the router

SSL flood

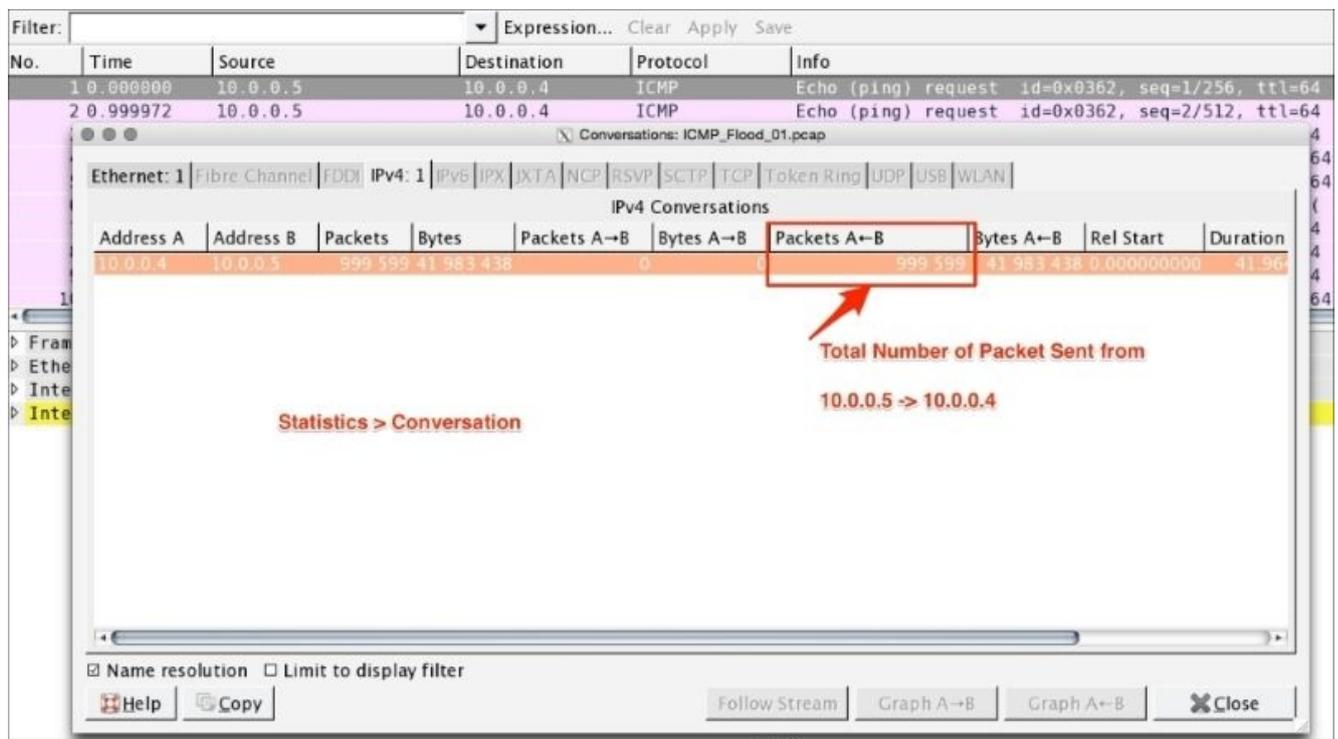
This kind of attack happens on Layer 7 and it is difficult to detect in the sense that it resembles legitimate website traffic. In Analyzing SSL/TLS, we learned about SSL and the handshake process. The attacker can use the handshake against the system to create a DoS/DDoS attack. As handshake involves larger exchange of message between client and the server, for example, in case of one way auth total number of packet exchanges to established a connection is approximate 12 (that is, *3 packets TCP handshake + 9 packets SSL handshake = 12 packets exchanged*).

The attacker can flood the SSL connection and make the server busy, to just establish the connection and try to create the DoS/DDoS scenario.

Wireshark can help in identifying from which IP maximum number of packet has arrived. This feature is called Wireshark Conversations, and can be used in any kind of flood scenario (DoS attack).

Open the ICMP_Flood_01.pcap packet capture file in Wireshark and perform the following steps:

1. Click on **Statistics | Conversations**.
2. A conversation dialog box will appear as shown in the screenshot. An unusually higher volume of traffic is generated from source B (10.0.0.5) to source A (10.0.0.4), causing the network to slow down:



Other categories of Layer 7 attacks are HTTP/HTTPS POST flood and HTTP/HTTPS GET flood.

Scanning

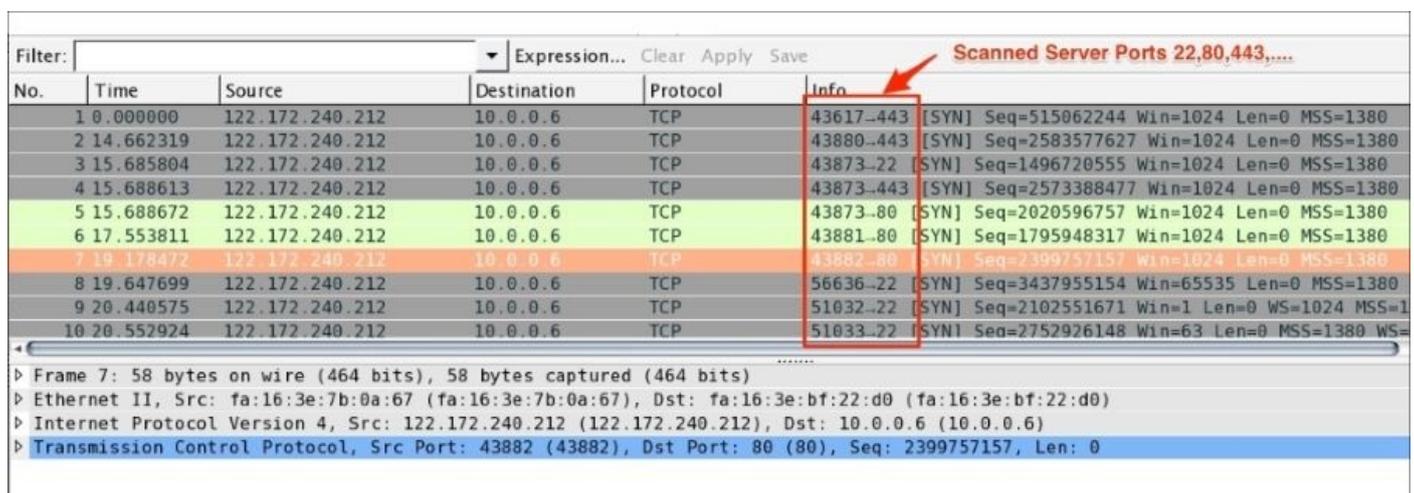
In this section, we will go over the basics of vulnerability scanning and verify what is happening when the host scan is performed with the help of Wireshark.

Vulnerability scanning

Host discovery, port scanning, and OS detection are part of vulnerability scanning. During this process, vulnerabilities are identified and addressed with a proper mitigation plan by the security auditor. For example:

- The security auditor scans hosts to check that only allowed ports are open to the external world
- The hacker scans the ports to find out which services are up and running, for example during this host scan process if the DB ports are open to the outside world then the DB system is compromised for attacks.

Open the `host_scan.pcap` file in Wireshark; the sample capture shows how the external client is scanning the ports:



No.	Time	Source	Destination	Protocol	Info
1	0.000000	122.172.240.212	10.0.0.6	TCP	43617..443 [SYN] Seq=515062244 Win=1024 Len=0 MSS=1380
2	14.662319	122.172.240.212	10.0.0.6	TCP	43880..443 [SYN] Seq=2583577627 Win=1024 Len=0 MSS=1380
3	15.685804	122.172.240.212	10.0.0.6	TCP	43873..22 [SYN] Seq=1496720555 Win=1024 Len=0 MSS=1380
4	15.688613	122.172.240.212	10.0.0.6	TCP	43873..443 [SYN] Seq=2573388477 Win=1024 Len=0 MSS=1380
5	15.688672	122.172.240.212	10.0.0.6	TCP	43873..80 [SYN] Seq=2020596757 Win=1024 Len=0 MSS=1380
6	17.553811	122.172.240.212	10.0.0.6	TCP	43881..80 [SYN] Seq=1795948317 Win=1024 Len=0 MSS=1380
7	19.178472	122.172.240.212	10.0.0.6	TCP	43882..80 [SYN] Seq=2399757157 Win=1024 Len=0 MSS=1380
8	19.647699	122.172.240.212	10.0.0.6	TCP	56636..22 [SYN] Seq=3437955154 Win=65535 Len=0 MSS=1380
9	20.440575	122.172.240.212	10.0.0.6	TCP	51032..22 [SYN] Seq=2102551671 Win=1 Len=0 WS=1024 MSS=1380
10	20.552924	122.172.240.212	10.0.0.6	TCP	51033..22 [SYN] Seq=2752926148 Win=63 Len=0 MSS=1380 WS=1

Frame 7: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
Ethernet II, Src: fa:16:3e:7b:0a:67 (fa:16:3e:7b:0a:67), Dst: fa:16:3e:bf:22:d0 (fa:16:3e:bf:22:d0)
Internet Protocol Version 4, Src: 122.172.240.212 (122.172.240.212), Dst: 10.0.0.6 (10.0.0.6)
Transmission Control Protocol, Src Port: 43882 (43882), Dst Port: 80 (80), Seq: 2399757157, Len: 0

During this process, a SYN packet is sent to all the ports for common services on each host, such as DNS, LDAP, HTTP and many more. If we get the ACK from the host, the host is considered ACTIVE on that port.

The security auditor or hacker can use network scanner tools to get the port, host, and OS information. For example, the `nmap` network utility command can be used to scan the active/open ports:

1. Scan standard ports in the host:

```
bash# nmap -T4 -A -v 128.136.179.233
```

2. Scan all active ports in the host:

```
bash# nmap -p 1-65535 -T4 -A -v 128.136.179.233
```

The online `nmap` tool can be found at <https://pentest-tools.com/network-vulnerability-scanning/tcp-port-scanner-online-nmap>.

SSL scans

SSL scans are done by different users (for example, security auditors and hackers) to achieve their own objectives:

- The security auditor uses a SSL scanner to find the weakest cipher suites or vulnerable SSL protocol versions present in the SSL server, to remove them
- The hacker uses a SSL scanner to hack the encrypted SSL communication by finding weak cipher suites or vulnerable protocol versions in the SSL server

An example using the nmap command to find available ciphers and the supported protocol version in a given server port 636 LDAP is as shown:

```
[root@ ~]# nmap --script ssl-cert,ssl-enum-ciphers -p 636 10.10.1.3To find available ciphers and the supported protocol version in a given server port 443 HTTPS
```

```
[root@ ~]# nmap --script ssl-cert,ssl-enum-ciphers -p 443 10.10.1.3
```


ARP duplicate IP detection

Wireshark detects duplicate IPs in the ARP protocol. Use the `arp.duplicate-address-frame` Wireshark filter to display only duplicate IP information frames.

For example, open the `ARP_Duplicate_IP.pcap` file and apply the `arp.duplicate-address-frame` filter, as shown in the screenshot:

The screenshot displays the Wireshark interface with the following components:

- Filter:** `arp.duplicate-address-frame` (highlighted in green).
- Expression...** Clear Apply Save
- ARP Protocol** (highlighted in red).
- 1. Note all IP belongs to same MAC address** (highlighted in red).
- Packet List:** A table of 18 ARP frames. All frames have the same Source MAC address: `fa:16:3e:bf:22:d0`. The Destination MAC addresses vary, and the IP addresses in the Info column are also duplicated.
- Packet Details:** Frame 2 details are expanded, showing Ethernet II and Expert Info. The Expert Info pane contains two entries:
 - `[Duplicate IP address detected for 10.0.0.8 (fa:16:3e:bf:22:d0) - also in use by fa:16:3e:52:0e:55 (frame 1)]`
 - `[Duplicate IP address detected for 10.0.0.7 (fa:16:3e:19:5a:cc) - also in use by fa:16:3e:bf:22:d0 (frame 1)]`

Wireshark is providing the following information in this case:

- Usually duplicate IP addresses are resolved by the DHCP server. It has to be taken seriously when it starts showing for every IP address in this case.
- All IPs have the same Sender MAC address: `fa:16:3e:bf:22:d0` and shows as a duplicate of that IP address.
- This could be ARP poisoning—a Man in Middle attack happening in the background.

DrDoS

Distributed Reflection Denial of Service (DrDoS), also known as UDP-based amplification attacks, uses publically accessible UDP servers and bandwidth amplification factors to overwhelm a system with UDP traffic.

Open the `DrDoS.pcap` file. In this packet capture, a SYN packet is sent over a server IP address with the victim's source IP address; note the destination port is HTTP 80 and the source port is NTP port 123, UDP. Now the server will respond with an ACK packet to the source that in this case will be the victim's IP address. If multiple servers were used, the server will flood the victim (target) with ACK packets.

There are UDP protocols (DNS, NTP, and BitTorrent) that are infected by UDP-Based amplification attacks. For more information on this, refer to alert TA14-017A published by US-CERT: <https://www.us-cert.gov/ncas/alerts/TA14-017A>.

BitTorrent

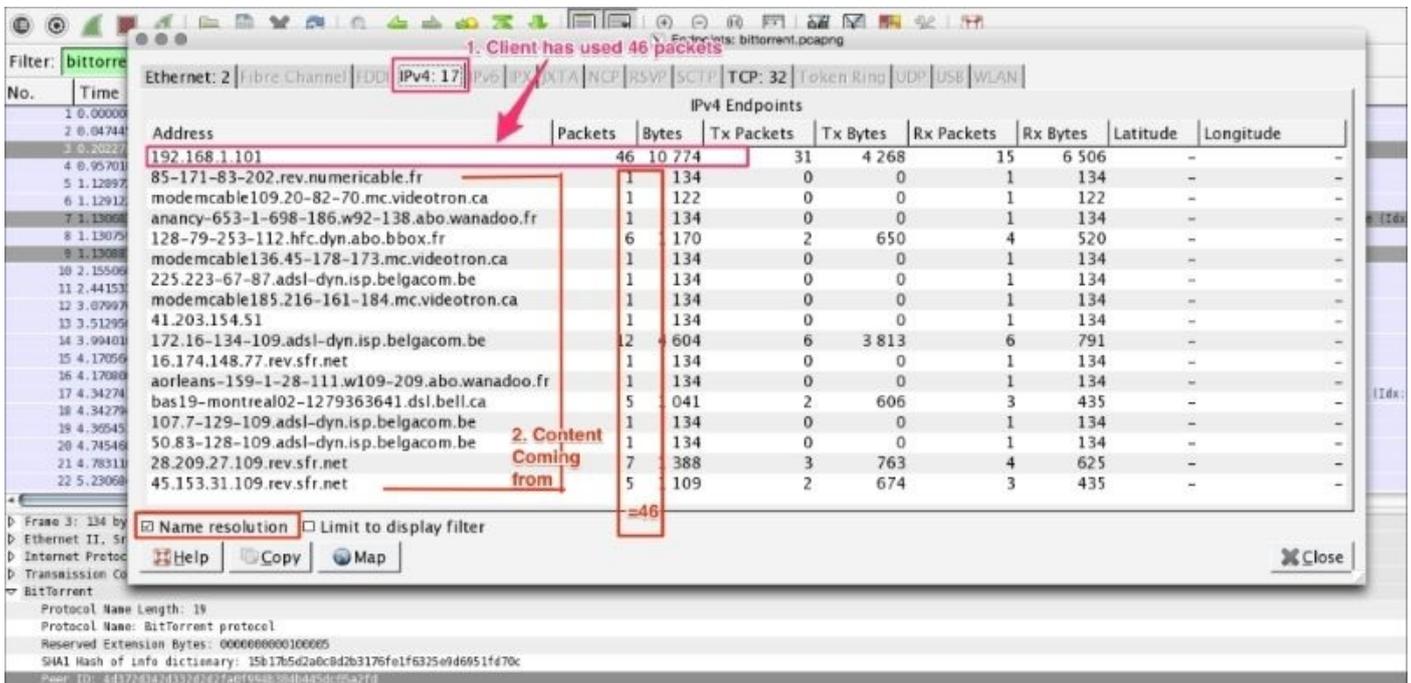
Wireshark supports the BitTorrent protocol. BitTorrent uses the Torrent file to download the content from the P2P network. The content that gets download through these programs is safe (depending on what kind of content is downloaded). Any download can contain Trojans or viruses so (this recommendation goes for any protocol used) be careful, especially when downloading any executable file or from unknown torrent URLs. All downloaded files are subjected to a scan. Open the bittorrent.pcapng file in Wireshark and check from that location that the content is getting downloaded.

The screenshot shows the Wireshark interface with a filter set to 'bittorrent'. A red arrow points to the filter box with the label 'Wireshark filter'. Another red arrow points to the 'Destination' column of the packet list with the label 'Many Destination for the same client'. The packet list shows 22 packets, all from source 192.168.1.101, going to various destinations. Below the list, the details pane shows the structure of a BitTorrent packet: Protocol Name Length: 19, Protocol Name: BitTorrent protocol, Reserved Extension Bytes: 000000000100005, SHA1 Hash of info dictionary: 15b17b5d2a0c8d2b3176fe1f6325e9d6951fd70c, and Peer ID: 4d372d342d332d2d2fa0f994b3b4b445dc65a2fd.

No.	Time	Source	Destination	Protocol
1	0.000000000	192.168.1.101	85-171-83-202.rev.numericable.fr	BitTorrent
2	0.047445000	192.168.1.101	modemcable109.20-82-70.mc.videotron.ca	BitTorrent
3	0.202278000	192.168.1.101	anancy-653-1-698-186.w92-138.abo.wanadoo.fr	BitTorrent
4	0.957010000	192.168.1.101	128-79-253-112.hfc.dyn.abo.bbox.fr	BitTorrent
5	1.128973000	128-79-253-112.hfc.dyn.abo.bbox.fr	192.168.1.101	BitTorrent
6	1.129122000	192.168.1.101	128-79-253-112.hfc.dyn.abo.bbox.fr	BitTorrent
7	1.130683000	128-79-253-112.hfc.dyn.abo.bbox.fr	192.168.1.101	BitTorrent
8	1.130759000	192.168.1.101	128-79-253-112.hfc.dyn.abo.bbox.fr	BitTorrent
9	1.130887000	192.168.1.101	128-79-253-112.hfc.dyn.abo.bbox.fr	BitTorrent
10	2.155060000	192.168.1.101	modemcable136.45-178-173.mc.videotron.ca	BitTorrent
11	2.441533000	192.168.1.101	225.223-67-87.adsl-dyn.isp.belgacom.be	BitTorrent
12	3.079976000	192.168.1.101	modemcable185.216-161-184.mc.videotron.ca	BitTorrent
13	3.512956000	192.168.1.101	41.203.154.51	BitTorrent
14	3.994010000	192.168.1.101	172.16-134-109.adsl-dyn.isp.belgacom.be	BitTorrent
15	4.170564000	172.16-134-109.adsl-dyn.isp.belgacom.be	192.168.1.101	BitTorrent
16	4.170806000	192.168.1.101	172.16-134-109.adsl-dyn.isp.belgacom.be	BitTorrent
17	4.342741000	172.16-134-109.adsl-dyn.isp.belgacom.be	192.168.1.101	BitTorrent
18	4.342794000	192.168.1.101	172.16-134-109.adsl-dyn.isp.belgacom.be	BitTorrent
19	4.365451000	192.168.1.101	16.174.148.77.rev.sfr.net	BitTorrent
20	4.745460000	172.16-134-109.adsl-dyn.isp.belgacom.be	192.168.1.101	BitTorrent
21	4.783116000	192.168.1.101	172.16-134-109.adsl-dyn.isp.belgacom.be	BitTorrent
22	5.230684000	192.168.1.101	aorleans-159-1-28-111.w109-209.abo.wanadoo.fr	BitTorrent

Frame 3: 134 bytes on wire (1072 bits), 134 bytes captured (1072 bits) on interface 0
Ethernet II, Src: Apple_1e:df:a9 (28:cf:e9:1e:df:a9), Dst: Shenzhen_b8:df:d8 (94:fb:b2:b8:df:d8)
Internet Protocol Version 4, Src: 192.168.1.101 (192.168.1.101), Dst: anancy-653-1-698-186.w92-138.abo.wanadoo.fr (92.138.200.186)
Transmission Control Protocol, Src Port: 49380 (49380), Dst Port: 37769 (37769), Seq: 964563004, Ack: 1831901839, Len: 68
BitTorrent
Protocol Name Length: 19
Protocol Name: BitTorrent protocol
Reserved Extension Bytes: 000000000100005
SHA1 Hash of info dictionary: 15b17b5d2a0c8d2b3176fe1f6325e9d6951fd70c
Peer ID: 4d372d342d332d2d2fa0f994b3b4b445dc65a2fd

The Wireshark BitTorrent dissector is able to decode the entire download process. To check what the endpoints are from this source, do the following. Click on **Statistics | Endpoints**; an Endpoint Window will appear:



As shown in the screenshot, Wireshark has obtained the following information:

1. Filter the protocol, in this case BitTorrent.
2. Select the Ipv4 TAB.
3. In this capture, name resolution is enabled.
4. The client (192.168.1.101) has downloaded 10744 bytes and the content is coming from different geographical locations. Since the content was downloaded from various sources, it is always advised to scan it before opening it.

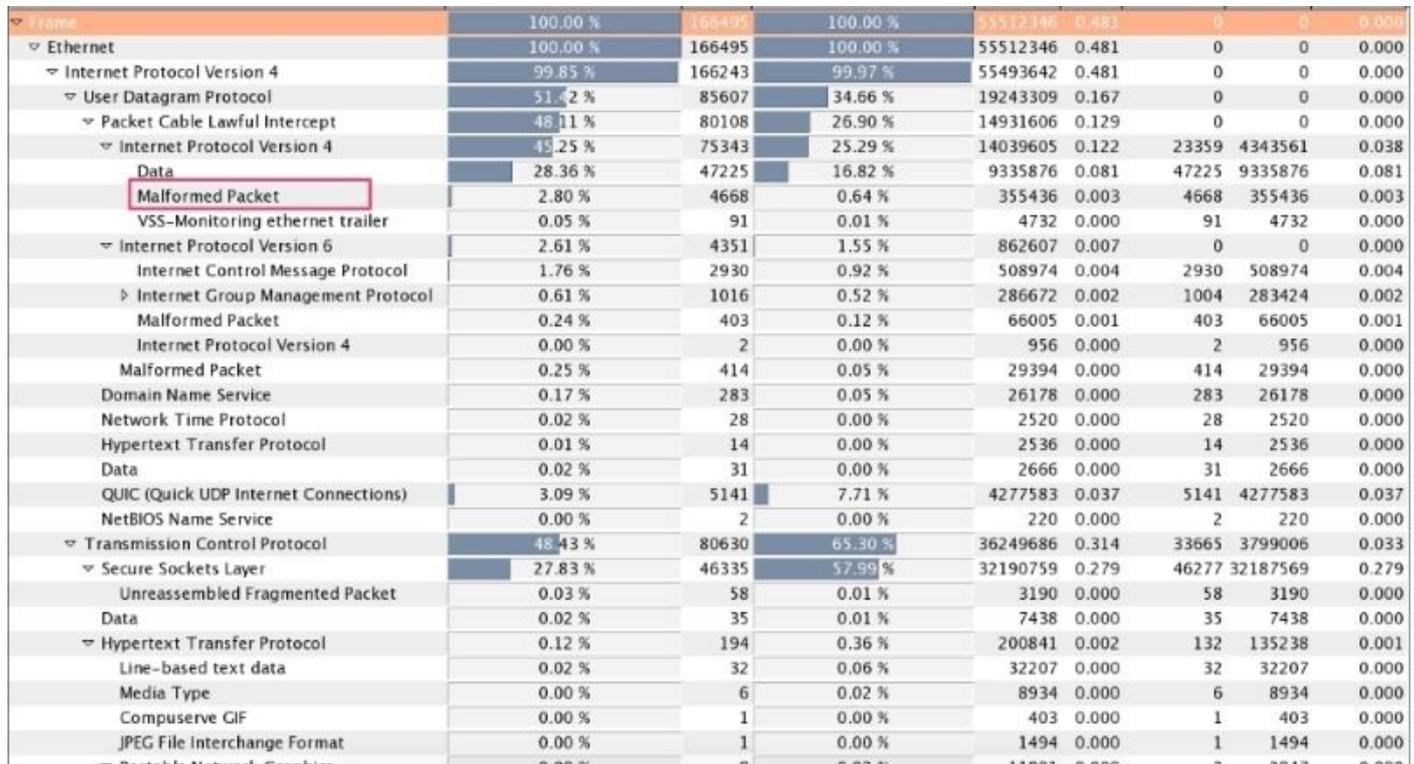
Endpoint statistics are a nice Wireshark feature. Endpoints reveal information such as outgoing connections for a given client. In this example, the client is connected to 16 different endpoint locations spread across different geographical locations. For any suspicious traffic, use the filter option directly on the Endpoint window.

Note

Note: Wireshark will not notify or scan for a virus; it helps to analyze the virus.

Wireshark protocol hierarchy

This feature is very useful when dealing with what protocols are running on the server. To find this, click on **Summary | Protocol Hierarchy** in the Wireshark menu. A protocol hierarchy of the captured packets will open, as shown in the screenshot:



Frame	100.00 %	166495	100.00 %	55512346	0.481	0	0	0.000
▼ Ethernet	100.00 %	166495	100.00 %	55512346	0.481	0	0	0.000
▼ Internet Protocol Version 4	99.85 %	166243	99.97 %	55493642	0.481	0	0	0.000
▼ User Datagram Protocol	51.42 %	85607	34.66 %	19243309	0.167	0	0	0.000
▼ Packet Cable Lawful Intercept	48.11 %	80108	26.90 %	14931606	0.129	0	0	0.000
▼ Internet Protocol Version 4	45.25 %	75343	25.29 %	14039605	0.122	23359	4343561	0.038
Data	28.36 %	47225	16.82 %	9335876	0.081	47225	9335876	0.081
Malformed Packet	2.80 %	4668	0.64 %	355436	0.003	4668	355436	0.003
VSS-Monitoring ethernet trailer	0.05 %	91	0.01 %	4732	0.000	91	4732	0.000
▼ Internet Protocol Version 6	2.61 %	4351	1.55 %	862607	0.007	0	0	0.000
Internet Control Message Protocol	1.76 %	2930	0.92 %	508974	0.004	2930	508974	0.004
▶ Internet Group Management Protocol	0.61 %	1016	0.52 %	286672	0.002	1004	283424	0.002
Malformed Packet	0.24 %	403	0.12 %	66005	0.001	403	66005	0.001
Internet Protocol Version 4	0.00 %	2	0.00 %	956	0.000	2	956	0.000
Malformed Packet	0.25 %	414	0.05 %	29394	0.000	414	29394	0.000
Domain Name Service	0.17 %	283	0.05 %	26178	0.000	283	26178	0.000
Network Time Protocol	0.02 %	28	0.00 %	2520	0.000	28	2520	0.000
Hypertext Transfer Protocol	0.01 %	14	0.00 %	2536	0.000	14	2536	0.000
Data	0.02 %	31	0.00 %	2666	0.000	31	2666	0.000
QUIC (Quick UDP Internet Connections)	3.09 %	5141	7.71 %	4277583	0.037	5141	4277583	0.037
NetBIOS Name Service	0.00 %	2	0.00 %	220	0.000	2	220	0.000
▼ Transmission Control Protocol	48.43 %	80630	65.30 %	36249686	0.314	33665	3799006	0.033
▼ Secure Sockets Layer	27.83 %	46335	57.99 %	32190759	0.279	46277	32187569	0.279
Unreassembled Fragmented Packet	0.03 %	58	0.01 %	3190	0.000	58	3190	0.000
Data	0.02 %	35	0.01 %	7438	0.000	35	7438	0.000
▼ Hypertext Transfer Protocol	0.12 %	194	0.36 %	200841	0.002	132	135238	0.001
Line-based text data	0.02 %	32	0.06 %	32207	0.000	32	32207	0.000
Media Type	0.00 %	6	0.02 %	8934	0.000	6	8934	0.000
CompuServe GIF	0.00 %	1	0.00 %	403	0.000	1	403	0.000
JPEG File Interchange Format	0.00 %	1	0.00 %	1494	0.000	1	1494	0.000

From the security point of view, it will give a high-level glance at all protocols that are happening over the Ethernet system. Network administrators use this information to harden the system configuration; for example, if the administrator found a DCE protocol running in the production system, after seeing this protocol hierarchy he can raise an alarm to stop this service.

Summary

Congratulation on completing this chapter and the book. So far, we have seen how Wireshark helps to analyze network protocols such as TCP/IP, DHCPv6, DHCP, and HTTP. We carried out a detailed analysis of the SSL/TLS protocol and WLAN setup capture; then we explored security-related issues and their mitigation plans. We also tried to be as practical as we can, and provided some real-time use case scenarios and their mitigation plans.

In this book, we have also emphasized other effective tools for capturing the packets, such as tcpdump and snoop. You should now be able to go forward and start analyzing other protocols not covered in this book by using it as a reference.

Index

A

- 802.11 auth process
 - about / [802.11 auth process](#)
- alerts
 - close_notify / [Alert Protocol](#)
 - unexpected_message / [Alert Protocol](#)
 - bad_record_mac / [Alert Protocol](#)
 - decryption_failed / [Alert Protocol](#)
 - record_overflow / [Alert Protocol](#)
 - decompression_failure / [Alert Protocol](#)
 - handshake_failure / [Alert Protocol](#)
 - bad_certificate / [Alert Protocol](#)
 - unsupported_certificate / [Alert Protocol](#)
 - certificate_revoked / [Alert Protocol](#)
 - certificate_expired / [Alert Protocol](#)
 - certificate_unknown / [Alert Protocol](#)
 - illegal_parameter / [Alert Protocol](#)
 - unknown_ca / [Alert Protocol](#)
 - decode_error / [Alert Protocol](#)
 - decrypt_error / [Alert Protocol](#)
 - export_restriction / [Alert Protocol](#)
 - protocol_version / [Alert Protocol](#)
 - insufficient_security / [Alert Protocol](#)
 - internal_error / [Alert Protocol](#)
 - user_canceled / [Alert Protocol](#)
 - no_renegotiation / [Alert Protocol](#)
- ARP duplicate IP detection
 - about / [ARP duplicate IP detection](#)

B

- Berkeley Packet Filter (BPF)
 - about / [The capture filter options](#)
- Bit-Twist
 - URL / [Other packet analyzer tools](#)
- BitTorrent protocol
 - about / [BitTorrent](#)
- BOOTP/DHCP
 - about / [BOOTP/DHCP](#)
 - Wireshark filter / [BOOTP/DHCP Wireshark filter](#)
 - address assignment / [Address assignment](#)
 - capture DHCPv4 traffic / [Capture DHCPv4 traffic](#)

C

- Cain
 - URL / [Other packet analyzer tools](#)
- Capture Options
 - packets, capturing with / [Capturing packets with Capture Options](#)
 - Capture Filter options / [The capture filter options](#)
- client certificate
 - about / [Client certificate](#)
- client certificate request
 - about / [Client certificate request](#)
- Client Hello message
 - about / [Client Hello](#)
 - structure / [Client Hello](#)
 - message / [Client Hello](#)
 - version / [Client Hello](#)
 - random / [Client Hello](#)
 - Session ID / [Client Hello](#)
 - cipher suites / [Client Hello](#)
 - compression methods / [Client Hello](#)
 - extensions / [Client Hello](#)
- Client Key Exchange message
 - about / [Client Key Exchange](#)
- control frames / [Control frames](#)

D

- data frames / [Data frames](#)
- decode-as feature
 - about / [Decode-As](#)
- DHCP/BOOT
 - URL / [References](#)
- DHE/ECHDE traffic
 - decrypting / [Decrypting DHE/ECHDE traffic](#)
 - forward secrecy / [Forward secrecy](#)
- Diffie-Hellman (DHE) key exchange
 - about / [The Diffie-Hellman key exchange](#)
 - naming convention / [The Diffie-Hellman key exchange](#)
 - URL / [The Diffie-Hellman key exchange](#)
- displayed packet
 - exporting / [Exporting the displayed packet](#)
- Display filter references
 - URL / [References](#)
- Distributed Reflection Denial of Service (DrDoS) / [DrDoS](#)
- Domain Name System (DNS)
 - about / [DNS](#)
 - Wireshark filter / [DNS Wireshark filter](#)
 - port / [Port](#)
 - resource records / [Resource records](#)
 - traffic / [DNS traffic](#)
 - URL / [References](#)
- DOS attack
 - about / [The DOS attack](#)
 - SYN flood / [SYN flood](#)
 - Internet Control Message Protocol (ICMP) flood / [ICMP flood](#)
 - SSL flood / [SSL flood](#)
- Dynamic Host Configuration Protocol for IPv6 (DHCPv6)
 - about / [DHCPv6](#)
 - Wireshark filter / [DHCPv6 Wireshark filter](#)
 - multicast addresses / [Multicast addresses](#)
 - UDP port information / [The UDP port information](#)
 - message types / [DHCPv6 message types](#)
 - message exchanges / [Message exchanges](#)
 - traffic capture / [DHCPv6 traffic capture](#)
 - URL / [References](#)

E

- EAPOL / [802.1X EAPOL](#)
- EAP over LAN / [802.1X EAPOL](#)
- Elliptic curve cryptography (ECC) / [Elliptic curve Diffie-Hellman key exchange](#)
- Elliptic curve Diffie-Hellman cryptography (ECDHE) / [Forward secrecy](#)
- Elliptic curve Diffie-Hellman key exchange
 - about / [Elliptic curve Diffie-Hellman key exchange](#)
 - URL / [Elliptic curve Diffie-Hellman key exchange](#)
- Ettercap
 - URL / [Other packet analyzer tools](#)
- Extensible Authentication Protocol (EAP) / [802.1X EAPOL](#)

F

- features, Wireshark
 - decode-as / [Decode-As](#)
 - protocol preference / [Protocol preferences](#)
 - IO graph, using / [The IO graph](#)
 - TCP stream, following / [Following the TCP stream](#)
 - displayed packet, exporting / [Exporting the displayed packet](#)
 - firewall ACL rules, generating / [Generating the firewall ACL rules](#)
- Filter toolbar
 - about / [The Filter toolbar](#)
 - filtering techniques / [Filtering techniques](#)
 - filter examples / [Filter examples](#)
- firewall ACL rules
 - generating / [Generating the firewall ACL rules](#)
- forward secrecy
 - about / [Forward secrecy](#)
 - references / [Forward secrecy](#)
- frames
 - about / [Frames](#)
 - management frames / [Management frames](#)
 - data frames / [Data frames](#)
 - control frames / [Control frames](#)

H

- Heartbleed
 - bug / [Heartbleed bug](#)
 - Wireshark filter / [The Heartbleed Wireshark filter](#)
 - Wireshark analysis / [Heartbleed Wireshark analysis](#)
 - testing / [The Heartbleed test](#)
 - Detector, URL / [The Heartbleed test](#)
 - online test, URL / [The Heartbleed test](#)
 - recommendations / [Heartbleed recommendations](#)
- HTTP
 - about / [HTTP](#)
 - Wireshark filter / [HTTP Wireshark filter](#)
 - use cases / [HTTP use cases](#)
 - URL / [References](#)
- HTTP, use cases
 - top http response time, finding / [Finding the top HTTP response time](#)
 - packets finding, HTTP methods based / [Finding packets based on HTTP methods](#)
 - sensitive information, finding in form post / [Finding sensitive information in a form post](#)
 - HTTP status code, using / [Using HTTP status code](#)
- HTTP protocol preferences
 - about / [Protocol preferences](#)

I

- initial sequence number (ISN) / [Handshake message – first step \[SYN\]](#)
- Interface Lists
 - packets, capturing with / [Capturing packets with Interface Lists](#)
 - interface names / [Common interface names](#)
- Internet Control Message Protocol (ICMP) flood, DOS attack
 - about / [ICMP flood](#)
 - mitigation / [ICMP flood mitigation](#)
- IO graph
 - using / [The IO graph](#)

K

- key exchange
 - about / [Key exchange](#)
- key exchange, types
 - Diffie-Hellman (DHE) key exchange / [The Diffie-Hellman key exchange](#)
 - Elliptic curve Diffie-Hellman key exchange / [Elliptic curve Diffie-Hellman key exchange](#)
 - RSA / [RSA](#)
- KisMac
 - URL / [Wi-Fi sniffing products](#)
- Kismet
 - URL / [Wi-Fi sniffing products](#)

M

- management frames / [Management frames](#)
- Maximum Segment Size (MSS) / [Handshake message – first step \[SYN\]](#)
- medium access control (MAC) layer / [The 802.11 protocol stack](#)
- message exchanges, Dynamic Host Configuration Protocol for IPv6 (DHCPv6)
 - about / [Message exchanges](#)
 - four-message exchange / [The four-message exchange](#)
 - two-message exchange / [The two-message exchange](#)
- message types, Dynamic Host Configuration Protocol for IPv6 (DHCPv6) / [DHCPv6 message types](#)

N

- NetStumbler
 - URL / [Wi-Fi sniffing products](#)
- No-Operation (NOP) / [TCP header fields](#), [Handshake message – first step \[SYN\]](#)

O

- online nmap tool
 - URL / [Vulnerability scanning](#)

P

- 802.11 protocol stack / [The 802.11 protocol stack](#)
- packet analyzer
 - tools / [Other packet analyzer tools](#)
 - mobile packet capture / [Mobile packet capture](#)
- packet analyzers
 - uses / [Uses for packet analyzers](#)
- Packet Bytes pane
 - about / [The Packet Bytes pane](#)
- packet capture process
 - about / [The Wireshark packet capture process](#)
- Packet Details pane
 - about / [The Packet Details pane](#)
- Packet List pane
 - about / [The Packet List pane](#)
- packets
 - capturing / [Guide to capturing packets](#)
 - capturing, with Interface Lists / [Capturing packets with Interface Lists](#)
 - capturing, with Start options / [Capturing packets with Start options](#)
 - capturing, with Capture Options / [Capturing packets with Capture Options](#)
 - file, auto-capturing periodically / [Auto-capturing a file periodically](#)
- PPP (Point-to-Point Protocol) / [802.1X EAPOL](#)
- protocol preference feature
 - about / [Protocol preferences](#)

R

- reset sequence
 - about / [TCP reset sequence](#)
 - RST after SYN-ACK / [RST after SYN-ACK](#)
 - RST after SYN / [RST after SYN](#)
- RFC675 TCP/IP
 - URL / [References](#)
- RFC793 TCP v4
 - URL / [References](#)
- RFMON (Radio Frequency Monitor) mode / [WLAN capture setup](#)
- Riverbed AirPcap adapter
 - URL / [Wi-Fi sniffing products](#)
- RSA / [RSA](#)
- RSA traffic
 - decrypting / [Decrypting RSA traffic](#)

S

- scanning
 - about / [Scanning](#)
 - vulnerability scanning / [Vulnerability scanning](#)
 - SSL scans / [SSL scans](#)
- Scapy
 - URL / [Other packet analyzer tools](#)
- server certificate
 - about / [Server certificate](#)
- Server Hello Done message
 - about / [Server Hello Done](#)
- Server Hello message
 - about / [Server Hello](#)
 - Handshake Type / [Server Hello](#)
 - version / [Server Hello](#)
 - session ID / [Server Hello](#)
 - cipher suite / [Server Hello](#)
 - extensions / [Server Hello](#)
- Server Key Exchange message
 - about / [Server Key Exchange](#)
- snoop tool
 - about / [Tcpdump and snoop](#)
- Snort
 - URL / [Other packet analyzer tools](#)
- SSL-related issues
 - debugging / [Debugging issues](#)
- SSL/TLS
 - about / [An introduction to SSL/TLS](#)
 - benefits / [An introduction to SSL/TLS](#)
 - versions / [SSL/TLS versions](#)
 - components / [The SSL/TLS component](#)
 - handshake / [The SSL/TLS handshake](#)
 - decrypting / [Decrypting SSL/TLS](#)
 - RSA traffic, decrypting / [Decrypting RSA traffic](#)
 - DHE/ECHDE traffic, decrypting / [Decrypting DHE/ECHDE traffic](#)
- SSL/TLS handshake
 - about / [The SSL/TLS handshake](#)
 - types / [Types of handshake message](#)
 - Client Hello message / [Client Hello](#)
 - Server Hello / [Server Hello](#)
 - server certificate / [Server certificate](#)
 - Server Key Exchange message / [Server Key Exchange](#)
 - client certificate request / [Client certificate request](#)

- Server Hello Done message / [Server Hello Done](#)
- client certificate / [Client certificate](#)
- Client Key Exchange message / [Client Key Exchange](#)
- Client Certificate Verify message / [Client Certificate Verify](#)
- Change Cipher Spec record type / [Change Cipher Spec](#)
- Finished message / [Finished](#)
- Application Data message / [Application Data](#)
- Alert Protocol / [Alert Protocol](#)
- SSL flood, DOS attack
 - about / [SSL flood](#)
- SSL testing
 - references / [Debugging issues](#)
- Start options
 - packets, capturing with / [Capturing packets with Start options](#)
- Stumbler
 - URL / [Wi-Fi sniffing products](#)
- Switch Port Analyzer (SPAN) port / [The Wireshark packet capture process](#)
- SYN flood, DOS attack
 - about / [SYN flood](#)
 - mitigation / [SYN flood mitigation](#)

T

- TAP (Test Access Point) / [The Wireshark packet capture process](#)
- TCP analyze sequence numbers
 - URL / [References](#)
- TCP CLOSE_STATE
 - about / [How to resolve TCP CLOSE_STATE](#)
- TCP CLOSE_WAIT
 - about / [TCP CLOSE_WAIT](#)
- TCP display filter
 - reference link / [Filter examples](#)
- tcpdump tool
 - about / [Tcpdump and snoop](#)
- TCP Dup-ACK
 - about / [TCP Dup-ACK](#)
- Tcpreplay
 - URL / [Other packet analyzer tools](#)
- TCP stream
 - following / [Following the TCP stream](#)
- TCP TIME_WAIT
 - about / [TCP TIME_WAIT](#)
- TCP Window Update
 - about / [TCP Window Update](#)
- three-way handshake, Transmission Control Protocol (TCP)
 - about / [TCP three-way handshake](#)
 - first step [SYN] / [Handshake message – first step \[SYN\]](#)
 - second step [SYN, ACK] / [Handshake message – second step \[SYN, ACK\]](#)
 - third step [ACK] / [Handshake message – third step \[ACK\]](#)
- TLS extensions
 - reference list / [Client Hello](#)
- Transmission Control Protocol (TCP)
 - about / [Recapping TCP](#)
 - header fields / [TCP header fields](#)
 - states / [TCP states](#)
 - connection establishment / [TCP connection establishment and clearing](#)
 - three-way handshake / [TCP three-way handshake](#)
 - data communication / [TCP data communication](#)
 - close sequence / [TCP close sequence](#)
 - Wiki, URL / [References](#)
 - TCP/IP guide, URL / [References](#)
- Transmission Control Protocol (TCP), latency
 - issues / [TCP latency issues](#)
 - identifying / [Identifying latency](#)
 - server latency example / [Server latency example](#)

- wire latency / [Wire latency](#)
- Transmission Control Protocol (TCP), latency issues
 - causes / [Cause of latency](#)
- Transmission Control Protocol (TCP), troubleshooting
 - about / [TCP troubleshooting](#)
 - reset sequence / [TCP reset sequence](#)
 - CLOSE_WAIT / [TCP CLOSE_WAIT](#)
 - TIME_WAIT / [TCP TIME_WAIT](#)
- troubleshooting
 - packets, capturing / [Troubleshooting](#)

U

- US-CERT
 - alert TA14-017A, URL / [DrDoS](#)
- user interface, Wireshark
 - about / [Wireshark user interface](#)
 - Filter toolbar / [The Filter toolbar](#)
 - Packet List pane / [The Packet List pane](#)
 - Packet Details pane / [The Packet Details pane](#)
 - Packet Bytes pane / [The Packet Bytes pane](#)

W

- Wi-Fi networks
 - analyzing / [Analyzing the Wi-Fi networks](#)
 - frames / [Frames](#)
 - 802.11 auth process / [802.11 auth process](#)
 - 802.1X EAPOL / [802.1X EAPOL](#)
 - 802.11 protocol stack / [The 802.11 protocol stack](#)
- Wi-Fi sniffing products
 - about / [Wi-Fi sniffing products](#)
 - Kismet / [Wi-Fi sniffing products](#)
 - Riverbed AirPcap / [Wi-Fi sniffing products](#)
 - KisMac / [Wi-Fi sniffing products](#)
 - Stumbler / [Wi-Fi sniffing products](#)
 - NetStumbler / [Wi-Fi sniffing products](#)
- WireEdit
 - URL / [Other packet analyzer tools](#)
- Wireshark
 - about / [Introducing Wireshark](#)
 - URL / [Introducing Wireshark](#), [References](#)
 - features / [Wireshark features](#), [Wireshark features](#)
 - dumpcap / [Wireshark's dumpcap and tshark](#)
 - tshark / [Wireshark's dumpcap and tshark](#)
 - packet capture process / [The Wireshark packet capture process](#)
 - wiki link / [802.1X EAPOL](#)
- Wireshark community
 - URL / [Troubleshooting](#)
- Wireshark protocol hierarchy
 - about / [Wireshark protocol hierarchy](#)
- Wireshark TCP sequence analysis
 - about / [Wireshark TCP sequence analysis](#)
 - retransmission / [TCP retransmission](#)
 - TCP ZeroWindow / [TCP ZeroWindow](#)
- WLAN capture setup
 - about / [WLAN capture setup](#)
 - multi-channel captures, URL / [WLAN capture setup](#)
 - wireless network interface controller (WNIC) / [WLAN capture setup](#)
 - AP (Access Point) / [WLAN capture setup](#)
 - monitor mode / [The monitor mode](#)

X

- 802.1X EAPOL / [802.1X EAPOL](#)