Wireshark Fundamentals

A Network Engineer's Handbook to Analyzing Network Traffic

Vinit Jain

Apress®

Wireshark Fundamentals

A Network Engineer's Handbook to Analyzing Network Traffic

Vinit Jain

Apress®

Wireshark Fundamentals: A Network Engineer's Handbook to Analyzing Network Traffic

Vinit Jain
San Jose, CA, USA

ISBN-13 (pbk): 978-1-4842-8001-0 ISBN-13 (electronic): 978-1-4842-8002-7
https://doi.org/10.1007/978-1-4842-8002-7

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Aditee Mirashi
Development Editor: Laura Berendson
Coordinating Editor: Aditee Mirashi

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-8001-0. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

I would like to dedicate this book to my late brother Lalit Jain (Dada Bhai), who was and will always be an inspiration in my life. You always treated me as your son and gave me so much love my whole life. Dada Bhai, I will always miss our conversations, your jokes and smile even during the tough moments, your love, and the bonding we shared. I didn't even get a chance to hug you and bid you a goodbye. It will always be the biggest regret of my life that I wasn't there when you needed me the most. I wish I was there to save you. With you gone it feels like I have lost the roof above my head for the rest of my life. I know I have lost a gem of a brother in this life but hopefully we will meet again in the life beyond this one. I just want to let you know that I will always love you and will keep doing my best to reach the heights that you wanted me to. May your soul rest in peace.

Table of Contents

About the Author

Vinit Jain is a Senior Technical Leader for Network Engineering at Cisco, focusing on architecting network infrastructure for edge computing solutions. Prior to that, he worked as a Network Development Engineer at Amazon as part of Amazon's backbone network operations team and as a technical leader at Cisco Technical Assistance Center, providing escalation support in enterprise, service provider, and data center technologies.

Vinit is a speaker at various networking forums, including Cisco Live events, NANOG, and CHINOG. He has coauthored several Cisco Press books and video courses with Cisco Press. Vinit holds a bachelor of arts degree in mathematics from Delhi University and also holds a master of science in information technology. Apart from CCIE, he also holds multiple certifications in programming, database, and system administration and is also a Certified Ethical Hacker. Vinit can be found on Twitter: @vinugenie.

About the Technical Reviewers

Carsten Thomsen is primarily a back-end developer, but he works with smaller front-end bits as well. He has authored and reviewed a number of books, and created numerous Microsoft Learning courses, all to do with software development. He works as a freelancer and contractor in various countries in Europe, using Azure, Visual Studio, Azure DevOps, and GitHub among other tools. Being an exceptional troubleshooter, asking the right questions, including the less logical ones, in a most logical to least logical fashion, he also enjoys working with architecture, research, analysis, development, testing, and bug fixing. Carsten is a very good communicator with great mentoring and team-lead skills, and great aptitude for researching and presenting new material.

Shyam Sundar Ramaswami is a Senior Research Scientist. He is also a two-time TEDx speaker. In addition, he enjoys being a digital detective. He loves security problems because they are easy, complex, and fun to solve. He also enjoys creating security prototypes and malware analysis.

His goals include guarding domains and predicting the possible threats that might hit the cyberworld by analyzing various scenarios, re-creating attack scenes, and educating the world on various attacks.

Acknowledgments

A special thanks to Aditee Mirashi for encouraging me to work on this project. Aditee, this book would not have been possible without your patience and help. You have helped me during this project in so many ways for which I shall always be thankful. I would also like to thank the technical reviewers, Shyam Sundar Ramaswami and Carsten Thomsen, for their valuable inputs and in-depth verification of the content. Their insightful feedback has made this project a successful one. Last but not least, I would like to thank Apress team for all their hard work and support on this project.

Introduction

Wireshark is one of the most sought out tools among network engineers and network security analysts. Wireshark gives engineers the capability to analyze network traffic by expanding through each section of a header and examining its value. Wireshark not only helps with network traffic analysis; it is also a critical tool when it comes to understanding or learning a network protocol or feature. Performing packet capture and analyzing network traffic can be complex, time-consuming, and tedious tasks. With the help of this book, users will be able to use Wireshark to its full potential and become expert at analyzing network traffic and more efficient at solving complex network problems.

This book helps build a strong foundation for how Layer 2, Layer 3, and Layer 4 traffic behaves and how various routing protocols and overlay protocols function, as well as an understanding of their packet structure. This book is a very useful handbook for troubleshooting engineers who want to analyze traffic to identify issues in the network, such as issues related to packet loss, bursty traffic, and so on. This book will help you get started on the journey of becoming a strong network engineer or a cybersecurity expert.

CHAPTER 1

Introduction to Wireshark

This chapter covers the following topics:

- Introduction to network traffic analysis
- Overview of Wireshark
- Installing Wireshark
- Setting up port mirroring

Introduction to Network Traffic Analysis

Modern-day networks and network designs are complex. A network is a graphical representation of how different elements in the network (nodes) are connected. Because every business or organization has its own set of network requirements, network architects come up with designs and solutions that are best suited for the given business requirements. The network design differs between enterprise, service provider, and datacenter networks in various aspects such as scale, redundancy, security, and so on. A few factors are usually considered when designing a network:

V. Jain, *Wireshark Fundamentals*, https://doi.org/10.1007/978-1-4842-8002-7_1

- *Simple:* A network design should be simple. Most practitioners in the network field are familiar with the KISS principle: Keep It Simple, Stupid. A network is dependent on various technologies, protocols, hardware and software resources, and so on. Even though each of these components might be simple individually, their combination in a network will always add to the complexity. Identifying problems in large-scale networks is often like finding a needle in a haystack. It thus becomes more important that the network design and architecture are kept as simple as possible.
- *Highly available:* Almost every network is designed to carry traffic for critical business applications, and a small network event could have a massive impact on the services provided by an organization. Thus, it is important to build redundancy into the network such that in the case of a failure event, the availability of services is maintained. Although redundancy in a network is vital, it is equally important to understand and define how much redundancy is acceptable. More redundant paths in the network result in higher costs.
- *Robust:* As stated in the IEEE document "Robust Network Design," robustness is defined as minimizing variations in network performance, such as average delay and throughput, due to perturbations in the network like topology, demand, and community of interest. If a network design as well as its requirements are not thought through in depth, the network will be affected with increased average delays and throughput or performance issues over a period of time.

- *Scalable:* Because the scale of the application and user traffic are constantly growing, the ability to scale the existing network infrastructure holds great importance. While designing computer networks, it is important to choose designs that will allow you to scale the network horizontally as well as vertically as and when required. In other words, the network design should allow the organization to scale the network for east–west as well as north–south traffic on an on-demand basis. One such example is the Clos network design, which primarily focuses on the spine–leaf architecture and has been widely adopted by large-scale datacenter networks to help them cater to the increased traffic demand over the years.
- *Futuristic:* In designing networks, it is also important to choose the right set of hardware and software. The right hardware and software choices will allow you to leverage the latest technologies and current innovations in the network industry and will enable you to upgrade the firmware or the network operating system to access the latest available features.

Even when all of these factors are taken into consideration while designing a network, every network still has to undergo changes to overcome dynamic application and resource demands. In addition, every network, irrespective of how carefully it has been designed, is prone to network issues. As the network grows, its complexity also grows. It is only a matter of time before a network problem arises and network engineers are called on to solve the problem. Network problems are usually difficult to manage, and the complexity is even greater when we have multiple features and encapsulations being used in the network.

The main goal of the network operations team is to keep the network as stable as possible and mitigate any problems as quickly as possible while keeping the blast radius of the event to a minimum. Most network outages can be quickly mitigated by following one of several techniques:

- Shutting down a bad or faulty link
- Rebooting or shutting down a network device (e.g., router, switch, firewall, etc.)
- Diverting traffic by adjusting the routing metrics
- Flapping a routing adjacency

Some issues could take longer time to resolve, especially when it comes to software or hardware defects. Among the different network-related issues faced by network engineers, issues such as continuous or intermittent packet loss, latency, or routing or switching issues require a deeper analysis. Other issues can be quickly mitigated by replacing the hardware or cable or a particular port, for example, but issues related to packet loss or routing problems involve multiple elements in the network that cannot be identified and mitigated quickly. Such issues sometimes require more visibility at the packet level as to what data are being transmitted on the wire. That said, it is now time to understand what network analysis is.

Network traffic analysis (NTA), often referred to as packet sniffing, is the process of collecting (capturing) network traffic and monitoring network activity and events by examining the collected traffic to identify anomalies in the network, including but not limited to operational issues such as packet loss or latency and security issues such as Transmission Control Protocol (TCP) SYN attacks or man-in-the-middle attacks. There are always situations in the network where the show commands or debugs from the network operating system do not yield the actual packet-level information that is being transmitted from one node to another. A tap in the wire allows the network administrators to gain a clear understanding of

what packets are being transmitted across the network elements. There are several use cases where NTA techniques could be applied:

- Understanding network characteristics
- Analyzing protocol behaviors
- Troubleshooting slowness in the network
- Troubleshooting packet forwarding issues
- Identifying vulnerabilities in the network protocols and ciphers
- Identifying malicious activities in the network
- Collecting real-time information on activities between different network elements

Better visibility into the network allows the network administrators to optimize performance, enhance the security posture of the network, minimize the blast radius of a network attack, and better analyze the utilization of network resources. The packets collected in the network also give network administrators a better understanding of how the network users are implementing their applications. Techniques such as deep packet inspection (DPI) allow complete network visibility by transforming the raw packet data as well as metadata into a readable format.

Typically, packet analysis or packet sniffing is performed by a packet sniffer, a tool that is used to capture raw network traffic going across the wire (network). There are several tools available, including the free or commercial ones, based on the command-line interface (CLI) as well as the graphical user interface (GUI). These are some of the most popular packet sniffer tools:

- *Tcpdump:* This is a powerful CLI-based packet analyzer tool that is freely available and runs on Linux or most UNIX-like operating systems.

- *Omnipeek:* This is a GUI-based commercial packet analyzer tool from Savvius, a LiveAction company.
- *Wireshark:* Wireshark is a free, open source, GUI-based packet analyzer available for download on various operating systems.

Note In this book, we primarily focus on Wireshark. Covering different network analyzer tools is outside the scope of the book.

Network Sniffing

As easy as it sounds, network sniffing is actually not easy. Most network engineers think that packet sniffing involves simply plugging a laptop into a network port and capturing traffic, but that is not the case. There are various factors to be considered when tapping into the wire and capturing traffic. Two factors play a vital role when sniffing network traffic:

1. Placement of the sniffer in the network
2. The number of sniffer placements

We discuss both these points in detail.

Sniffer Placement

Location of the sniffer placement varies based on the network topology. Different network topologies have varied requirements and complexities and identifying the point of sniffer placement in such complex environments is not easy. As we know, networks include different elements, such as routers, switches, wireless controllers, firewalls, and so on. Some of these components might not even have support for enabling traffic mirroring for sniffing purposes. Traffic mirroring on switches can be

enabled using a technique known as port mirroring. *Port mirroring* or *port spanning* can be configured on the switch using a CLI command or a web management interface of the switch. The best feature of port mirroring is that it leaves no network footprint and does not generate any additional packets. It can be configured without taking any of the active traffic interfaces or host interfaces offline, which makes it an ideal option for mirroring not just switch traffic, but also router or server and host ports. As part of the configuration, you define the source interface(s) along with the direction of the traffic (incoming or outgoing) you want to capture and a destination interface that is connected to a host with a packet capture tool, such as Wireshark, installed on it to collect all the mirrored packets that can later be used for analysis. Once this is set up, the mirrored traffic from all the source ports is sent to a host connected on the destination port. Figure 1-1 illustrates how port mirroring works. In this given topology, traffic sourced from host A and destined for host B is mirrored and sent to the capture device connected on Eth2/0.

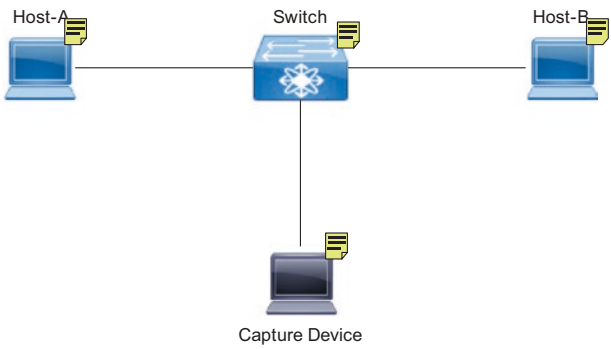


Figure 1-1. *Port mirroring*

For a capture device to be able to capture packets, the network interface card (NIC) should support promiscuous mode. A promiscuous mode driver allows a NIC to view all packets crossing the wire. When tools

such as Wireshark are installed on the capture device, they also install a libpcap or WinPcap driver on the device. These drivers allow the NIC to switch to promiscuous mode and capture packets across the network.

Now, we return to the question of what the right place is to locate the sniffer capture in the network. The simple answer is that it depends on the troubleshooting being performed and the relevant network topology between the problematic source and destination. First of all, it should never be a goal of any kind of network troubleshooting to begin with placing sniffer captures at multiple places in the network. The goal should always be to isolate and narrow down the problem as much as possible. In narrowing down, you might find that you do not need to place sniffer captures at all. To troubleshoot a network problem, it is important to first understand the problem and its scope. Unless a problem is clearly understood, troubleshooting cannot move in the right direction and it could take more time and effort to mitigate or resolve a problem. Second, as part of the problem statement, the scope information should also be gathered to understand the blast radius of the problem. In other words, the scope of the network problem can be identified by asking few simple questions such as these:

- When did the problem start?
- What is the problematic source and destination?
- What is the working source and destination?
- What is the relevant topology between the source and destination?
- How many users and services are affected?
- What was the trigger of the problem?

Once the problem statement and the scope are clearly defined, the next step is to isolate the direction of the problem. For instance, let's say there is a complete packet loss between hosts A and B as the traffic is flowing across multiple network devices. To isolate the direction of the problem, we need to identify if the problem is when the packet is sent from A to B or in the reverse direction. It could be, based on the way the network devices are configured, that the traffic from host B to host A might not flow via the same set of devices that it took when flowing from host A to host B. This is known as asymmetrical routing. If direction of the problem is not identified, we would end up placing sniffer captures randomly across multiple devices in the network, which consumes more time. Once the direction of the problem is isolated, then it is important to further narrow down the problem to a minimum set of devices or even interfaces. To understand this in detail, examine the topology shown in Figure 1-2. The topology shows the connectivity between two sites of an enterprise network. In this topology, each site has access, distribution, core, and wide area network (WAN) layers. The WAN routers are connected to the Internet service providers (ISPs) that provide connectivity to the remote site across the Internet. In this topology, the distribution switches, the core switches, and the WAN routers are all connected via an interior gateway protocol (IGP). The WAN routers at Site 1 are connected to the remote site WAN routers via Border Gateway Protocol (BGP) peering toward the ISP, which is exchanging certain prefixes along with a default route. The BGP prefixes are then redistributed into the Open Shortest Path First (OSPF) database at both sides to provide end-to-end connectivity between both sites.

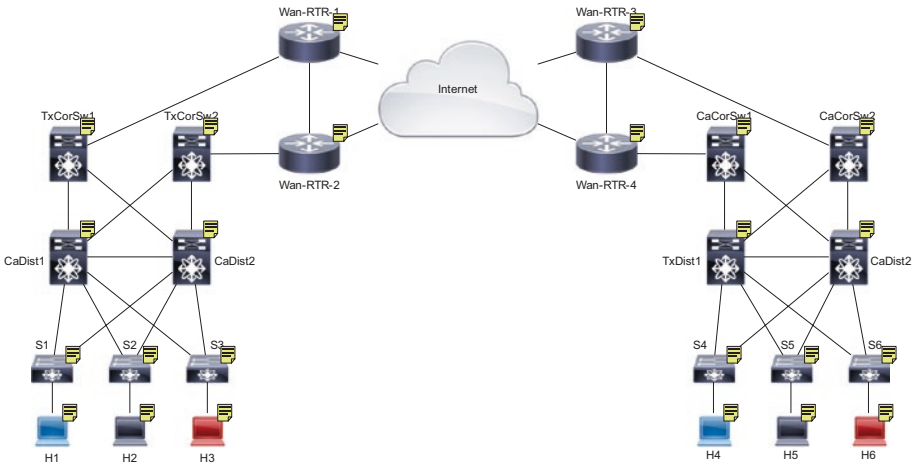


Figure 1-2. *Enterprise network topology*

Now, let's examine a scenario where the network operations team has reported a problem, stating that host H1 is having connectivity issues reaching multiple hosts across the remote site. As a network engineer, you start isolating the problem scope by asking few basic questions as stated previously. Let's assume that host H1 is having issues reaching host H4 and host H5, but not host H6. Host H1 is not having any issues reaching any of the hosts that are local to the site. Now you have baseline information on what is working and what is broken. As a next step in troubleshooting, there are a few simple steps that you can take:

- *Verify Address Resolution Protocol (ARP) information:* Verify if the ARP entry is complete for host H4 or H5 versus host H6.
- *Traffic pattern:* Verify if the issue is sending broadcast traffic or unicast traffic. If the ARP table on host H1 shows that the ARP entry is present or is getting completed for host H4 and H5 even after clearing the

ARP, then it might be a problem just with the unicast traffic instead of all traffic (broadcast, unicast, or multicast traffic) between H1 and H4 or H5.

- *Path information:* Perform traceroutes between working and nonworking hosts to identify any difference in paths taken by each of them. If the network has equal cost multiple paths (ECMP), then most routing and switching platforms perform flow-based hashing to send traffic out on one of the interfaces. If the traceroute fails at one of the hops in the path, that would indicate the problem might be isolated to that segment of the network. Note that it is important to perform traceroutes from both endpoints so that any possibility of asymmetrical routing could be detected.
- *Access control lists (ACLs):* Leverage ACLs whenever possible to isolate where the traffic loss is happening. Users can configure both Layer 3 ACLs (standard and extended) and Layer 2 ACLs (media access control [MAC] ACLs) to match Layer 3 as well as Layer 2 traffic at different segments of the network. However, there could be instances where ACLs might not be of any help. For instance, ACLs do not allow users to capture Multiprotocol Label Switching (MPLS) packets. Thus, it becomes important to identify the kind of packets being investigated during troubleshooting.

- *Hop-by-hop ping tests:* If a traceroute fails at a segment of the network, it might make sense to check the reachability of the source device to that segment of the network. It is possible that only the transit traffic might be affected and not the traffic destined for those devices in the segment. This usually happens if there is an ACL blocking the traffic in the path or due to a software misprogramming (software defect). In such instances, ping tests should be performed before performing deep-dive troubleshooting.
- *Platform troubleshooting tools:* Most routing and switching platforms come with troubleshooting tools as part of the network operating system (OS). These platform troubleshooting tools can help you understand if a packet is being dropped on the device itself or not and why. These tools are primarily helpful when the issue has been isolated to a particular device or a network segment. Note that some network OS's come with platform-level packet capture tools. These tools can be very useful to perform packet captures to understand if the packet is being received on the device or not and if what action is being taken on the packet by the network OS.
- *Debugs:* It might sometimes help to run debugs on the network devices. The debug logs allow you to gain more insight on what is happening on the network device. For instance, if a BGP prefix is not being received, you could run a BGP protocol debug to understand if the prefix is being received or not.

Once all the basic and some advanced-level troubleshooting steps have been performed and you are unable to isolate the issue to one particular device or segment, that's where external sniffer capture tools come into play. Here are some of the scenarios in which sniffer capture can be useful:

- When dealing with corrupted packets
- Gathering more information about the packet headers, as they might be affecting forwarding decision
- Troubleshooting encapsulated packets
- Troubleshooting packet loss or retransmission issues
- Voice or video traffic-related issues
- Protocol issues such as OSPF not forming adjacency due to wrong information being exchanged or BGP not establishing peering due to TCP or wrong or missing information in BGP packets

If we talk about the problem displayed in Figure 1-2, if the traffic loss is happening between host H1 and host H4 or host H5, then some of the preceding steps could be followed to isolate the problem to a smaller segment of the network and sniffer devices could be attached in those segments to further investigate the issue. For instance, if the issue was narrowed down between the site 2 WAN router and Distribution layer switch(es), then enabling port spanning on core switches will help identify if the issue lies on the WAN router, the core switch, or the distribution switch. If the packet sniffers are to be used to analyze the problem, however, then the network engineers would have to enable the sniffer captures at the following devices:

- Access switch connected to the source and destination host

- Core switch(es) at each site
- WAN routers (if they support enabling port spanning)

The captures taken at each site can easily help determine where the packet loss is happening. Even though the sniffer captures help a lot in investigating the issue, that is not the final step of the troubleshooting process. There are a few more steps that are involved in mitigating and remediating the problem, which we will see in the coming chapters based on different problem scenarios.

Number of Sniffer Placements

Placing sniffers is not always easy. Each organization, be it an enterprise, service provider, or datacenter, has its own set of policies for managing and troubleshooting in its network environment. Most organizations require scheduling a change and maintenance window to perform troubleshooting, let alone performing sniffer captures. Also, when the whole network environment that is under investigation is geographically displaced or made up of remote unmanned sites, it takes a while to get field engineers on site to help with sniffer captures. Further, when the sniffer captures are to be performed at multiple places in the network, the complexity is compounded. When the troubleshooting requires sniffer captures in the network, it is important that the points of placement should be carefully considered before actually enabling port spanning.

Some network environments are also set up in a way that supports remote spanning with specific hosts configured to collect mirrored traffic. Such network deployments allow users to perform port spanning at almost any given node in the network without having to wait for any human to be present on site. The only limitation of such deployments is either the support for the remote span feature on all network devices or the host or the switch performance with higher throughput interfaces.

In the example discussed in Figure 1-2, the ideal process would be to isolate the segment where the problem is and then place sniffer captures in that segment. If there are issues related to voice traffic such as users facing choppy voice or even TCP retransmission issues, it would require sniffer placements at multiple points across the network to determine where the issue is actually happening. For such a huge span of segments to troubleshoot, the approach for performing packet captures should be to isolate between the internal network versus the ISP network. For instance, the sniffer placements between the access and core or WAN layer at each site will allow us to identify if the issue is local to any of the two sites. If the packet sent from one site is not received by the WAN router on the remote site, that means the issue would be isolated to the ISP network instead of the site local networks.

Network Tap

A network tap is a hardware device that creates a copy or mirror of the traffic flowing between two points in your cabling system. The hardware is specially designed for network analysis. When setting up network taps, the hosts or network devices might be temporarily offline. The network taps can be useful in enterprises for performing packet captures and continuous monitoring, as they are reliable and support high-throughput links. There are two primary types of network taps:

- *Aggregated:* The aggregated network taps allow bundling of multiple streams of data across multiple ports to one monitoring port. This type of network tap is useful when it is required to monitor bidirectional streams of traffic but only one NIC for monitoring.

- *Nonaggregated:* The nonaggregated network taps provide additional flexibility for capturing traffic but also add to complexity when compared to aggregated network taps. In nonaggregated network taps, two ports are required for monitoring purposes, each of them capturing traffic in only one direction.

Based on the monitoring requirements, the choice can be made between aggregated and nonaggregated network taps.

So far, we have learned about the port spanning and network taps that can be used to enable and perform packet captures in the network. Next, we learn about the Wireshark tool that will be used for analyzing the captured traffic.

Overview of Wireshark

Wireshark is a widely used open source network protocol analyzer. The first version of the application was called Ethereal and was developed and released by Gerald Combs in 1998 under the GNU Public License (GPL). After some conflicts over the Ethereal brand rights with his employer, Combs, along with the rest of the development team, rebranded the project as Wireshark in mid-2006. Wireshark is freely available for personal, educational, and commercial purposes and is supported and maintained by a community of more than 1,800 developers.

It is the go-to tool for almost every network administrator or network engineer to analyze network traffic patterns, troubleshoot network protocol issues, and perform in-depth analysis of network security loopholes. Wireshark comes with tons of features, supports the most common and uncommon set of protocols and encapsulations, and is supported on all the well-known OSs. It provides an easy-to-use and easy-to-understand GUI and advanced filtering capabilities to search through millions of packets to allow network administrators to quickly analyze the events in the network.

Installing Wireshark

At the time of writing, the latest and stable version of Wireshark is 3.4.4. Wireshark installer is available in both 32-bit and 64-bit versions and have builds available for Windows, Mac, and various Linux OSs. Wireshark installer can be downloaded from https://www.wireshark.org/download.html. Installation of Wireshark is fairly simple. In the section, we cover the installation of Wireshark on different OSs.

Installing Wireshark on Windows

Follow these steps to install Wireshark on Windows:

- Download the installer (.exe file) from https://www.wireshark.org.
- Double-click the installer to begin the installation process.
- Click Next to begin the installation.
- Acknowledge the License Agreement by clicking Noted.
- Select the components that you want to install as shown in Figure 1-3 and click Next.

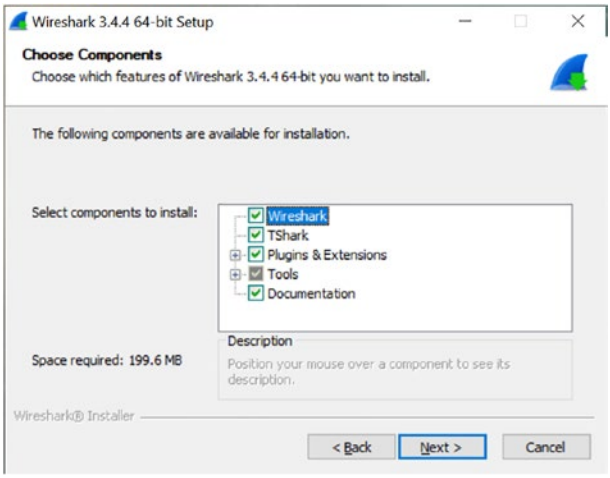


Figure 1-3. *Wireshark Installation Choose Components screen*

Note In the component selection you can see an option to install TShark. TShark is a CLI version of Wireshark, which is designed to capture and analyze network traffic. It supports the same options as Wireshark. To view all the options of TShark, use the command `man tshark` or `tshark --help` option.

- Select the different shortcuts that you want to place on your PC and click Next.
- Select the installation directory for Wireshark and click Next.
- Select the Npcap or WinPcap version that is currently installed or is available to install and click Next. Note that Npcap or WinPcap is required by Wireshark to capture live network packets.

- Optionally, you can install USBPcap to capture USB traffic and click Install to begin the installation process.
- During the installation, another installer window will open for Npcap or WinPcap software. Select the necessary installation options and begin the installation process by clicking Install.
- Once the installation completes, click Finish.
- The Wireshark installation will continue further.
- Once the installation process is completed, click Finish. At this point, Wireshark is now ready to perform packet captures on your system.

Installing Wireshark on Mac

Follow these steps to install Wireshark on MacOS:

- Download the installer (`.dmg` file) from https://www.wireshark.org.
- Double-click the installer file to begin the extraction process.
- The extraction process will create a volume with all the necessary files on the desktop.
- Once the extraction is completed, a pop-up window gives you an option to move the Wireshark app into the Applications directory
- Drag the Wireshark app into the Applications directory to make Wireshark accessible from the launch pad.

Installing Wireshark on Ubuntu

Wireshark can be installed quickly on Ubuntu from the terminal using the apt-get package installer. Follow these steps to install Wireshark on Ubuntu:

- Update the repository on the Ubuntu machine using the command `apt update`.
- Install Wireshark using the command `apt install wireshark`.

If you just have CLI access to the Ubuntu server or machine, then it might be a better option to install TShark. Users can install TShark using the command `apt install tshark` as shown in Example 1-1. Once installed, users can review the CLI options for TShark using the `tshark` command with the `--help` option or the `man tshark` command.

Example 1-1. Installing tshark on Ubuntu

```
root@genie-rnd-server:~# apt install tshark
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  tshark
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 137 kB of archives.
After this operation, 411 kB of additional disk space will
be used.
Get:1 http://us.archive.ubuntu.com/ubuntu focal/universe amd64
tshark amd64 3.2.3-1 [137 kB]
Fetched 137 kB in 0s (323 kB/s)
Selecting previously unselected package tshark.
```

```
(Reading database ... 279306 files and directories currently
installed.)
Preparing to unpack .../tshark_3.2.3-1_amd64.deb ...
Unpacking tshark (3.2.3-1) ...
Setting up tshark (3.2.3-1) ...
Processing triggers for man-db (2.9.1-1) ...

root@genie-rnd-server:~# tshark --help
Running as user "root" and group "root". This could be dangerous.
TShark (Wireshark) 3.2.3 (Git v3.2.3 packaged as 3.2.3-1)
Dump and analyze network traffic.
See https://www.wireshark.org for more information.

Usage: tshark [options] ...

Capture interface:
  -i <interface>, --interface <interface>
                           name or idx of interface (def: first
                           non-loopback)
  -f <capture filter>      packet filter in libpcap
  filter syntax
  -s <snaplen>, --snapshot-length <snaplen>
                           packet snapshot length (def:
                           appropriate maximum)
  -p, --no-promiscuous-mode
                           don't capture in promiscuous mode
  -I, --monitor-mode       capture in monitor mode, if available
  -B <buffer size>, --buffer-size <buffer size>
                           size of kernel buffer (def: 2MB)
  -y <link type>, --linktype <link type>
                           link layer type (def: first
                           appropriate)
<snip>
```

Once set up, the GUI-based Wireshark app or CLI-based tshark app can be used to capture traffic traversing the network.

It is important to learn how to capture the packets and analyze the network traffic, but it is equally important to know the tools available with different network devices that you can use to set up packet captures.

Setting Up Port Mirroring

As we all know, there are multiple elements in the network such as routers, switches, firewalls, load-balancers, servers, and so on, and troubleshooting a network with multiple elements involved can be complex. Packet capture tools are very handy when investigating issues at the packet level. When a deep-dive investigation is required at the packet level, the issue is usually found in one of the following places:

- At the transmitting device or the device that initiated the packet
- At the receiving device or the device for which the packet is destined
- At the transit device
- In the transmission media

The packet-level issues require the network engineers to perform packet captures and investigate the issues by analyzing the captured traffic. In most cases, switching devices have the capability of mirroring network traffic and sending it to a mirroring port that is connected to a PC. Let's see how port mirroring can be enabled on different vendor devices.

SPAN on Cisco IOS/IOS-XE

The port mirroring capability on Cisco devices is known as *Switched Port Analyzer* (SPAN). SPAN can be set up on both Layer 2 and Layer 3 interfaces. When setting up SPAN, the source and the destination interfaces are defined. Source ports are a collection of physical ports such as Gigabit Ethernet or TenGig interfaces and virtual interfaces such as virtual local area network (VLAN) switch virtual interfaces (SVIs). In defining the source ports, users can also define the direction of the traffic; that is, `rx` for incoming direction, `tx` for outgoing direction, or both, which means mirror both `rx` and `tx` traffic. A SPAN session on a Cisco IOS or Cisco IOS-XE switch can be configured using the command monitor session session number. Under the SPAN session configuration, you can define the source and the destination ports along with their direction. Example 1-2 illustrates how to configure a SPAN session on a Cisco IOS-XE switch.

Example 1-2. Configuring SPAN

```
SW1#configure terminal
SW1(config)# monitor session 10 source interface
GigabitEthernet1/5 rx
SW1(config)# monitor session 10 source interface
GigabitEthernet1/7 tx
SW1(config)# monitor session 10 destination interface
GigabitEthernet2/1
SW1#configure terminal
SW1(config)# monitor session 11 source vlan 3 - 5 rx
SW1(config)# monitor session 11 source vlan 20
SW1(config)# monitor session 11 destination interface
GigabitEthernet2/2
```

Note As per the Cisco.com documentation, you cannot have two SPAN sessions using the same destination port.

Once the SPAN is set up, you can view the state of the SPAN using the command `show monitor session [session-number]`. If the command is executed without specifying the `session-number` element, it displays all the SPANs that are configured on the switch. When the `session-number` option is specified, the command displays information about only the specified session. It is important to note that the destination interface is the one running in promiscuous mode. Thus, no other protocol or feature will work on that port. Example 1-3 displays how to verify the SPAN session and also displays the destination interface state in monitoring state. The monitoring state indicates that the port is running in promiscuous mode.

Example 1-3. SPAN Session Verification

```
SW1#show monitor session 10
Session 10
---------
Type                   : Local Session
Source Ports           :
    rx                 : Gi1/5
    tx                 : Gi1/7
Destination Ports      : Gi2/1
MTU                    : 1464

Egress SPAN Replication State:
Operational mode       : Distributed
Configured mode        : Distributed
SW1#show interface GigabitEthernet1/1
GigabitEthernet2/1 is up, line protocol is down (monitoring)
```

```
  Hardware is Gigabit Ethernet, address is 2c54.2d68.1207 (bia
  2c54.2d68.1207)
  MTU 1998 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 18/255
  Encapsulation ARPA, loopback not set
  Keepalive not set
  Full-duplex, 1000Mb/s, link type is auto, media type is
  10/100/1000BaseTX SFP
<snip>
```

Note Refer to the Cisco online product documentation to verify how many SPAN sessions can be configured. The supported number of SPAN sessions varies from platform to platform and from vendor to vendor.

SPAN on Cisco Nexus Switches

The Cisco Nexus OS (NX-OS) SPAN feature is pretty similar to the SPAN feature on Cisco IOS-XE software. Different Nexus series switches might vary on the number of monitor sessions they support. Before configuring the monitor session on NX-OS, the destination switchport should be configured with the command `switchport monitor`. Nexus supports hierarchical configuration, thus the source ports and destination port configuration are defined under the monitor configuration mode. Example 1-4 illustrates how to configure SPAN session on Nexus switches. Note that a monitor session on NX-OS does not become active unless a `no shut` command is configured under the monitor session.

Example 1-4. Configuring SPAN Session on Cisco NX-OS

```
NX-2(config)# interface Etherner1/5-6
NX-2(config-if)# switchport
NX-2(config-if)# switchport monitor
NX-2(config-if)# no shut
NX-2(config)# monitor session 1
NX-2(config-monitor)# source interface ethernet 1/1
NX-2(config-monitor)# source interface ethernet 1/2 tx
NX-2(config-monitor)# destination interface ethernet 1/5
NX-2(config-monitor)# no shut
```

Once the monitor session is configured, the session state can be verified using the command `show monitor session session-number`. Example 1-5 displays the output of the command `show monitor session 1`. Notice that in the output, the type is set to local. When defining the monitor session, if the type is not specified, then by default the monitor session is configured as a local SPAN session. The output shown in Example 1-5 displays the source rx and tx interfaces that are configured in the SPAN along with the destination interface. In NX-OS, the **show interface** command does not show the state of the interface as monitoring, but rather has another line in the output indicating that the switchport monitor is enabled on the port.

Example 1-5. Verifying SPAN Session on Cisco NX-OS

```
NX-2# show monitor session 1
   session 1
---------------
type              : local
state             : up
acl-name          : acl-name not specified
```

```
source intf       :
    rx            : Eth1/1
    tx            : Eth1/1        Eth1/2
    both          : Eth1/1
source VLANs      :
    rx            :
    tx            :
    both          :
filter VLANs      : filter not specified
source fwd drops  :
destination ports : Eth1/5
PFC On Interfaces :
source VSANs      :
    rx            :
NX-2# show interface ethernet 1/5
Ethernet1/5 is up
admin state is up, Dedicated Interface
  Hardware: 100/1000/10000 Ethernet, address: 0c5f.3d16.260d
  (bia 0c5f.3d16.260d
)
  MTU 1500 bytes, BW 1000000 Kbit, DLY 10 usec
  reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, medium is broadcast
  Port mode is access
  full-duplex, 1000 Mb/s
  Beacon is turned off
  Auto-Negotiation is turned on  FEC mode is Auto
  Input flow-control is off, output flow-control is off
  Auto-mdix is turned off
  Switchport monitor is on
  EtherType is 0x8100
```

```
  EEE (efficient-ethernet) : n/a
    admin fec state is auto, oper fec state is off
  Last link flapped 00:13:37
<snip>
```

Nexus 9000 series switches also support filtering of spanned traffic at the VLAN level or by applying an ACL. Only one type of filtering is supported on a given monitor session and filtering can only be applied when the source interfaces are configured in the rx direction. Example 1-6 illustrates how to set up a monitor session with VLAN as well as ACL-based filtering and also how the show monitor session output differs between the monitor sessions when different filtering methods are applied. Also, as shown in this example, a monitor session can be deleted using the command `no monitor session session-number`.

Example 1-6. SPAN Sessions with Filtering

```
NX-2(config)# ip access-list TEST-ACL
NX-2(config-acl)# permit icmp any any
NX-2(config-acl)# exit
NX-2(config)# no monitor session 1
NX-2(config)# monitor session 1
NX-2(config-monitor)# source interface ethernet 1/1 rx
NX-2(config-monitor)# destination interface eth1/5
NX-2(config-monitor)# filter access-group TEST-ACL
NX-2(config-monitor)# exit
NX-2(config-if)# monitor session 2
NX-2(config-monitor)# source interface eth1/3 rx
NX-2(config-monitor)# destination interface eth1/5
NX-2(config-monitor)# filter ?
  access-group  Access control group
  vlan          Vlan type
```

```
NX-2(config-monitor)# filter vlan 100
NX-2(config-monitor)# no shut
NX-2# show monitor session 1
   session 1
---------------
type              : local
state             : up
acl-name          : TEST-ACL (Rx only)
source intf       :
    rx            : Eth1/1
    tx            :
    both          :
source VLANs      :
    rx            :
    tx            :
    both          :
filter VLANs      : filter not specified
source fwd drops  :
destination ports : Eth1/5
PFC On Interfaces :
source VSANs      :
    rx            :
NX-2# show monitor session 2
   session 2
---------------
type              : local
state             : up
acl-name          : acl-name not specified
source intf       :
    rx            : Eth1/3
    tx            :
    both          :
```

```
source VLANs      :
    rx            :
    tx            :
    both          :
filter VLANs      : 100
source fwd drops  :
destination ports : Eth1/6
PFC On Interfaces :
source VSANs      :
    rx            :
```

Enabling Port Mirroring on Arista EOS

Port mirroring on Arista can be enabled by configuring one or more mirroring sessions. Port mirroring configuration of Arista EOS is very similar to that for Cisco IOS-XE devices. You can enable port mirroring sessions using the command `monitor session` with the difference that instead of specifying the session number, you specify the name of the monitor session. Example 1-7 demonstrates how to configure port mirroring on Arista devices. Once configured, use the command `show monitor session` to verify the state of the monitor session and the `show interface interface-name status` command to verify the state of the interface. The destination port on the Arista device also displays in monitoring state.

Example 1-7. Configuring Port Mirroring on Arista

```
eos-1(config)# monitor session test1 source ethernet 1,7-9 rx
eos-1(config)# monitor session test1 source ethernet 4 tx
eos-1(config)# monitor session test1 destination ethernet 20
eos-1# show monitor session
Session test1
------------------------
```

```
Source Ports
Rx Only:     Et1, Et7, Et8, Et9
Tx Only:     Et4
Destination Port: Et20
eos-1# show int et20 status
Port   Name   Status    Vlan         Duplex   Speed   Type
Et20          connect   monitoring   full     10G     Not Present
```

Enabling Port Mirroring on JunOS

On JunOS, we use the term analyzers to set up port mirroring. JunOS also supports configuring of port mirroring to capture bridged packets (Layer 2 packets) as well as routed packets (Layer 3 packets). On a JunOS device, the following packets can be mirrored:

- Packets entering or exiting a port
- Packets entering or exiting a VLAN or a bridge domain
- Policy-based sample packets

For policy-based sample packets, a firewall filter with a policy is configured to mirror the packets. The sample traffic based on the firewall filter can be sent to the port-mirroring instance for further analysis.

Analyzers on JunOS can be set up in few simple steps:

- Get into forwarding-options configuration mode.
- Define a name for the analyzer and specify the input interface along with the direction of the traffic you wish to capture.
- Choose the destination interface.
- Commit the configuration.

In JunOS, you can also configure firewall filters to limit the amount of traffic being mirrored. Example 1-8 displays a sample configuration of analyzer and the use of the command `show forwarding-options analyzer analyzer-name` to verify the state of the analyzer.

Example 1-8. Configuring Port Mirroring on JunOS

```
root> show configuration forwarding-options
analyzer {
    testCapture {
        input {
            ingress {
                interface ge-0/0/1.0;
            }
            egress {
                interface ge-0/0/1.0;
            }
        }
        output {
            interface ge-0/0/4.0;
        }
    }
}
root> show forwarding-options analyzer testCapture
  Analyzer name                    : testCapture
  Mirror rate                      : 1
  Maximum packet length            : 0
  State                            : up
  Ingress monitored interfaces     : ge-0/0/1.0
  Egress monitored interfaces      : ge-0/0/1.0
```

So far, we have seen how to configure local port mirroring on various vendor devices running their respective network OS. Similarly, you can also set up remote port mirroring. In remote port mirroring, the configuration is pretty much the same as local port mirroring with the minute difference that the destination interface does not reside on the local device, but is multiple hops away. Each vendor has its own method of implementing remote port mirroring. Unless necessary, it is not required to set up remote port mirroring.

Summary

In this chapter, we learned what NTA is and why it is important in the network. We also learned the factors that should be considered when implementing port mirroring and how we can set up the minimum number of capture points in the network to isolate a problem in the network. Unless it is necessary, one should avoid enabling port mirroring on network devices. Further, we learned what Wireshark is and how to install it on various OSs. Finally, we concluded the chapter by seeing how port mirroring can be enabled on network devices from different vendors.

References in This Chapter

- Wireshark: https://www.wireshark.org
- Network Management Configuration Guide: https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst9300/software/release/16-10/configuration_guide/nmgmt/b_1610_nmgmt_9300_cg/configuring_span_and_rspan.html

- Cisco Nexus 9000 Series NX-OS System Management Configuration Guide: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/7-x/system_management/configuration/guide/b_Cisco_Nexus_9000_Series_NX-OS_System_Management_Configuration_Guide_7x/b_Cisco_Nexus_9000_Series_NX-OS_System_Management_Configuration_Guide_7x_chapter_010000.html
- Arista EOS Introduction to Port Mirroring: https://eos.arista.com/introduction-to-port-mirroring/
- Juniper Port Mirroring and Analyzers: https://www.juniper.net/documentation/us/en/software/junos/network-mgmt/topics/topic-map/port-mirroring-and-analyzers.html

CHAPTER 2

Getting Familiar with Wireshark

Network administrators and security analysts often work packet captures to analyze the traffic and determine the cause of network events and attacks in the network. With Wireshark being the preferred tool to capture and analyze network traffic, it is important to have an understanding of how to use Wireshark's features and know about its options. This chapter focuses on various features and options available in Wireshark.

This chapter covers the following topics:

- Overview of Wireshark tool
- Performing packet capture using Wireshark
- Working with Wireshark capture files
- Analyzing packets in Wireshark

Overview of Wireshark Tool

In the previous chapter we learned about what Wireshark is and how to install Wireshark on various OSs. In this chapter, we focus on how to use the Wireshark tool. After Wireshark is installed, you can open

V. Jain, *Wireshark Fundamentals*, https://doi.org/10.1007/978-1-4842-8002-7_2

the Wireshark tool using the Wireshark shortcut from the installed applications list. Before diving into how to use Wireshark, let's take a closer look at the user interface (UI), which is shown in Figure 2-1.

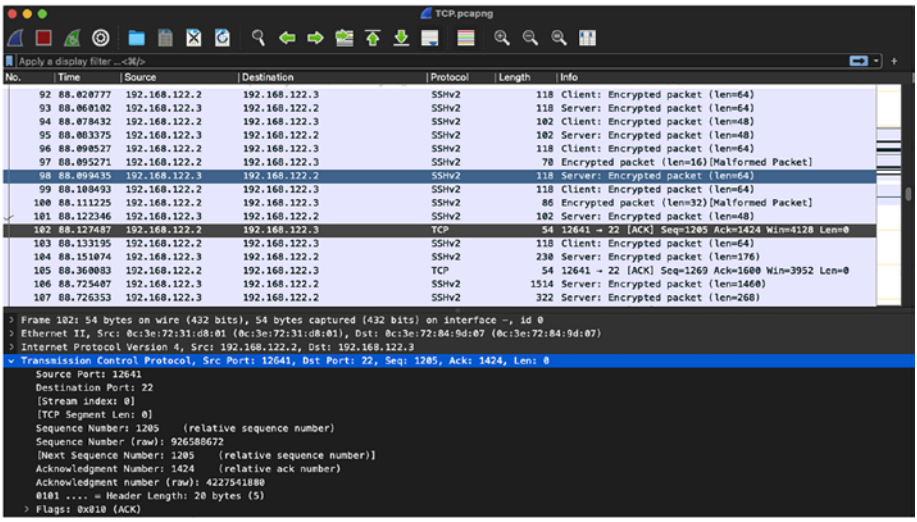


Figure 2-1. *Wireshark user interface*

Wireshark Preferences

There are numerous settings that a user can customize when using Wireshark. All these settings can be changed from the Preferences window. To open the Preferences window, navigate through the following menu:

- Mac: Wireshark | Preferences
- Windows: Edit | Preferences
- Linux: Edit | Preferences

From the Preferences window, users can change the settings for the following sections.

Appearance

The Appearance section of the Wireshark preferences allows you to change the UI settings of Wireshark. In this section, you can adjust the text for the window title, specify the columns that you want to see when using Wireshark, and set the font and colors and the layout of Wireshark UI. For instance, if you want to change the default layout of Wireshark to a layout that is more comfortable for you, you can change what information the different panes in the layout will display. Figure 2-2 displays the modified layout where Pane 1 displays the packet list, Pane 2 displays the packet details, and Pane 3 displays the packet bytes in the layout that you selected in the Preferences window.

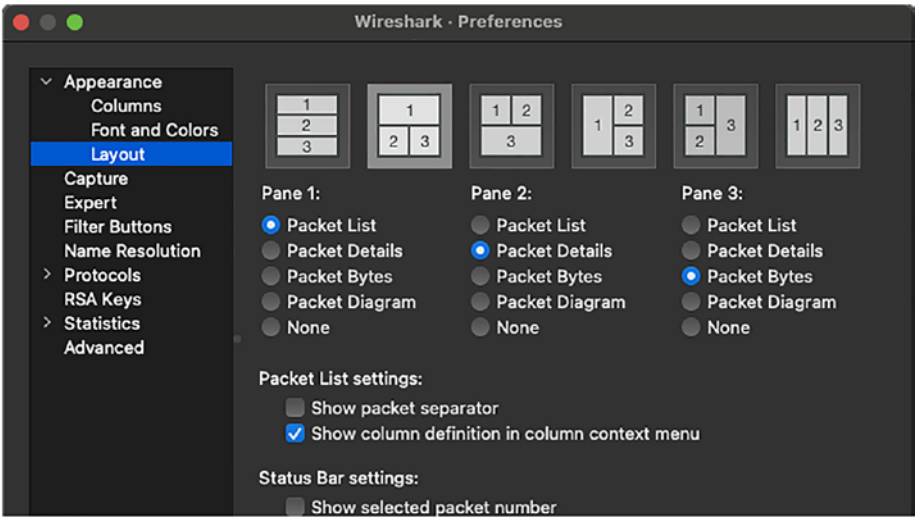


Figure 2-2. *Custom Wireshark layout*

Capture

The Capture section allows you to select the default interface that Wireshark will use for capturing traffic. Users can also select other settings in this section:

- Enable or disable the option to capture packets in promiscuous mode
- Enable or disable the option to capture packets in pcapng format
- Enable or disable the option to update the list of packets in real time
- Enable or disable the option for automatic scrolling when capturing live packets
- Enable or disable the option to not load interfaces on startup
- Enable or disable the option to disable external capture interfaces

Expert

The Expert section of the Wireshark Preferences window allows you to define different field names and set the severity for those fields.

Note The Expert section is covered later when covering Expert Information.

Filter Buttons

The Filter Buttons options allow the user to create custom shortcuts on the toolbar for various filter expressions. By using these buttons, users don't have to repeatedly type the filter expressions; instead they can just click the button to apply the filter on the captured traffic. Figure 2-3 displays how to create a filter button for the HTTP GET method.

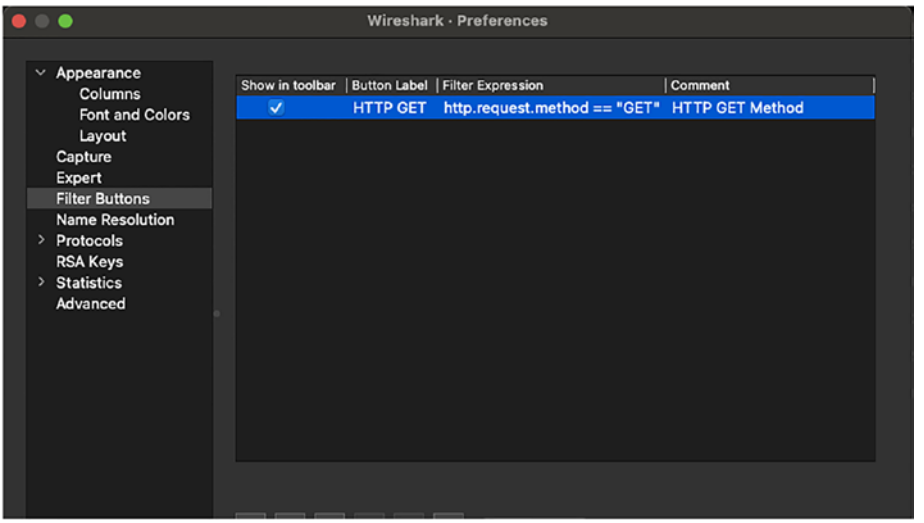


Figure 2-3. *HTTP GET filter button*

Name Resolution

The Name Resolution settings allow users to update the settings with regard to MAC address resolution and transport and network address resolution. These settings also allow users to use Domain Name System (DNS) packets for address resolution, use an external network name resolver, and also use the list of DNS servers for name resolution. The Name Resolution section also has options to list the DNS servers that can be used by Wireshark.

Protocols

The Protocols section of the Preferences window allows the user to configure settings for various lists of protocols supported by Wireshark. This is useful for analyzing traffic in network environments where the protocols are being used ports different than the default port numbers.

RSA Keys

The RSA Keys section allows user to configure the RSA private keys for decryption. In this section, use the Add New Keyfile button to select a file. The user will be prompted for a password if necessary. The Add New Token button can be used to add keys from a hardware security module (HSM), which might require using Add New Provider to add a vendor-specific configuration.

Statistics

This section allows you to customize the settings used by Wireshark to perform and display statistical analysis of the captured traffic. Settings such as burst rate resolution, burst rate window size, tab update interval, and so on, can be configured under this section.

Advanced

The Advanced section of the Preferences window allows user to view and edit all Wireshark preferences. If you are familiar with `about:config` in Firefox or `chrom:flags` in the Chrome web browser, then making changes using the Advanced pane will be a walk in the park. Users can search for a preference by typing text in the Search box on this window as shown in Figure 2-4.

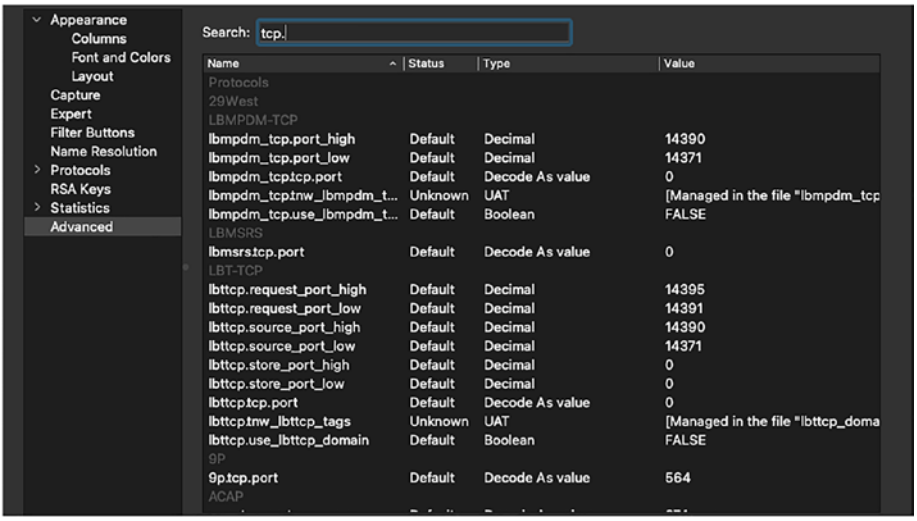


Figure 2-4. *Wireshark Advanced preferences*

Performing Packet Capture Using Wireshark

When the Wireshark application is launched, a welcome screen displays options to either open old files or to start a new packet capture on the current device. In the Capture section, all the wired, wireless, and virtual network interfaces that can be selected to begin the packet capture immediately are listed. Alternatively, users can go to the Capture menu and then select the Option submenu. This will open the Wireshark – Capture Options window shown in Figure 2-5, which has three tabs:

- *Input:* This tab displays all the interfaces. You can enable the listed network interfaces and select one of the interfaces on which you wish to capture incoming and outgoing packets.
- *Output:* This tab allows the user to edit the output settings, such as the output format, permanent file

name where the packet capture will be saved, and also options to save the captured traffic into a new file by limiting the number of packets, file size, duration, and so on.

- *Options:* The Options tab gives you options to set the display settings of the captured packets, such as updating the list of packets in real time, automatic scrolling during live capture, and showing capture information during live capture. It also has options for name resolution such as resolving MAC addresses, network names, and transport names. Users can also define the settings for when they can stop the packet capture.

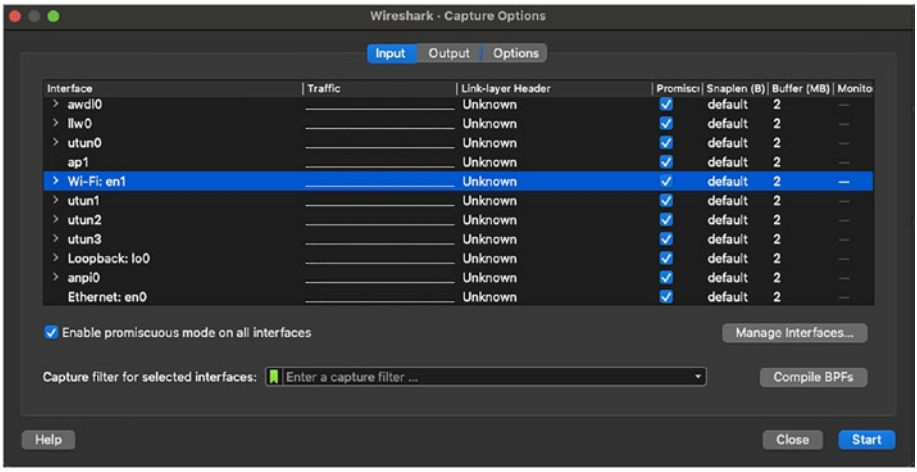


Figure 2-5. *Wireshark - Capture Options window*

Once all the capture options are set, users can click the Start button to begin the packet capture. It is important to note that you can capture traffic on interfaces that are in promiscuous mode. This mode allows you to see all the traffic coming into the NIC. In Figure 2-5, notice that all the listed interfaces have promiscuous mode enabled.

Note To perform on Mac OS, users are required to install the ChmodBPF application. By default, users on Mac OS do not have privileges or permission to capture traffic on local interfaces. Once the ChmodBPF daemon is launched, it creates the `access_bpf` group and adds the user to that group. Similarly, on Windows, Wireshark requires either Npcap or WinPcap to capture live network traffic.

Dissectors

As most of you might already know, traffic enters the NIC in binary format. Wireshark uses the Enhanced Packet Analyzer (EPAN), which decodes the binary data into human-readable format. EPAN is the main core of the Wireshark tool. It is the packet analyzer engine that uses dissectors to re-create the protocol packets from the binary data. EPAN primarily consists of four components:

- *Protocol tree:* Performs detailed analysis of a single packet.
- *Dissectors:* Hold the information from the Request for Comment (RFC) and other specifications on how to decode and interpret fields of different protocol packets.
- *Dissector plug-ins:* Allows the use of default dissectors that come with Wireshark and also allows the use of user-created dissector plug-ins.
- *Display filters:* – Provide options to perform filtering on captured data.

Dissection of any packet can be broken down into a few simple steps:

1. Wireshark identifies the frame type of any incoming packet and hands it off to the correct frame dissector, for instance, Ethernet.
2. The dissector breaks down the contents of the frame header to understand which section to look up next. For instance, Ethernet type 0x0800 in the Type field of the Ethernet header indicates Internet Protocol version 4 (IPv4). Wireshark then hands off the packet to the IP dissector.
3. After the IP dissector decodes the IP header, it identifies the next protocol header by looking at the Protocol field in the IP header. If the value is 0x06 it hands off the packet to the TCP dissector. If the value is 0x11, it hands off the packet to the User Datagram Protocol (UDP) dissector.
4. This process is followed until there are no further dissections identified by the current dissector.

Although Wireshark is a very mature application and supports a wide range of protocol specifications and dissectors, there might still be scenarios where you are required to guide Wireshark on how to decode a protocol. For such scenarios, users can simply right-click the frame and select the Decode-As option. This option will open the window shown in Figure 2-6. Using this window, a user can select the field type from which the user can select any of the options such as a TCP port, a UDP port, and so on. Once that is selected, the user can then define the value and then map the field and value to a particular protocol from the drop-down list in the Current column. For example, let's presume that Wireshark does not understand a Virtual Extensible Local Area Network (VXLAN) packet. When Wireshark receives such a packet, the user can select the packet and

choose UDP port as the Field, set the Value to port 4789, and in the Current column, map the packet to the VXLAN protocol. This setup is shown in Figure 2-6.

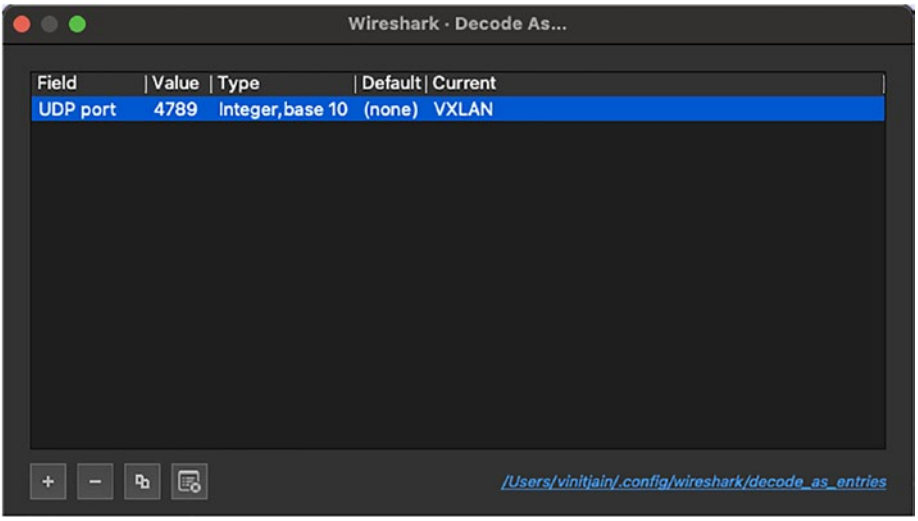


Figure 2-6. *Wireshark - Decode As*

This feature comes in very handy when the network administrators are running the protocols on a port numbers other than their defaults.

Configuration Profiles

Wireshark allows users to define and maintain configurations and preferences in the form of configuration profiles. Wireshark comes with four predefined configuration profiles:

- *Default:* Default profile
- *Bluetooth:* Global profile
- *Classic:* Global profile
- *No Reassembly:* Global profile

The configuration profiles store the following set of information:

- Preferences
- Capture filters (*cfilters*)
- Display filters (*dfilters*)
- Coloring rules
- Disabled drotocols
- User accessible tables (e.g., custom HTTP headers, custom LDAP `AttributeValue` types, etc.)
- Dissector assignments (`decode_as_entries`)
- Recent settings such as pane sizes, column widths, and so on

Users can create custom profiles in few simple steps.

1. On the Edit menu, click Configuration Profiles. This opens the Configuration Profiles dialog box.
2. In the Configuration Profiles dialog box, click the + icon to add a new profile. For example, let's create a configuration profile named Network Profile. This profile will be of type Personal. The newly created profile is created with the default settings that are part of the Default profile.
3. Select the Network Profile and click OK.

Once the custom profile is created and selected, all the preferences and other settings such as capture or display filters will be saved under the custom configuration profile.

Filtering with Wireshark

When packet capture is performed using Wireshark, all the incoming and outgoing traffic on the selected NIC is captured. This limits the user to capturing a huge amount of packets on high-speed as well as high-traffic links. Although capturing more data is never bad, it could also lead to other issues:

- Crashing of Wireshark application due to large file size
- Longer time needed to load and analyze the captured packets
- Might not be able to capture problematic traffic during a short time span due to higher packet per second (pps) rate

Filtering in Wireshark can be of two types:

- *Capture filter:* This is used to filter or restrict the packets that will be captured by Wireshark.
- *Display filter:* This is used to filter the packets from the captured traffic.

We next discuss both these filtering capabilities in detail.

Capture Filters

As stated previously, the capture filter in Wireshark is used to limit the packets that can be captured during a live capture. This means that the capture filter cannot be applied on existing packet capture or `pcap` files. In scenarios where the network is busy with heavy traffic or during network troubleshooting when a user wants to capture a specific packet, capture filters are a very useful feature. Capture filters are applied on the packets after they are processed by WinPcap in a Windows installation or the libpcap library in a Linux installation. Once the packets are passed through

the filter criteria, they are then passed to the Wireshark capture engine as shown in Figure 2-7. Note that once the packets are parsed through the capture filter, only the filtered packets are received by the capture engine. The remaining packets are dropped and discarded before being sent to the capture engine.

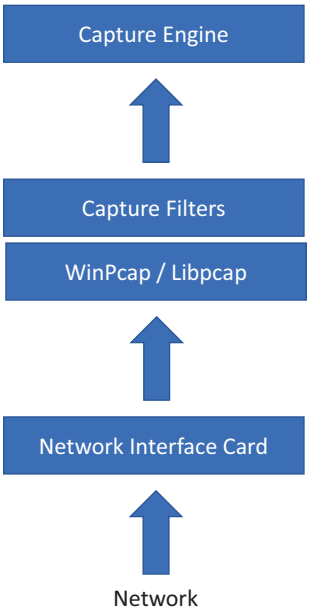


Figure 2-7. *Wireshark capture filter*

Capture filters follow the Berkeley Packet Filtering (BPF) syntax, which is also used by Tcpdump. Wireshark comes with default capture filters named *cfilters* that are stored in the Wireshark application or program file directory. Wireshark comes with the default capture filters shown in Table 2-1.

Table 2-1. *Default Wireshark Capture Filters*

Filter Name	Filter Config
Ethernet address 00:08:15:00:08:15	`ether host 00:08:15:00:08:15`
Ethernet type 0x0806 (ARP)	`ether proto 0x0806`
No Broadcast and no Multicast	`not broadcast and not multicast`
No ARP	`not arp`
IPv4 only	`ip`
IPv4 address 192.0.2.1	`host 192.0.2.1`
IPv6 only	`ip6`
IPv6 address 2001:db8::1	`host 2001:db8::1`
TCP only	`tcp`
UDP only	`udp`
Non-DNS	`not port 53`
TCP or UDP port 80 (HTTP)	`port 80`
HTTP TCP port 80	`tcp port http`
No ARP and no DNS	`not arp and port not 53`
Non-HTTP and non-SMTP to/from www.wireshark.org	`not port 80 and not port 25 and host www.wireshark.org`

Users can also create custom cfilters that can be part of the default profile or a custom profile. Let's now create a custom capture filter for capturing only VXLAN traffic. To filter VXLAN encapsulated traffic, we can simply filter on UDP port 4789. This filter can be created in a few simple steps:

1. Go to the Capture menu and select Capture Filters to open the Capture Filters dialog box.

2. In the Capture Filters dialog box, click the + icon, which will add an entry at the end of the existing default list.
3. Edit the name of the filter and set it to VXLAN only and then edit the Filter Expression and set it to udp port 4789.
4. Click OK to save.
5. Once saved, go to the Capture menu and select Options. This will open the Wireshark Capture Options dialog box.
6. In this dialog box, click the green bookmark icon next to Capture Filter for Selected Interfaces. This displays the list of all capture filters that are available. Select the VXLAN Only option as shown in Figure 2-8.
7. Once the interface and capture filter are selected, click Start to start the capture.

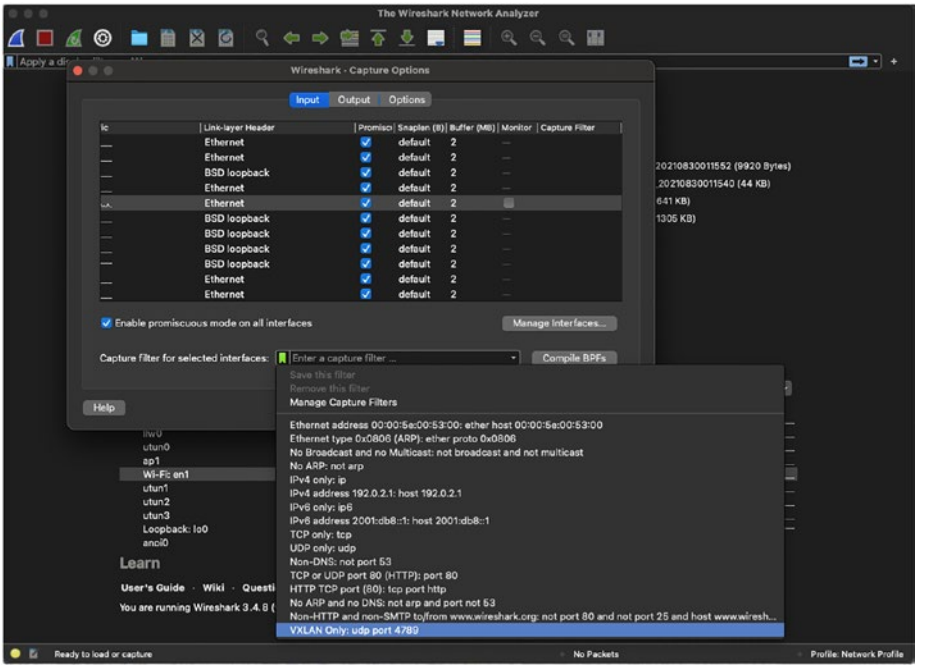


Figure 2-8. *Selecting a customdefined capture filter*

You will now notice that only the VXLAN packets are being captured in Wireshark. This method of capturing packets has the benefit of capturing only specific traffic, but if the user is unsure about which traffic to capture, it might be a better option to use display filters.

Display Filters

Most traffic analysis is performed during live traffic or on precaptured packet captures. To analyze traffic in both these scenarios, display filters can help users easily narrow down the problematic traffic quickly by

applying the filter criteria on the packets. Display filters enable users to focus on specific packets based on the filter expressions that are specified. There are several ways of creating display filters:

- Typing the display filter criteria with the help of auto-complete
- Applying saved display filters
- Using expressions
- Right-clicking the filter
- Applying conversation or endpoint filters

Before moving on to checking different ways of implementing display filters, let's talk about the syntax for display filters. Wireshark uses a proprietary Wireshark display filter that is different than the capture filter's BPF format. Even though the syntax for both capture and display filters is different, there are a few examples where the syntax for both of them happens to be the same. For instance, the syntax for filtering TCP traffic on both filters is specifying the `tcp` keyword. Figure 2-9 illustrates how the packets are filtered after applying the display filter for TCP traffic. Notice that in this example, there are 76 packets that have been identified and filtered for TCP traffic out of 1,452 packets.

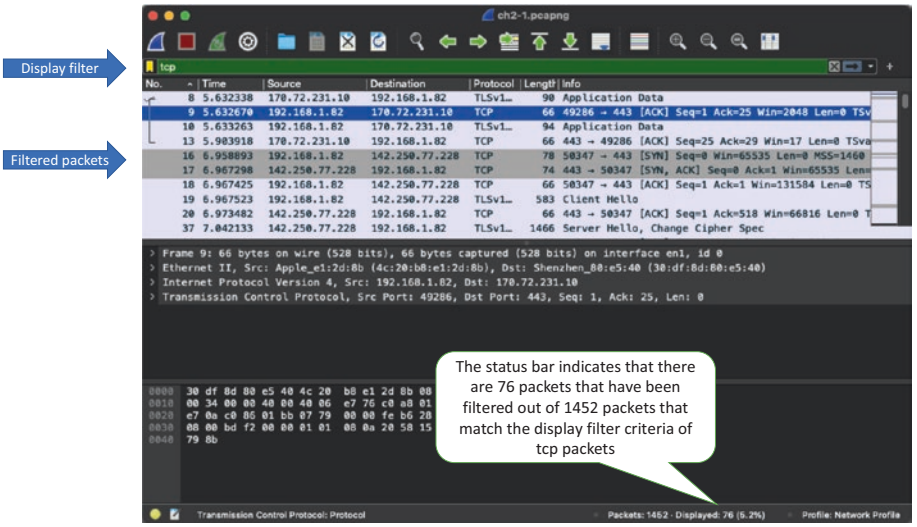


Figure 2-9. *Filtering packets using a display filter*

Display filters can be relatively simple or quite complex. It all depends on the display filter expression. Users can perform simple filtering by specifying the protocol traffic that they want to filter. For instance, Table 2-2 displays a sample list of packets that can be filtered with just a single filtering keyword.

Table 2-2. *Simple Display Filters*

Filter Config	Filter Description
`tcp`	Filtering only TCP packets
`udp`	Filtering only UDP packets
`ip`	Filtering only IPv4 traffic
`ipv6`	Filtering only IPv6 traffic
`arp`	Filtering only ARP broadcast packets
`dns`	Filtering only DNS packets

Display filters also allow users to filter packets based on packet characteristics. For instance, if a user wants to filter packets that have an invalid IP header checksum, they can simply set the display filter to `ip.checksum_bad.expert`. Note that by packet characteristics we do not mean an actual field in the headers. Some examples of display filters based on packet characteristics are listed in Table 2-3.

Table 2-3. *Display Filters for Packet Characteristics*

Filter Config	Filter Description
`tcp.analysis.flags`	Displays packets that contain one of the TCP analysis flags packets
`tcp.bogus_header_length`	Filters TCP packets that have bogus header length in the TCP header
`ip.bogus_header_length`	Filters packets that have bogus header length in the IP header

The display filters provide an option to filter more specific packets by the use of expressions. Expressions allow users to define filters based on the contents of a field and matching specific values that can be set using comparison operators. Display filters can also be a combination of two or more expressions that are evaluated based on the evaluation operators. The operators that can be used with display filters are listed in Table 2-4.

Table 2-4. *Operators for Display Filters*

Operators	Operator Description
==	Exactly matches the specified value
>	Matches when the value of the field is greater than the specified value
<	Matches when the value of the field is less than the specified value
>=	Matches when the value of the field is greater than or equal to the specified value
<=	Matches when the value of the field is less than or equal to the specified value
!	Filters all the values of the field that do not match the specified expression
!=	Filters all the values that do not match the specified value
&&	Allows AND operation between two different expressions; filters the packets that match all the specified expressions
||	Allows OR operation between two different expressions; filters the packets that match any of the two expressions

Let's now examine how we can use these operators to create expressions for display filters. Table 2-5 displays a list of expressions for filtering various types of traffic.

Table 2-5. *Expressions for Display Filters*

Filter Expressions	Filter Description
`http.request.method == "POST"`	Filters traffic that includes the HTTP POST method in the HTTP headers
`tcp.window_size < 1500`	Matches packets that have TCP window size less than 1,500
`dns.qry.name == "www.google.com"`	Filters DNS queries for `www.google.com`
`udp.port != 686`	Filters out packets that do not match UDP port number 686
`(arp.opcode == 0x0001) && (arp.src.hw_mac == 00:01:ab:cd:0e:02)`	Displays ARP request only from MAC address 00:01:ab:cd:0e:02
`(tcp.flags.syn == 1) && !(tcp.flags.ack == 1)`	Displays packets that have the TCP SYN bit set but do not have the TCP ACK bit set.
`(icmp.type == 3) && ((icmp.code = 0x01) || (ip.addr == 192.168.100.1))`	Displays Internet Control Message Protocol (ICMP) unreachable packets where the host is unreachable or either the source or destination address is 192.168.100.1

Because there is a different set of fields within each header, it is nearly impossible to remember all the fields to create the display filters. Wireshark comes with an auto-complete feature that helps users to create filters. Users are only required to know the top-level header and the Wireshark Intellisense or auto-complete feature kicks in as soon as any character is typed. The Wireshark auto-complete feature displays all the available options within that header that can be used to create the filter. For example, if the user wants to check for any traffic with destination port 53 or DNS traffic, the user can just type in tcp and it will display all the

available options. In this case, the option would be tcp.dstport, as shown in Figure 2-10. Once the user identifies the right filter option, he or she can then complete the expression by using the comparison operators. For this example, the display filter is `tcp.dstport == 53`.

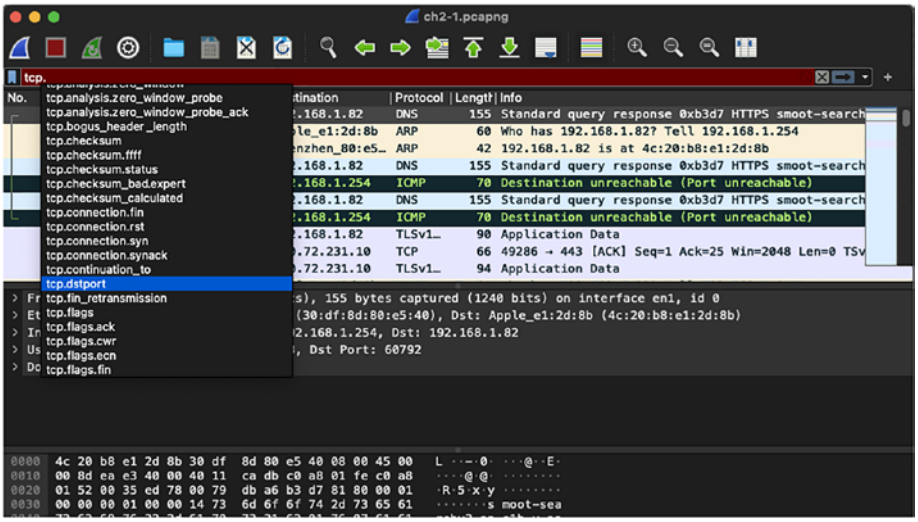


Figure 2-10. *Display filter using auto-complete*

Users are also allowed to select display filters from previously used filters or save their display filter like a capture filter. To use the previously searched display filter, use the drop-down list seen at the end of the display filter bar. To save the current display filter, use the bookmark icon at the beginning of the display filter area, as shown in Figure 2-11. The drop-down list shows the default display filters available as well as other options to save or manage the display filters. Selecting the Save This Filter option opens the Display Filters dialog box. There you can click the + icon to add the current display filter.

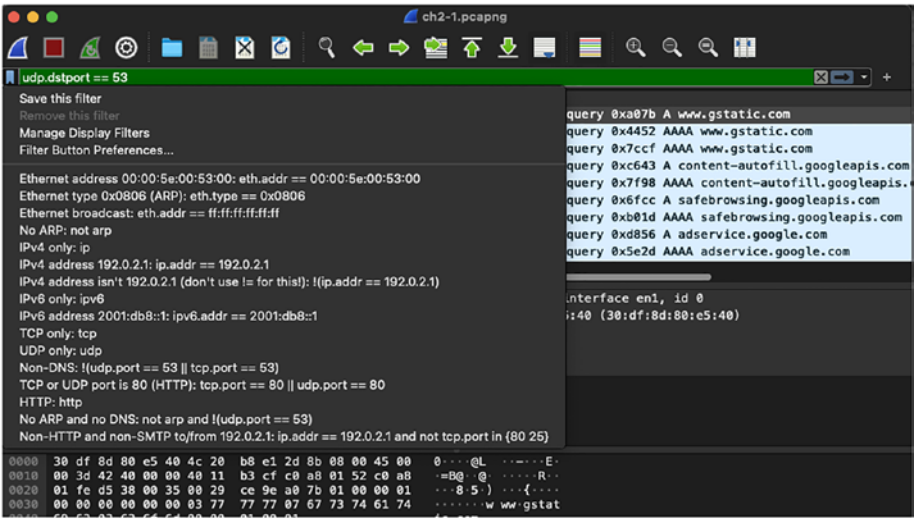


Figure 2-11. *Display filter bookmarks and options*

Another quick method of filtering the packets is using the right-click filtering method. While navigating through the list of packets, if you come across a packet that looks suspicious or you are interested in checking out similar packets, you can simply right-click the packet or field of interest and select either the Apply as Filter | Selected or the Prepare as Filter | Selected option as shown in Figure 2-12. The Apply as Filter | Selected option directly places the filter on the live traffic or captured packets, whereas the Prepare as Filter | Selected option prepares the display filter and gives the user an option to edit the filter before it is applied on the live traffic or captured packets.

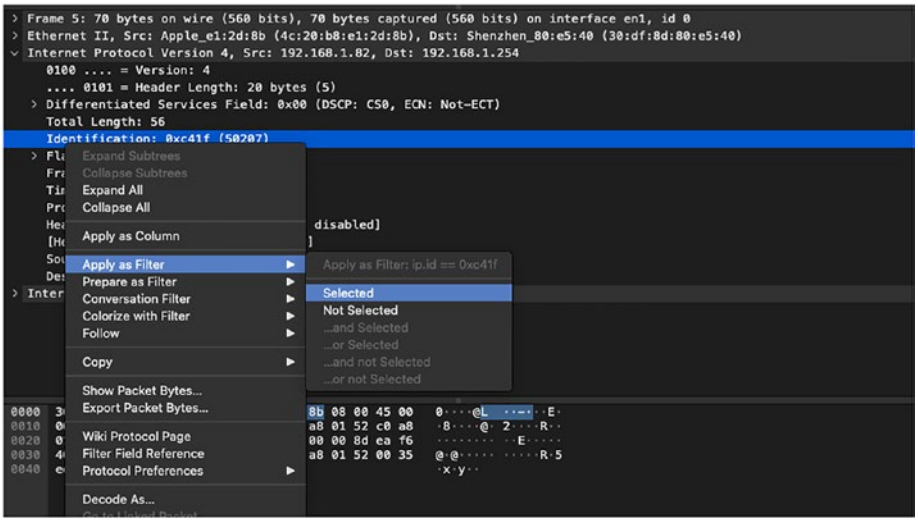


Figure 2-12. *Right-click filtering*

Within both Apply as Filter | Selected and Prepare as Filter | Selected, users can choose from one of the available filter options:

- *Selected:* Creates a filter matching the selection
- *Not Selected:* Creates an exclusion filter
- *And Selected:* Must match both the existing filter and the selection
- *Or Selected:* Must match either the existing filter or the selection
- *And Not Selected:* Must match the existing filter with the exclusion of the selection
- *Or Not Selected:* Must match either the existing filter or filter based on the exclusion of the selection

Users can also leverage the Copy | As Filter feature available in Wireshark as part of right-click filtering. This feature allows users to copy the filter expression without applying or listing the filter in the display filter pane. This feature can be very useful for creating complex display filters or for copying filters between different Wireshark instances where we want to trace the packets across multiple capture files.

Although there are many ways of creating display filters, one of the features that really stands out in Wireshark is its ability to catch errors or mistakes in display filters, which prevents users from applying the wrong display filters on the packet captures. The display filter pane turns red and disables the option to apply a filter if there is an incomplete or incorrect display filter expression typed in the pane.

Working with Wireshark Capture Files

As stated before, Wireshark captures network traffic and allows the user to save the packets with either `.pcap` or `.pcapng` extensions. The `pcap` file format is the initial version of the file format that was originally implemented in UNIX and Linux using the libcap library. This file format was implemented in Windows using the WinPcap library. The `pcapng` file format was the result of an Internet Engineering Task Force (IETF) draft that specifies the *PCAP Next Generation (pcapng) Capture File Format*. Through this IETF draft, the proponents defined standardized blocks and fields, thus making the `pcapng` format a more extensible and futureproof file format.

PCAP vs. PCAPng

There are several differences between the `pcap` and `pcapng` file formats, some of which are listed in the sections that follow.

Capture from Multiple Interfaces

The pcap format contains some information about the capture interface but does not have support for multiple interfaces. This is because the interface information is included as part of the common header and not stored on a per-packet basis, making it difficult to capture traffic from multiple interfaces in the same capture file. On the other hand, the pcapng file format supports multiple interfaces by using the interface description block defined in the PCAP Next Generation (pcapng) capture file format IETF draft. Using the interface description block, each packet can be associated with a specific interface. Figure 2-13 displays the interface description block as defined in the IETF draft. Note that the block type of the interface description block is 1 (0x00000001).

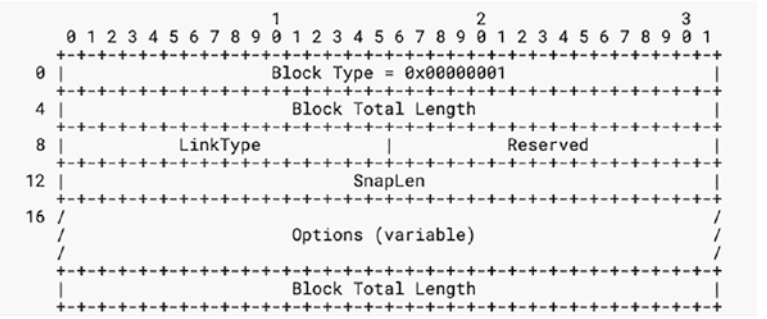


Figure 2-13. *Interface description block*

Note A simple packet block, which is a smaller and simpler packet block that is easy to process and contains a minimal set of information, does not contain the Interface ID field and is thus set to a default value of 0. With a simple packet block, it is assumed that the packets have been captured on the interface that was previously specified in the first interface description block.

When a packet capture is initiated for multiple interfaces, a user can see the packet of each interface in the pcapng file using their interface_id field. Figure 2-14 displays the packets belonging to interface id 1, which in this case is a loopback interface (IP address 127.0.0.1) of the PC itself.

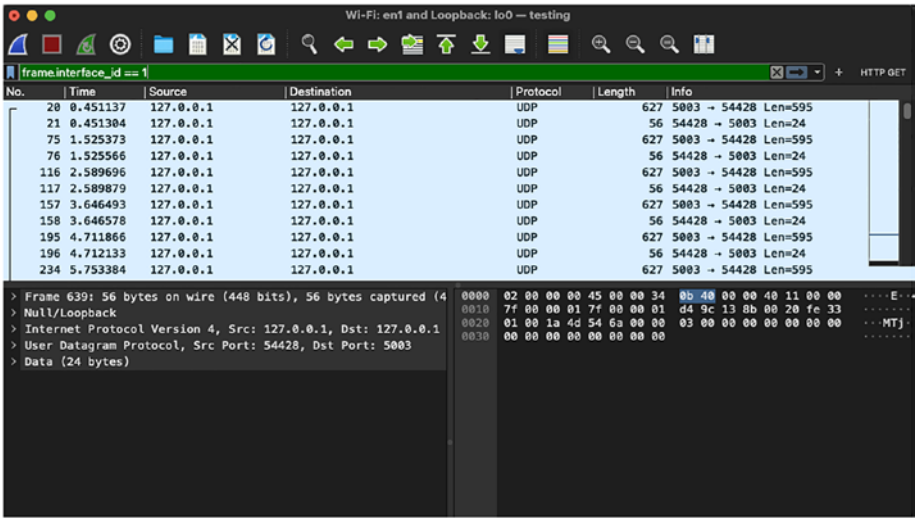


Figure 2-14. *Packets with interface id 1*

Timestamps

With pcap format, one of the major concerns for network analysts was its resolution on packet timestamps. Each packet in the pcap format has a time resolution accurate to the microsecond level (i.e., 10^{-6} seconds), which provides a resolution for 999,999 packets per second. On first look, this number looks reasonable, but with the modern-day networks evolving to 25 Gig, 40 Gig, and 100 Gig links, microsecond-level accuracy can create a huge gap. It is imperative to note that even a common 1 Gig link can easily exceed this link. The pcapng file format provides the capability to adjust the resolution using a flexible timestamp format, which is now expressed as a 64-bit time unit that can easily accommodate evolving

network speeds. The default resolution value on packet timestamps is still given in microseconds, but this can be altered by setting the `if_tsresol` option in the interface description block.

Embedding Comments

Troubleshooting networks can be complex and time consuming and could be further delayed when information is shared across peers or customers. The pcapng format allows the user to embed both top-level and per-packet comments that can be helpful when traces are shared across multiple users for analysis. To add a comment to a packet, select a packet and right-click to choose the Packet Comment option. Once selected, this opens a window that will allow the user to add a comment on the packet, as shown in Figure 2-15.

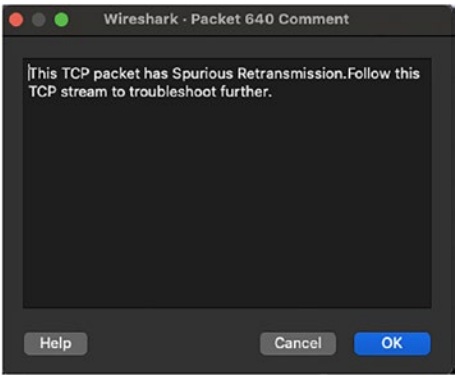


Figure 2-15. *Adding comments on a packet*

Once the comment is added, the packet headers will also have a packet comments section added at the top, as shown in Figure 2-16.

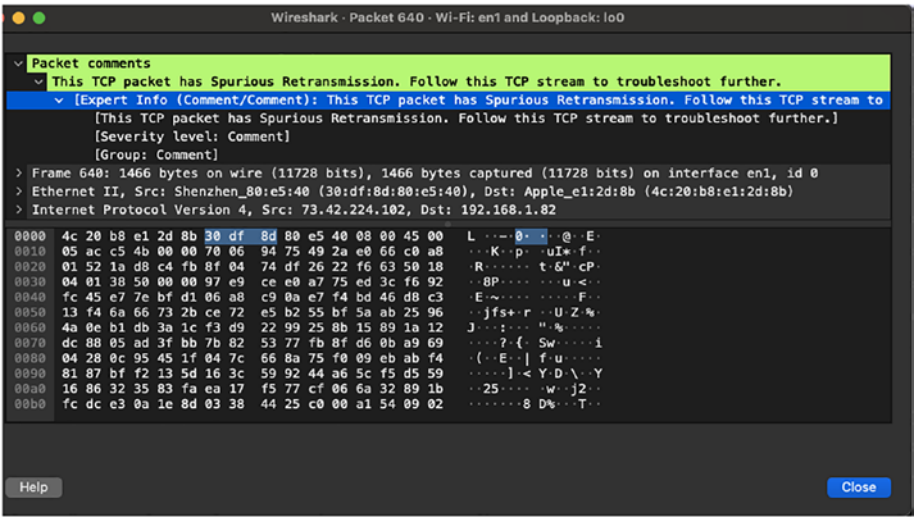


Figure 2-16. *Packet headers with comments*

To add top-level comments or file-level comments, go to Statistics | Capture File Properties. This opens a window that includes a Capture File Comments section. Users can add the comments and then click Save Comments to save the top-level comments.

Metadata

Additional information is always useful when investigating network issues. Although adding top-level and per-packet comments can be extremely useful, additional information such as the source of the packet capture can be very useful. With pcapng, additional fields such as a description field, OS field, and filter field within the interface description block can provide additional information regarding the capture source.

Extendable Format

Because the pcapng format is standardized and deploys a generic block structure, it allows the format to evolve over time. In pcapng, specific

blocks are defined for packets (enhanced packet block or simple packet block) and interfaces (interface description block). Additional information such as metadata can be stored in other optional blocks, such as a name resolution block or interface statistics block. With the options to define experimental blocks and metadata, pcapng allows organizations to develop their own customized yet compatible network analysis tools.

Splitting Packet Captures into Multiple Files

When capturing network traffic on high-speed links, the Wireshark file size can increase rapidly. This could increase the loading time when the packet capture file is opened for analysis. To overcome this challenge, network administrators or analysts can adjust the capture options in Wireshark to automatically split the packet captures into multiple files. Follow these simple steps to do that:

1. On the Capture menu, select Capture Options. This opens the Capture Options window in Wireshark.
2. In the Capture Options window, click the Output tab.
    - On the Output tab, set the capture file from under Capture to a Permanent File by clicking Browse and specifying the file name. Click Save.
    - Choose the output format. The default option is pcapng.
    - Select the Create a New File Automatically check box.
    - You can then select one or multiple options to decide which factors will trigger the creation of a new file. For instance, you can select an option to create a new file after the file has reached 100 packets, as shown in Figure 2-17.

3. Once these options are selected, click Start.

Once you have completed the capture and stopped the capture, you will notice that multiple files have been created.

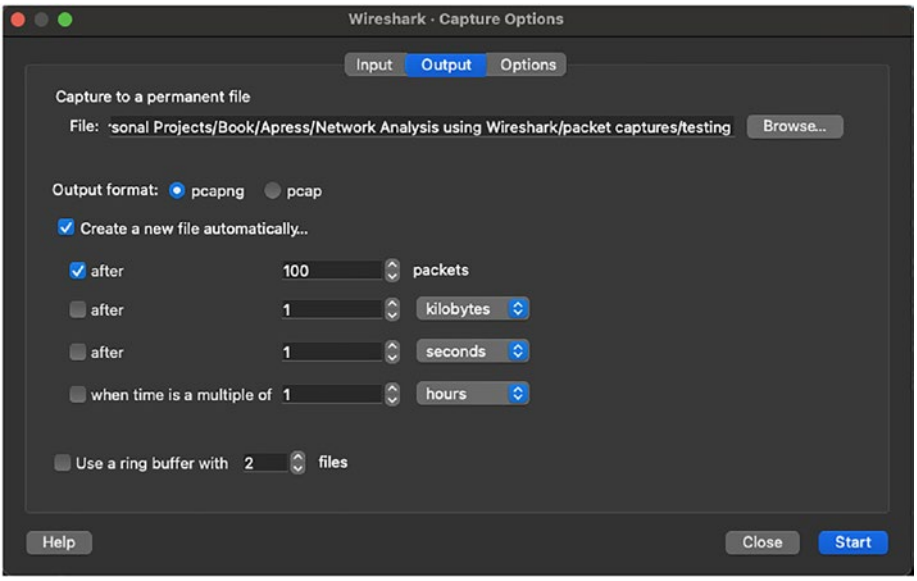


Figure 2-17. *Splitting packet capture into multiple files*

Merging Multiple Capture Files

While splitting helps load the packet capture files quickly, merging is required to analyze the packets, especially in scenarios where captures are taken from multiple interfaces or when the packet stream is split across multiple capture files. The Wireshark merge option tries to merge the files based on one of the following selected options:

- *Prepend Packets:* Prepends the packets from the selected file before the currently loaded packets.

- *Merge Chronologically:* Merge packets from both opened and selected files in chronological order. This option is selected by default.
- *Append Packets:* Appends the packets from the selected file after the currently loaded packets.

To merge multiple files, few simple steps can be followed:

1. Open or load a packet capture file on Wireshark.
2. On the File menu, select Merge to open the Merge dialog box.
3. Select the file that you want to merge with the opened file, as shown in Figure 2-18.

Once the packets are merged, the user can then save the merged file with the same or a different name.

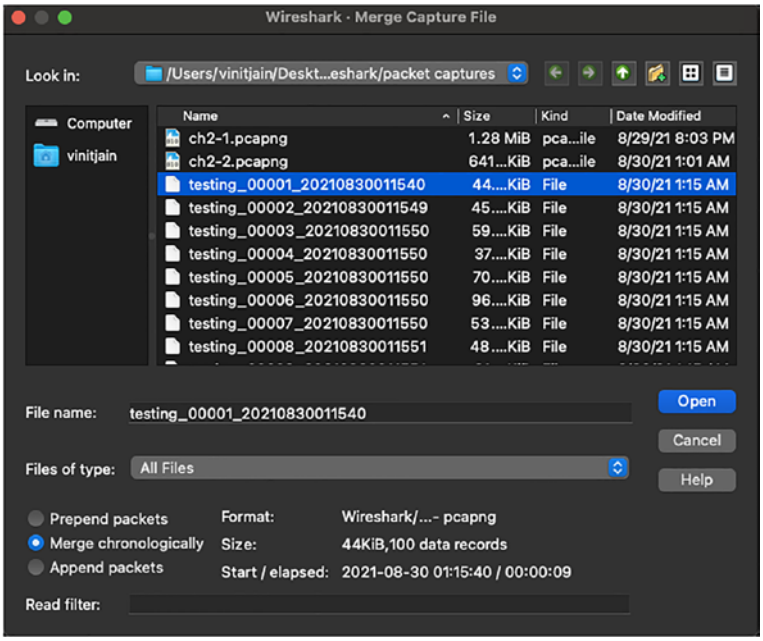


Figure 2-18. *Merging multiple capture files*

Analyzing Packets in Wireshark

Now that we have learned about the basics of the Wireshark UI, how to perform a packet capture, and how to work with capture files, the next step is learning how to analyze the packets using Wireshark. Before we jump into analyzing the packets, there are few critical factors that must be kept in mind, and this goes back to the question of why we need to analyze the packets. Usually, network packet analysis is done when there is a problem in the network and we need to tackle get to the root of any network event. For instance, Company ABC is seeing some anomaly in their network behavior, and they want to investigate what is causing the problem. To get to the root cause of the anomaly, network administrators or security analysts might begin by asking few basic questions such as these:

1. When did the problem start?
2. What is or was the trigger for the problem?
3. Can we re-create the problem?
4. Does the problem happen at a particular time in the day?
5. How frequently does the problem occur in the network?
6. What kind of traffic is affected?
7. Is the issue currently occurring?
8. To which segment of the network is the problem isolated?
9. How many network users are affected due to the given problem?

All these questions, though, might or might not directly answer why we need to perform packet analysis, but these questions will always help get to the bottom of the problem or at least one or few steps closer to it. Out of these questions, the answers to questions 2, 3, 4, 5, 6, and 7 are required when performing network analysis by performing packet captures at different points in the network. If you do not know the answer to question 8, you will eventually find the answer to that question while investigating any network event. Even though, there can be several reasons for performing packet analysis, it is usually done for two primary reasons:

- Baselining the network
- Troubleshooting a network issue (e.g., packet loss, latency issue, network attack, etc.)

OSI Model

Before diving into the steps involved in performing packet analysis, it is important to understand the Open Systems Interconnection (OSI) model. The OSI model was developed by the International Organization for Standardization (ISO) in 1984 with the sole intent of standardizing the communication functions of a telecommunications or computing system irrespective of its underlying structure and technology. The OSI model helps with interoperability across different computers or network devices.

The OSI model outlines the data flow in a network device (or a communication system) through its seven abstraction layers, as shown in Figure 2-19.

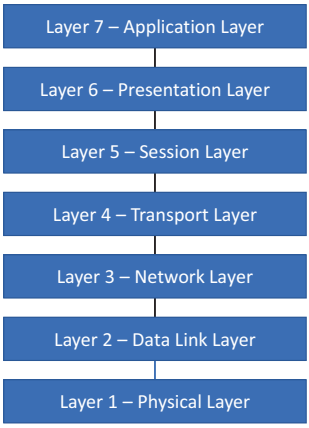


Figure 2-19. *OSI model*

Each layer in the OSI model defines different functions, as listed in Table 2-6.

Table 2-6. *OSI Model Layers and Their Functions*

Layer	Functions
Physical layer	Transits and receives raw bit streams over a physical medium Examples: 1000BaseTX, ISDN, etc.
Data Link layer	Provides reliable transmission of data frames between two devices connected via the physical layer Examples: Ethernet, Frame Relay, ATM, etc.
Network layer	Provides mechanism for structuring and managing a multinode network. The network layer takes care of IP/IPv6 addressing, routing protocols, and traffic control. Examples: IPv4, IPv6, ICMP, IPSEC
Transport layer	Provides reliable transmission of data segments between two points in a network through transport layer protocols. Examples: TCP and UDP

(*continued*)

Table 2-6. (*continued*)

Layer	Functions
Session layer	Manages communication sessions. Examples: NetBIOS, SAP
Presentation layer	Also known as the Translation layer; Provides three primary functions: Translation Encryption/decryption Compression Examples: SSL, TLS, MPEG
Application layer	Provides high-level application programming interfaces (APIs) including resource sharing and remote file access. Examples: FTP, SMTP

To enable communication across each layer, communication protocols enable the communication between two hosts on the same corresponding layer. We learn more about these communication protocols in the coming chapters.

Analyzing Packets

Wireshark organizes the captured packets in an incredibly easy-to-read packet list pane. Once the packets are captured, and if users want to identify the details of the packet, all they need to do is find the packet and click on it. On clicking any packet in the packet list pane, the details about the structure of the packet along with all its fields are visible in the packet details pane. The details displayed in the packet details pane make it incredibly easy to learn and understand more about the packet.

To start analyzing the packets, it is important to first understand the different columns available in the Wireshark UI. Figure 2-20 displays the Wireshark UI and information present across various columns in the packet list pane.

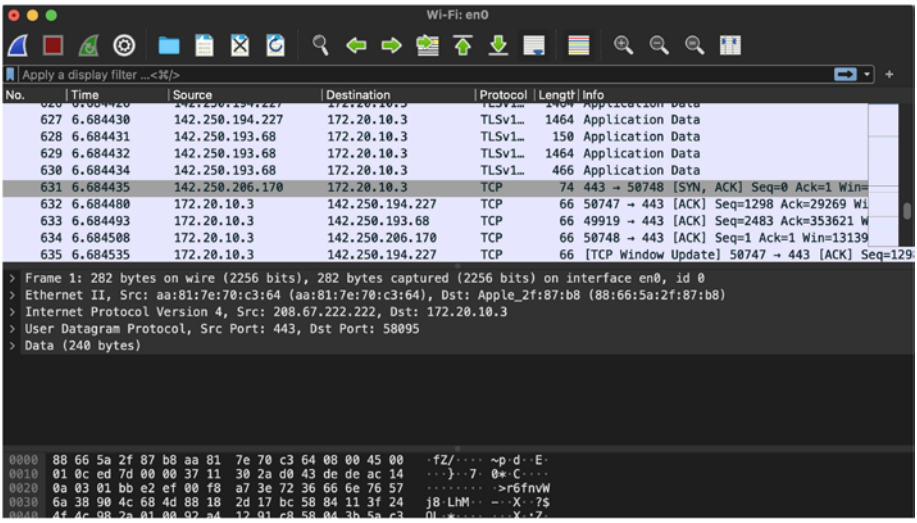


Figure 2-20. *Wireshark user interface*

Here are high-level details about the various columns available at the top of the packet list pane.

- *No.:* This column displays the number order of the captured packet. If there is a bracket displayed along with the packet number, it indicates that the packet is part of the conversation.
- *Time:* This column displays how long after the packet capture was started each packet got captured.
- *Source:* This column displays the source address of the system from where the packet originated.

- *Destination:* This column displays the address of the destination device or host for which the packet is destined.
- *Protocol:* This column displays the type of each packet; for instance, TCP, ICMP, DNS, and so on.
- *Length:* Displays the length of each packet in bytes.
- *Info:* This column displays more information about the packet and could have varied information from packet to packet.

Out of these fields, the Time and Length fields require a bit more explanation, as the rest of the fields are self-explanatory.

Time

As packets are captured in Wireshark, each packet is timestamped. These timestamps are available for each packet in the packet list pane, which can be further used for analysis. It is important to note that the timestamps are created by the Npcap library, but the source of the timestamps is the system's kernel. That is the primary reason timestamps can vary from file to file. Users can choose from one of the following time precision formats in which they wish the timestamps to be displayed:

- Seconds
- Tenths of a second
- Hundredths of a second
- Milliseconds
- Microseconds
- Nanoseconds

Apart from choosing the format of the timestamps, user can also change the display format of the Time column. Users can right-click the column and select the Edit Column option from the menu. That opens a column edit pane just below the display filter bar. In this pane, users can select one of the following time format options from the Type field:

- Time (format as specified); this is the default option
- Absolute date, as YYYY-MM-DD, and time
- Absolute date, as YYYY/DOY, and time
- Absolute time
- Delta time
- Delta time displayed
- Relative time
- UTC date, as YYYY-MM-DD, and time
- UTC date, as YYYY/DOY, and time
- UTC time

After selecting one of these options, click OK in the edit column pane. Figure 2-21 displays the packet list pane with Time column displayed in UTC date and time option.

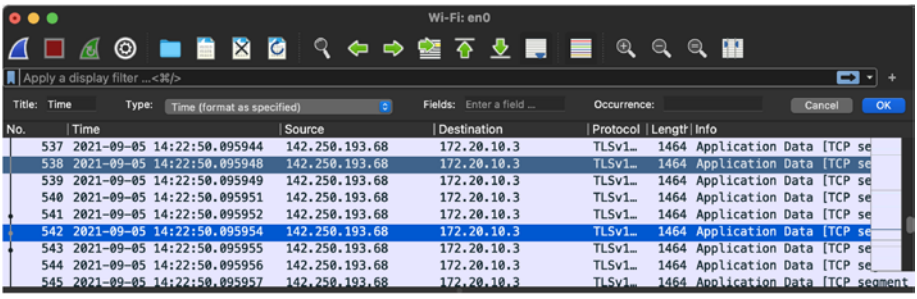


Figure 2-21. *Wireshark with UTC date and time timestamps*

Length

In Wireshark, the length column displays the number of bytes captured for that packet. The number of bytes usually corresponds to the raw data bytes listed at the bottom of the Wireshark window. Now, you must be wondering what is so significant about these captured bytes of the packet. The significance is the statistics that can be gathered from these captured bytes. Based on the captured bytes of each packet, users can examine the distribution of lengths across the captured traffic. To do so, users can go to the Statistics menu and select Packet Length. This will open the Packet Lengths window, which displays the statistical information for varied packet lengths and includes the following columns, as shown in Figure 2-22:

- Packet Lengths
- Count
- Average
- Min Val
- Max Val
- Rate (ms)
- Percent
- Burst Rate
- Burst Start

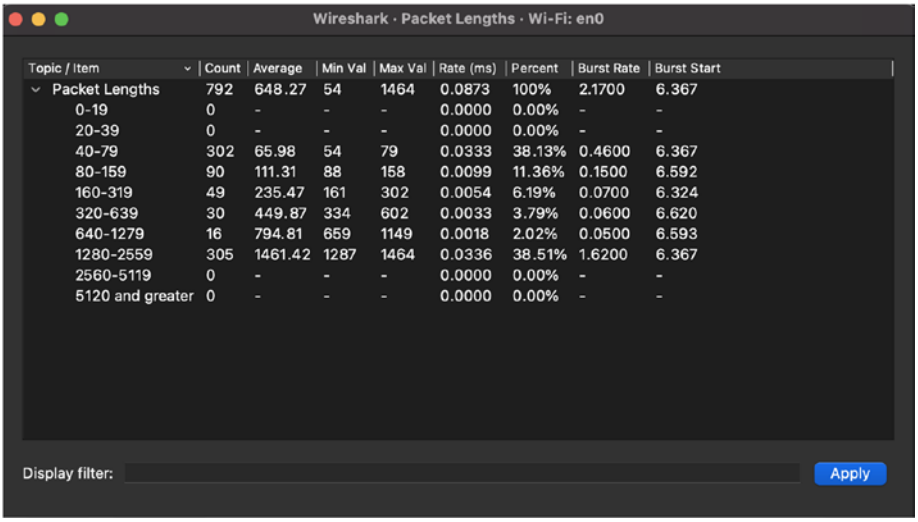


Figure 2-22. *Packet Lengths statistics*

Capture File Properties

Wireshark provides a summarized view of the captured packets in the Capture File Properties dialog box. Users can select Capture File Properties from the Statistics menu. This opens the Capture File Properties dialog box, which shows when the first and the last packets were captured, the device hardware on which the packet was captured, interfaces on which the packets were captured, and statistics from the captured packets as shown in Figure 2-23:

- Packets (total packets captured)
- Time span (total time span for which the capture was running)
- Average pps
- Average packet size in bytes

- Bytes
- Average bytes/second
- Average bits/second

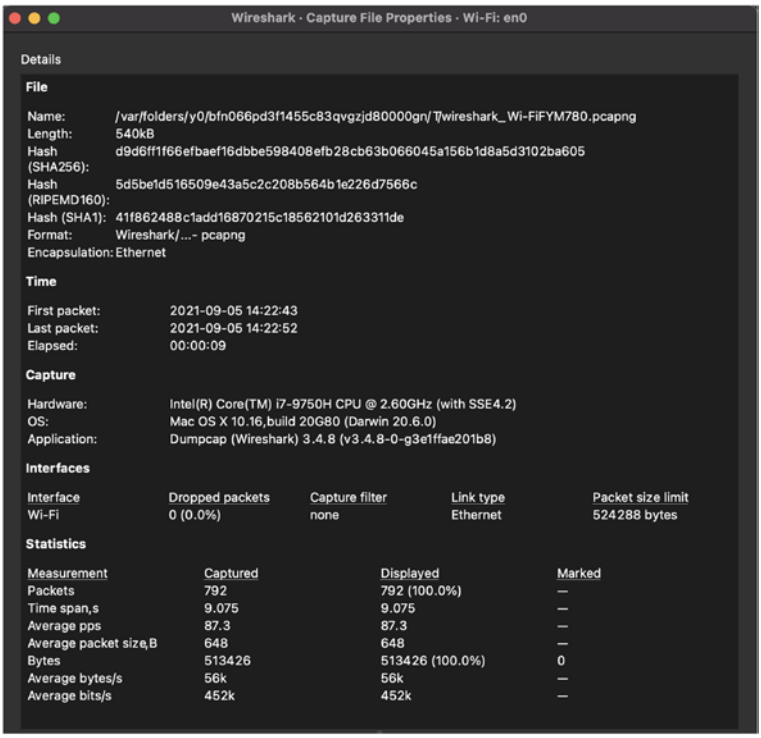


Figure 2-23. *Capture File Properties dialog box*

There is other statistical and deep packet analysis that can be done, but those topics are covered in the coming chapters once we have built a more foundational knowledge on troubleshooting issues with different packet types.

Summary

In this chapter we gained a basic understanding of how to use the Wireshark tool and became familiar with its UI. Initially, we learned about Wireshark preferences and how users can change the default settings and UI according to their requirements and oreferences. We then learned how to perform packet captures and how dissectors play a role in Wireshark to break down packets into a more consumable format. We also covered various filtering techniques, such as capture filters, display filters, and how users can save the filters based on their usage. This chapter also discussed in detail the differences between the `pcap` and the `pcapng` file formats and the information available across these file formats. Finally, we concluded this chapter by learning how to analyze the packet using the Wireshark UI and how various statistical information can be identified from the captured packets.

CHAPTER 3

Analyzing Layer 2 and Layer 3 Traffic

This chapter covers the following topics:

- Layer 2 frames
- Layer 3 packets
- Analyzing QoS markings

Layer 2 Frames

Layer 2 of the OSI or TCP/IP model is the Data Link layer. The Data Link layer is responsible for performing encapsulation of the packets. Appropriate addressing is chosen at each of the Transport, Network, and Data Link layers during the encapsulation process. The Transport layer uses port numbers, the network layer uses IP address, and the Data Link layer uses MAC address, as shown in Figure 3-1.

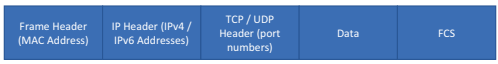


Figure 3-1. *Data encapsulation with different headers*

V. Jain, *Wireshark Fundamentals*, https://doi.org/10.1007/978-1-4842-8002-7_3

At each layer, the data are encapsulated in a specific format known as the packet data unit (PDU). The PDU defines the structure or format in which the data will be shared with the layer above or layer beneath the current layer. The PDU could simply be the data, a segment (at the Transport layer), a packet (at the Network layer), a frame (at the Data Link layer), or even bits.

The Data Link layer encapsulates the outgoing IP datagrams from the network layer and packages them into frames that are transferred between two nodes. This layer is also responsible for correcting any errors that might have occurred at the Physical layer. The Data Link layer has two sublayers:

- *Media Access Control (MAC):* Controls access to the network medium by interfacing with the network adapter. It is responsible for flow control and multiplexing device transmissions over the network.
- *Logical Link Control (LLC):* LLC provides error control and flow control over the physical medium. It is also used for identifying line protocols.

Layer 2 Protocols are required for two devices to communicate over the Layer 2 medium. They provide the communication mechanism between different Layer 2 devices such as NICs, switches, bridges, and more, over a LAN or WAN. There are different types of Layer 2 protocols, some of which are described in Table 3-1.

Table 3-1. *Layer 2 Protocols*

Protocol	Description
Cisco Discovery Protocol (CDP)	CDP is a Cisco proprietary protocol that is primarily used to exchange information between directly connected Cisco devices.
Link Layer Discovery Protocol (LLDP)	LLDP is a vendor-neutral Layer 2 discovery protocol that is commonly used by devices to advertise information to their directly connected devices.
Point-to-Point Protocol (PPP)	PPP provides the standard mechanism for transmitting data over point-to-point links.
Frame Relay	Frame Relay is a packet-switched WAN protocol that operates over the Physical and Data Link layers.
Asynchronous Transfer Mode (ATM)	ATM is a cell-switched WAN protocol that is designed to facilitate various types of traffic streams.
Ethernet	Ethernet is the most widely used Data Link layer protocol used in both LAN and WAN environments.

There are several other protocols that are used at Layer 2, but most of them are now obsolete or have very limited implementation. In this chapter, we focus on Ethernet frames.

Ethernet Frames

When talking about Ethernet frames, we can start by taking a close look at the IEEE 802.3 standard. The fields within the Ethernet header based on IEEE 802.3 standard are described here and shown in Figure 3-2.

- *Preamble:* Ethernet frame starts with a 7-byte Preamble field. Initially this field was introduced to allow for loss of a few bits due to signal delays, but high-speed Ethernet links do not require the Preamble field.
- *Start of frame delimiter (SFD):* SFD is a 1-byte field that is always set to 10101011. This field indicates the start of the frame.
- *Destination Address (DA):* DA is a 6-byte field that holds the destination MAC address of the machine.
- *Source Address (SA):* SA is also a 6-byte field that holds the source MAC address of the machine from which the packet originated.
- *Length:* This 2-byte field indicates the length of the entire Ethernet frame.
- *Data:* The Data section holds the payload of the frame. Note that both the IP header and data will be inserted into this section if IP is being used over Ethernet. The minimum length of the data field is 46 bytes, and the maximum data can be as long as 1,500 bytes, assuming the interface maximum transmission unit (MTU) is set to 1,500. If the data length is less than the minimum length of 46 bytes, then 0s are padded to meet the minimum possible data length.
- *Cyclic Redundancy Check (CRC):* The checksum is computed based on the 32-bit hash code generated using the destination address, source address, length, and data field of the frame and stored in the CRC field. If the checksum computed by the source or sender is not the same as that of the client or receiving device, the data received seem corrupted.

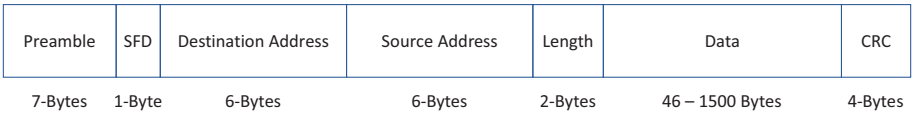


Figure 3-2. *802.3 Ethernet frame*

As stated earlier, the 802.3-based Ethernet frame only supported payloads between 46 bytes and 1,500 bytes, which was good enough for 10 Mbps or 100 Mbps ports, but not helpful for Gigabit Ethernet technology. To support payloads greater than 1,500 bytes, an Extended Ethernet frame was introduced. The Extended Ethernet frame, also known as Ethernet II frame, had the following fields:

- Destination Address
- Source Address
- Type (EtherType)
- Data (Variable size)
- Frame Checksum (FCS)

Figure 3-3 displays the Wireshark capture packet view of an Ethernet frame for an IP packet. Notice that the EtherType in the Type field is set to 0x0800, which indicates the encapsulated packet is an IPv4 packet.

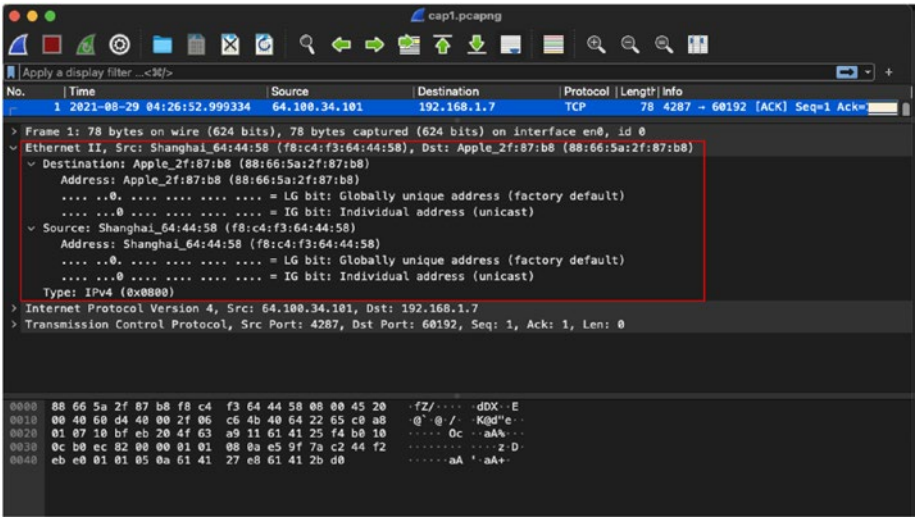


Figure 3-3. *Ethernet II header in IPv4 packet*

Note By default, FCS is not visible in the Wireshark capture. To view the FCS for the Ethernet header, go to Wireshark ➤ Preferences ➤ Protocols ➤ Ethernet and enable Assume Packets Have FCS option. Once that option is enabled, the Wireshark packet detail view will display the FCS field under the Ethernet II header.

When investigating packets at Layer 2, it is important to take note of some well-known EtherTypes that are seen inside most production networks. Some of these well-known EtherTypes are listed in Table 3-2.

Table 3-2. *Well-Known EtherTypes*

EtherType	Protocol
0x0800	IPv4
0x0806	ARP
0x8100	VLAN-Tagged Frame (IEEE 802.1Q)
0x8847	MPLS
0x86DD	IPv6

When analyzing Layer 2 frames, packets in Wireshark can be filtered by applying filters based on source MAC, destination MAC, or EtherType. Figure 3-4 displays the filtering of Layer 2 ARP broadcast frames coming from a specific MAC address. The filters can be applied on `eth.src` and `eth.type` fields on the captured Wireshark packets.

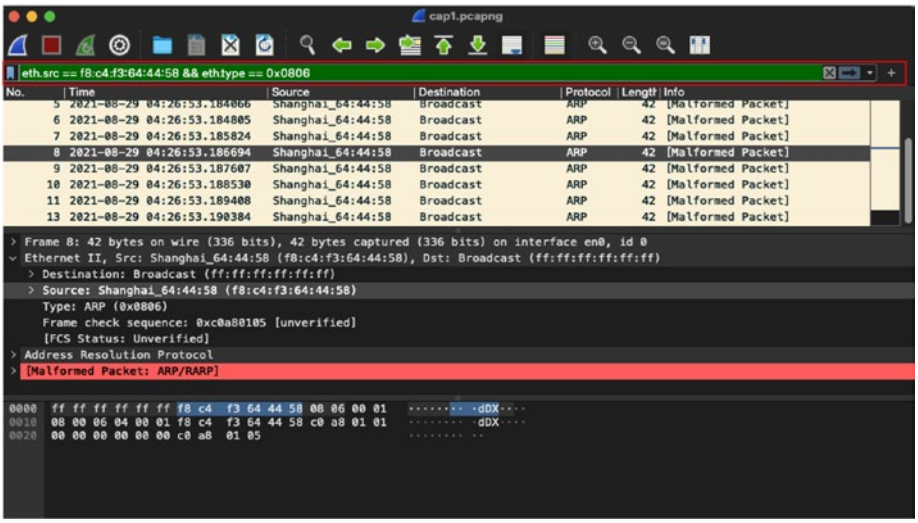


Figure 3-4. *Filtering of broadcast frames sourced from a specific MAC address*

Layer 3 Packets

When troubleshooting issues within a single broadcast domain or local LAN environments, Layer-2-based captures are more relevant, but when investigating issues that span multiple network segments that might be residing in different geographical location, we primarily focus on looking at Layer 3 and upper layer information in the packet captures. When talking about Layer 3 packets, we are primarily referring to either IPv4 packets or IPv6 packets. The protocols at Layer 3 provide logical addressing in a network (Internet or intranet) and ensure routing of data across different network segments. Even when dealing with tunneling technologies, the logical addressing of the tunnel interfaces and routing traffic across tunnel interfaces is still required. Before diving into IPv4 or IPv6 packets, let's first understand ARP and its importance for establishing network communication.

Address Resolution Protocol

It is important to remember that both physical and logical addresses are required to establish communication in the network. Logical addresses allow users to establish communication across multiple network segments, and physical addresses are used for establishing communication within the same network segment. To forward traffic within the same broadcast segment, MAC addresses are required by the switch. Unless the switch knows about the MAC address of the host, it will not be able to forward the traffic toward the port where the destination host is connected. The MAC addresses of all the hosts connected to a switch within the same broadcast domain are stored in a *Content Addressable Memory* (CAM) table. If the MAC address of the destination address is not known, the switch will first perform a lookup in its cache. If the address is not found even in the cache, then a request is flooded to all the ports within the same broadcast domain until the MAC address is identified.

TCP/IP uses ARP to map the IPv4 address with the MAC address. The ARP protocol functionality, defined in RFC 826, relies on primarily two packets:

- *ARP request:* An ARP request is basically a broadcast packet that is initiated by the sender or source host when it does not know the MAC address of the destination host or receiver.
- *ARP response:* When a host with the destination IP address for which the ARP request was sent receives the ARP request, it replies with an ARP response, which is basically a unicast packet directed toward the sender or the source host.

To understand how ARP works, examine the topology shown in Figure 3-5. In this topology, Host 1, Host 2, Host 3, and Host 4 are connected to an L2 switch and Host 1 wishes to send data toward Host 2. Because Host 1 does not know the MAC address of Host 2, it broadcasts an ARP request toward the switch.

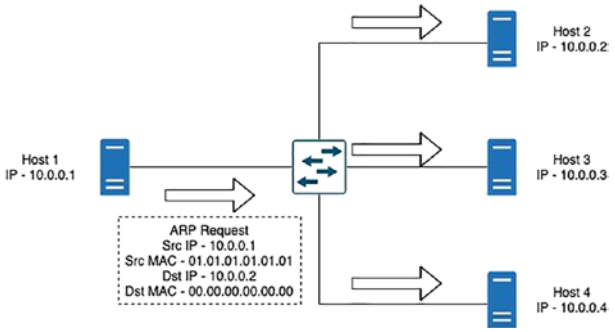


Figure 3-5. *ARP request*

On receiving the ARP request, the switch first updates its MAC address table with the MAC address of Host 1 (if it doesn't already know about the MAC address) and then broadcasts the ARP request to all the hosts within the same VLAN. All three hosts receive the ARP request. Because Host 2 holds the destination IP address in the ARP request, it updates its MAC ARP cache and sends an ARP response toward Host 1 as shown in Figure 3-6. The reply is a unicast reply because Host 2 knows its own MAC address but it also knows about the MAC address of Host 1, which it learned via the ARP request.

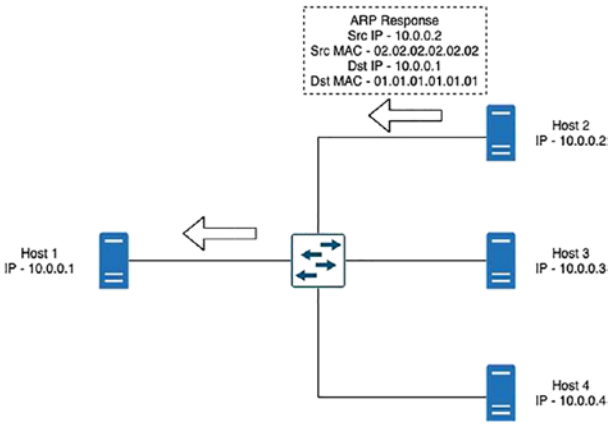


Figure 3-6. *ARP response*

Any further communication between Host 1 and Host 2 will be unicast unless one of them has its ARP entry time out. Figure 3-7 and Figure 3-8 show the Wireshark capture of the ARP request and ARP response.

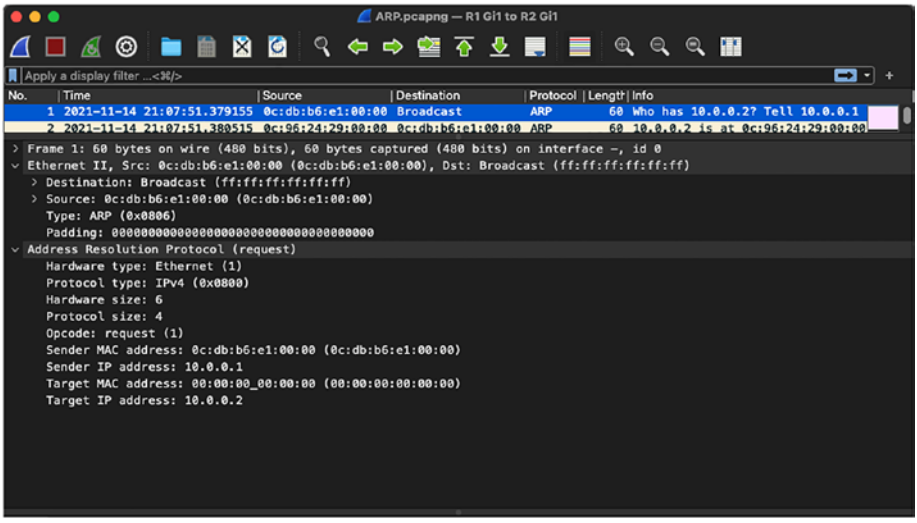


Figure 3-7. *Wireshark capture ARP request*

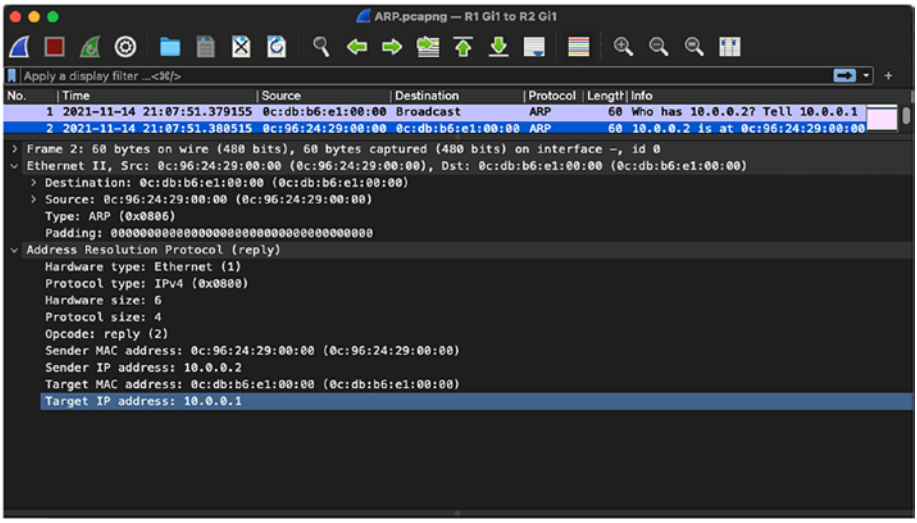


Figure 3-8. *Wireshark capture ARP response*

IPv4 Packets

Defined in RFC 791, The *Internet Protocol version 4* (IPv4) address is a 32-bit address that could allow anywhere from two to hundreds and thousands of hosts to be in each network segment. The way subnets are implemented in organizations allow them to scale quickly without having to make much change in routing. As stated before, the purpose of IP was to deliver logical addressing for various network elements and to provide routing capability across different network segments. Because there was not much advancement that happened in the early 1980s and there were limited bandwidth options available, when IPv4 addressing was standardized in 1983, one of the foci was solving the fragmentation problem that would allow the packets to be broken into smaller chunks. Today, with Gigabit Ethernet technologies in place and support for jumbo MTUs, we rarely have to deal with fragmentation in the network. All the fields of the IPv4 header are shown in Figure 3-9.

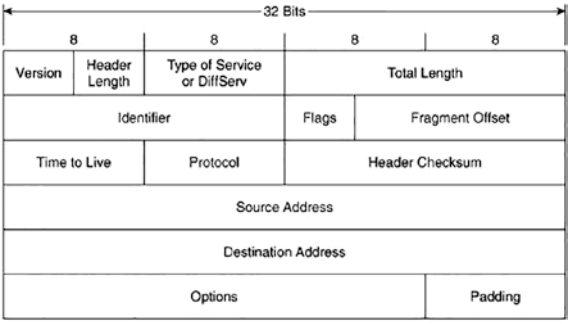


Figure 3-9. *IPv4 header*

To understand the capabilities of IPv4, let's examine the IPv4 header in detail. The IPv4 address has several fields as shown in Figure 3-9:

- *Version:* This 4-bit field indicates the IP version being used. There are devices that could be dual stack (support for IPv4 and IPv6 address) and the version field helps the device understand how to treat the traffic.

- *Internet Header Length (IHL):* This is a 4-bit field that contains the size of the IPv4 header. The 4 bits are used to specify the number of 32-bit words in the header. The minimum value of this field is 5 and the maximum value is 15, which basically indicates that the minimum IPv4 header length can be 20 bytes and the maximum can be 60 bytes.
- *Differentiated Services Code Point (DSCP):* This is a 6-bit field that was previously known as a Type of Service (ToS) field. This field specifies differentiated services (DiffServ), defined in RFC 2474, and it is used to provide service quality features such as Voice over IP (VoIP) calls or data streaming. Based on the values assigned in this field, different traffic streams are given different priority in the network and treated differently by routers and switches.
- *Explicit Congestion Notification (ECN):* ECN is a 2-bit field that allows for end-to-end network congestion notification without dropping packets. For the ECN feature to work, both endpoints are required to support this feature.
- *Total Length:* This is a 16-bit field that defines the entire packet size in bytes including the header and payload. The minimum size is 20 bytes and the maximum size is 65,535 bytes.
- *Identification:* The Identification field is used to identify a group of IP datagram packets uniquely and is also widely used for packet tracing purposes.

- *Flags:* This 3-bit field is used to control and identify fragments of an IP datagram. There are three possible values that are set in the Flags field:
  - Bit 0: Reserved
  - Bit 1: Do not fragment (also known as DF bit)
  - Bit 2: More fragments
- *Fragment Offset:* This 13-bit field specifies the fragment offset relative to the start of the original unfragmented IP datagram in blocks. Each block is measured in units of 8 bytes. The maximum possible offset is 65,528 ($(2^{13} - 1) * 8$).
- *Time to Live (TTL):* TTL is an 8-bit field that indicates the maximum time that a packet can live in an Internet system. The maximum value of TTL is 255 seconds and it is decremented when a packet is processed at each routed hop and forwarded to the next hop. If the TTL value is zero (0), the packet is discarded or dropped. This is to ensure that the packets do not keep looping in the Internet system.
- *Protocol:* The 8-bit Protocol field is used to denote which protocol will be used in the data section of the datagram. For instance, the two most common protocol numbers that are usually seen in the network are protocol number 6, which is used to represent TCP, and protocol number 17, which represents a UDP packet. The protocol numbers are assigned and maintained by the Internet Assigned Numbers Authority (IANA).

- *Header Checksum:* The 16-bit Header Checksum field in the IPv4 header is used for validating the integrity of the packet. When an IPv4 packet arrives at the router, the router calculates the checksum of the packet and compares it with the value in the this field. If the value matches, the packet is forwarded; otherwise, the packet is dropped.
- *Source Address (SA):* The 32-bit Source Address is used to specify the IPv4 address of the source device that originated the packet.
- *Destination Address (DA):* The 32-bit Destination Address field is used to specify the IPv4 address of the destination device to which the packet is destined.
- *Options:* The Options field is an optional field that is only set when the IHL value is greater than 5 (i.e., between 6 and 15). The Options field contains values and settings for security-related options and might be considered dangerous by some routers and dropped. You might see the Options field set when using the Record Route option with extended ICMP pings or for Timestamps. Table 3-3 shows the list of options that can be used in an IPv4 header and Table 3-4 displays the defined options for IPv4.

Table 3-3. *IPv4 Header Options*

Field	Size (Bits)	Description
Copied	1	Set to 1 if the options need to be copied across all fragments of a fragmented packet
Option Class	2	0 – Control Options 1 – Reserved 2 – Debugging and Measurement 3 – Reserved
Option Number	5	Specifies an option
Option Length	8	Indicates the size of the entire option; might not be set for simple options
Option Data	Variable	Holds option specific data; might not be set for simple options

Table 3-4. *Defined Options for IPv4*

Option Type (Decimal/Hexadecimal)	Option Name	Description
0/0x00	EOOL	End of Option List
1/0x01	NOP	No Operation
2/0x02	SEC	Security (defunct)
7/0x07	RR	Record Route
10/0x0A	ZSU	Experimental Measurement
11/0x0B	MTUP	MTU Probe
12/0x0C	MTUR	MTU Reply
15/0x0F	ENCODE	ENCODE
25/0x19	QS	Quick-Start

(*continued*)

Table 3-4. (*continued*)

Option Type (Decimal/Hexadecimal)	Option Name	Description
30/0x1E	EXP	RFC 3692-style Experiment
68/0x44	TS	Timestamp
82/0x52	TR	Traceroute
94/0x5E	EXP	RFC 3692-style Experiment
130/0x82	SEC	Security (RIPSO)
131/0x83	LSR	Loose Source Route
133/0x85	E-SEC	Extended Security (RIPSO)
134/0x86	CIPSO	Commercial IP Security Option
136/0x88	SID	Stream ID
137/0x89	SSR	Strict Source Route
142/0x8E	VISA	Experimental Access Control
144/0x90	IMITD	IMI Traffic Descriptor
145/0x91	EIP	Extended Internet Protocol
147/0x93	ADDEXT	Address Extension
148/0x94	RTRALT	Router Alert
149/0x95	SDB	Selective Directed Broadcast
151/0x97	DPS	Dynamic Packet State
152/0x98	UMP	Upstream Multicast Packet
158/0x9E	EXP	RFC 3692-style Experiment
205/0xCD	FINN	Experimental Flow Control
222/0xDE	EXP	RFC 3692-style Experiment

- *Data:* The data or payload in the Data field is based on the value set in the Protocol field of IPv4 header. For instance, if the protocol number is set to 1, then the payload will contain ICMP-related data.

Examine Figure 3-10, which displays the Wireshark capture of an IPv4 packet detailing all the IPv4 header fields. Notice that the Options field is not present in this capture.

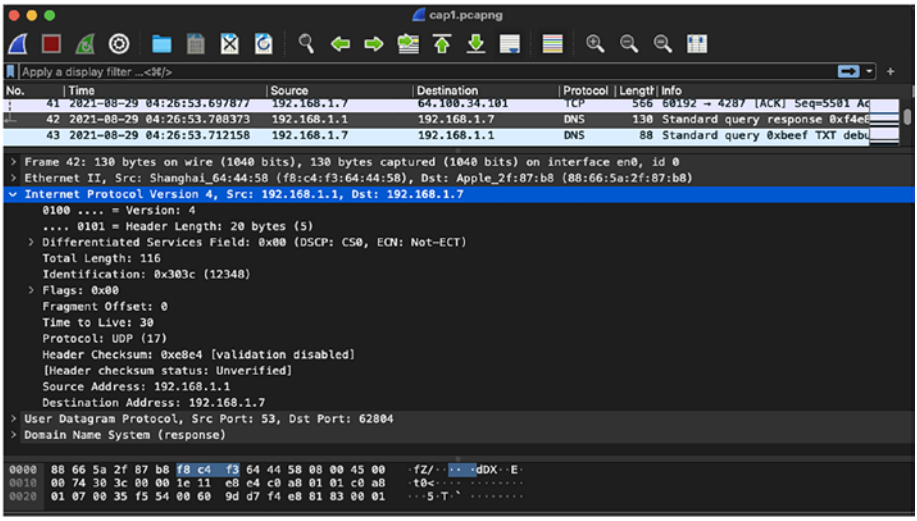


Figure 3-10. *IPv4 header in Wireshark capture*

IPv4 Addressing

Based on the understanding from the IPv4 header, the Source Address and the Destination Address fields store the source and destination IPv4 addresses, respectively. Although these fields are important, it is equally important to understand different types of IPv4 addresses. Primarily, IPv4 addressing is divided into five different classes:

- *Class A:* 0.0.0.0 to 127.255.255.255
- *Class B:* 128.0.0.0 to 191.255.255.255

- *Class C:* 192.0.0.0 to 223.255.255.255
- *Class D (multicast addresses):* 224.0.0.0 to 239.255.255.255
- *Class E (experimental addresses):* 240.0.0.0 to 255.255.255.255

Within Classes A, B, and C, the IPv4 addresses are further divided into public and private addresses.

- *Public address:* Public IPv4 addresses are the addresses that are uniquely identified on the Internet and are usually allocated to organizations by IANA.
- *Private addresses:* Private IPv4 addresses are primarily used in almost every organization for managing hosts in LAN environments. These addresses are not advertised in the global Internet routing table. The private IPv4 address range is shown here:
  - *Class A private IP:* 10.0.0.0 to 10.255.255.255
  - *Class B private IP:* 172.16.0.0 to 172.31.255.255
  - *Class C private IP:* 192.168.0.0 to 192.168.255.255

RFC 1918 defines the range of private addresses that can be used by organizations within their LAN environments. Because the private addresses are inherently private, multiple organizations could have the same addressing schemes within their organizations. They communicate to outside networks (Internet) through their public IPv4 addresses, which are allocated to their Internet Gateway routers. Even a home broadband connection works on the same concept. The local network sits behind a modem and the hosts that are part of that network are allocated private IPv4 addresses. These hosts have their default gateway set to the modem and when the hosts want to communicate to the Internet, they use the

default route pointing to the gateway. The modem has a dynamically learned or statically assigned IPv4 address that allows the broadband user to access the Internet. Figure 3-11 illustrates how a simple broadband connection is set up.

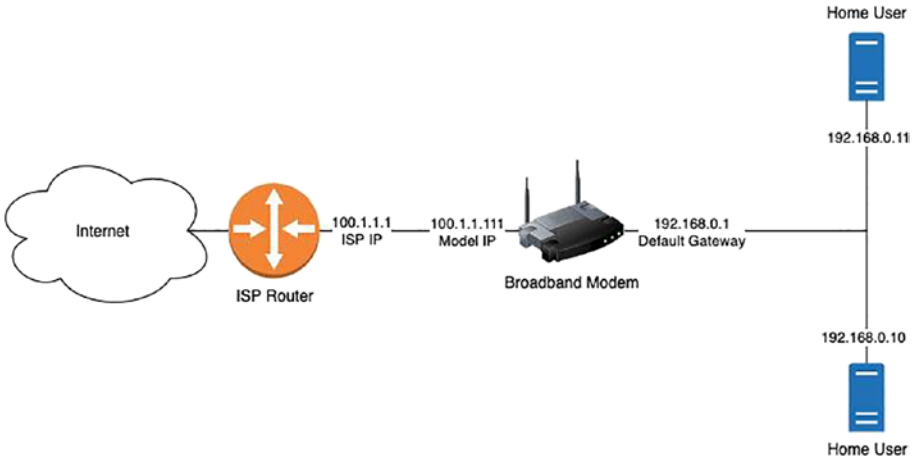


Figure 3-11. *Home broadband Internet connection*

There are other addresses, too, apart from the public and private IPv4 addresses, listed here:

- *Loopback addresses:* 127.0.0.0 to 127.255.255.255
- *APIPA:* 169.254.0.0 to 169.254.255.255
- *Limited broadcast:* 255.255.255.255

ICMP

ICMP is one of the key protocols that is used for validating connectivity in the network and for network troubleshooting. ICMP was initially published in RFC 777, which was later deprecated by RFC 792 and was later updated by RFC 4884, RFC 6633, RFC 6918, and so on. Because IP is a best-effort and an unreliable connectionless protocol, ICMP allows identifying and

communicating issues that prevent data delivery. Network engineers frequently use the ICMP ping utility, which relies on an ICMP request and ICMP reply to identify reachability between source and destination. There are two versions of ICMP that are used by both the IP versions:

- ICMPv4 used for IPv4
- ICMPv6 used for IPv6

To communicate errors or reachability information, ICMP relies on ICMP messages that are set within the ICMP header. The ICMP header has the following fields, shown in Figure 3-12:

- *Type:* 8-bit
- *Code:* 8-bit
- *Checksum:* 16-bit

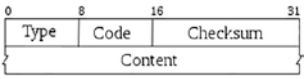


Figure 3-12. *ICMP header*

Note When sending an ICMP packet, the Protocol field within the IP header is set to a value of 1.

Some of the most commonly used ICMP messages are as follows:

- *ICMP Echo Request and Echo Reply:* The ICMP Echo Request has the Type/Code value of 8/0 and the ICMP Echo Reply has the Type/Code value of 0/0. The Echo Request and Echo Reply messages are used to validate the connectivity between the source and destination device in the network and are commonly used via the ICMP Packet InterNet Grouper (PING) tool. When the source device tries to verify the connectivity

toward the destination device using the PING tool, it sends an ICMP request, and if the destination device is reachable, it responds back with the ICMP reply message.

- *ICMP Redirect message:* ICMP Redirect messages are used by routers on nonoptimal paths to notify hosts about the availability of an optimal path between the source and the destination. An ICMP Redirect message has the ICMP Type value of 5 and has four codes:
    - Code 0: Redirect datagram for the network
    - Code 1: Redirect datagram for the host
    - Code 2: Redirect datagram for the type of service and network
    - Code 3: Redirect datagram for the type of service and host
- *ICMP Destination Unreachable message:* If a router receives a datagram that it is unable to forward or deliver to the destination, it replies with an ICMP Destination Unreachable message. There can be multiple reasons for the router being unable to deliver the packet. The different reasons are covered under various ICMP codes. The ICMP Destination Unreachable message has the Type value of 3 and the following code options:
    - Code 0: Destination network unreachable
    - Code 1: Destination host unreachable
    - Code 2: Destination protocol unreachable
    - Code 3: Destination port unreachable

  - Code 4: Fragmentation required, and DF set
  - Code 5: Source route failed
  - Code 6: Destination network unknown
  - Code 7: Destination host unknown
  - Code 8: Source host isolated
  - Code 9: Network administratively prohibited
  - Code 10: Host administratively prohibited
  - Code 11: Network unreachable for type of service
  - Code 12: Host unreachable for type of service
  - Code 13: Administratively prohibited
- *ICMP Time Exceeded message:* This message is sent by the router to the source device or host if the TTL value reaches 0 before it reaches the destination. One reason that could cause the TTL to expire is that the destination router is more than 255 hops away or, alternatively, there is a routing loop in the network that has caused the TTL value to reach 0. The ICMP Time Exceeded message has the Type value of 11 and has the following codes:
  - Code 0: TTL expired
  - Code 1: Fragment reassembly time exceeded
- *ICMP Source Quench message:* If a router receives a large amount of data that it can handle and it can send an ICMP Source Quench message to the sender asking it to slow down the rate at which it is sending the traffic. The ICMP Source Quench message has the Type and Code value set to 0/0.

To see what ICMP packets look like, let's examine the network topology shown in Figure 3-13. In this topology, there are three routers—R1, R2, and R3—each configured with loopback interfaces.

Figure 3-13. *Network topology*

The goal is to test the connectivity between the R1 and R3 loopback address. To exchange the routing information, the device is running the OSPF routing protocol. To verify connectivity, the network engineers will initiate a ping request destined to R3 loopback 0 interface IP (i.e., 3.3.3.3) sourcing loopback 0 interface IP (i.e., 1.1.1.1). Figure 3-14 and Figure 3-15 display the ICMP request and ICMP reply packets between 1.1.1.1 and 3.3.3.3.

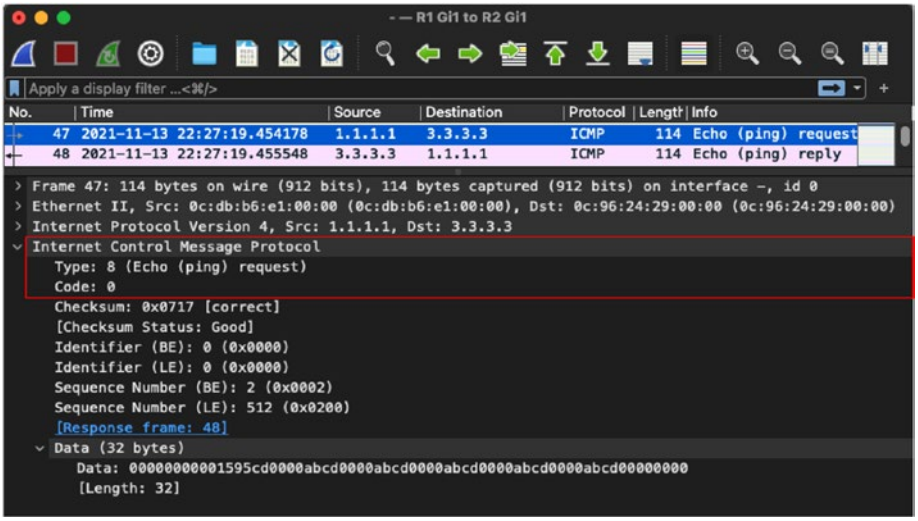


Figure 3-14. *Wireshark capture for ICMP request*

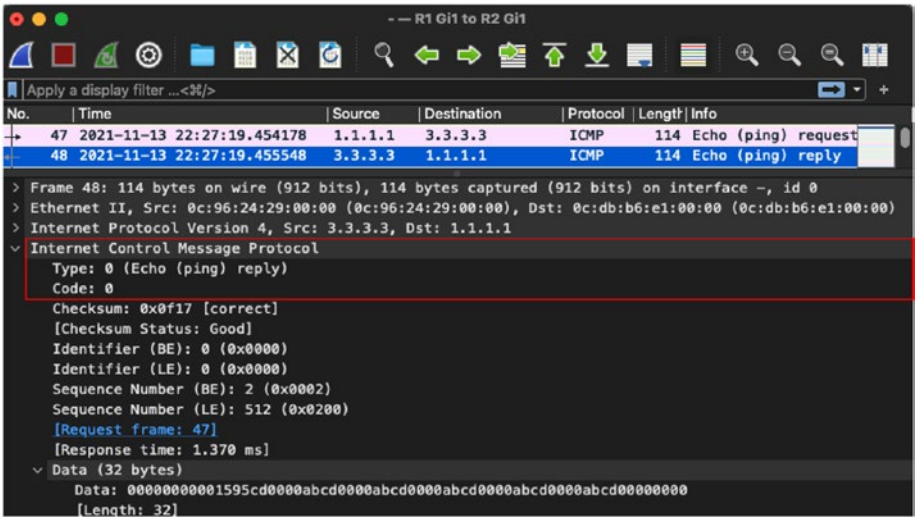


Figure 3-15. *Wireshark capture for ICMP reply*

Although in most cases you might not want to perform a packet capture for ICMP request and ICMP reply packets, in scenarios where even the ICMP packets are unreachable, you might want to perform packet capture to isolate the node that might be dropping the packets.

For basic ICMP packets, there is no special Options field that must be set in the IP header, but when performing a ping test with the Record Route (RR) option, the Options field is set to enable the RR feature. But what is the RR feature? It is used mostly with the ping tool. When the RR option is set in the Options field of an IPv4 header, it forces every router that handles the IP datagram to add its outgoing interface IP to a list in the Options field. When the datagram reaches the destination, the list of IP addresses is copied into the ICMP reply, which is then returned to the sender. Example 3-1 illustrates how the RR option is used with the ping request on a Cisco IOS device.

Example 3-1. ICMP Ping with Record Route Option

```
R1#ping
Protocol [ip]:
Target IP address: 3.3.3.3
Repeat count [5]: 1
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Ingress ping [n]:
Source address or interface: 1.1.1.1
DSCP Value [0]:
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0x0000ABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]: Record
Number of hops [ 9 ]:
Loose, Strict, Record, Timestamp, Verbose[RV]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 1, 100-byte ICMP Echos to 3.3.3.3, timeout is 2
seconds:
Packet sent with a source address of 1.1.1.1
Packet has IP options:  Total option bytes= 39, padded
length=40
```

```
 Record route: <*>
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)

Reply to request 0 (71 ms).  Received packet has options
 Total option bytes= 40, padded length=40
 Record route:
   (10.1.2.1)
   (10.2.3.2)
   (3.3.3.3)
   (10.1.2.2) <*>
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
 End of list
```

When examining the ICMP packet in Wireshark as shown in Figure 3-16, you will also notice that the IPv4 header now has the Options field with the Record Route option set and the list of IP addresses.

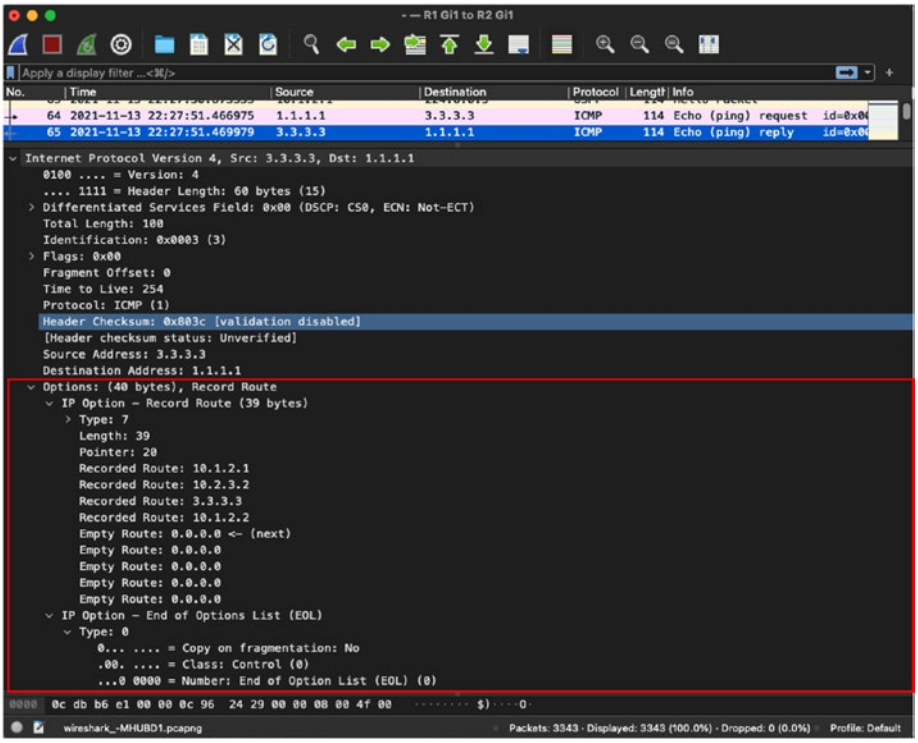


Figure 3-16. *Wireshark capture of ping packet with Record Route option*

IP Fragmentation and Reassembly

Fragmentation allows the larger sized packets to be broken down into smaller chunks to be sent across network segments that do not support large packets. When traffic is routed across the network devices, the devices in the path might encounter a segment with an interface MTU value that is smaller than the incoming packet size. For such situations, IP fragmentation can be used. It allows a datagram to be divided into

smaller chunks so that it can be transmitted across a segments that have a lower interface MTU. When talking about fragmentation, two terms are widely used:

- Maximum segment size (MSS): Data payload
- MTU – MSS + IP header (20 bytes) + TCP header (20 bytes)

So, if an interface MTU is set to a default valie of 1500, then the MSS will be calculated as follows:

MSS = 1500 (MTU) – 20 bytes (IP header) – 20 bytes (TCP header)

For a better understanding of how fragmentation works, examine topology shown earlier in Figure 3-13. In this topology, the link between R1 and R2 has an interface MTU value set to 9200, whereas the link between R2 and R3 has the interface MTU value set to the default of 1500. When a ping packet is initiated from R1 sourcing 1.1.1.1 destined to R3 loopback 3.3.3.3 with packet size set to 9140, the packet will be fragmented because the network segment between R2 and R3 has a lower MTU value. In such a case, the more fragments bit will be set in the IPv4 header Flags field as shown in Figure 3-17. In Figure 3-17, we can see the more fragments bit is set but also notice that the Fragment Offset field shows 0. This is because the first datagram packet in the fragmented packets will not have any offset, thus it begins from 0. Any other packet after this packet will have incremental fragment offset values.

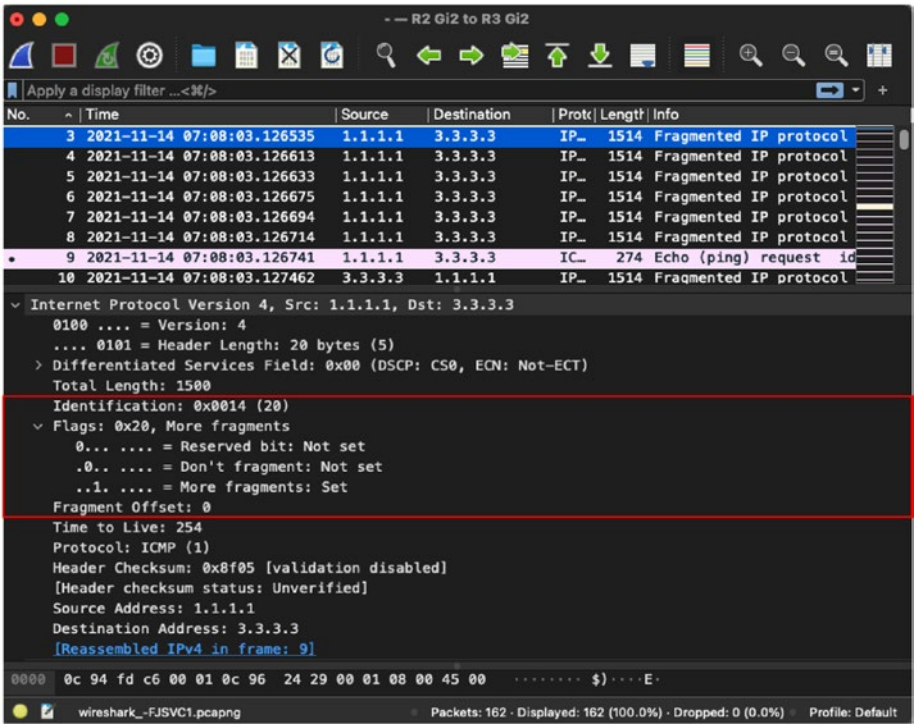


Figure 3-17. *Wireshark capture of first fragmented packets*

Similarly, if we notice the last fragment of the fragmented packets, we can see that the more fragments bit is not set anymore, but the Fragment Offset field is set to 8880, as shown in Figure 3-18. This means that the all the previous fragments cumulatively hold 8880 bytes of data in the payload. Note that the IP Identification field will have the same value across all the fragmented packets of the single large packet. Wireshark also displays all the packets that will be used for fragment reassembly, which will combine to form the final packet of payload size 9140 bytes.

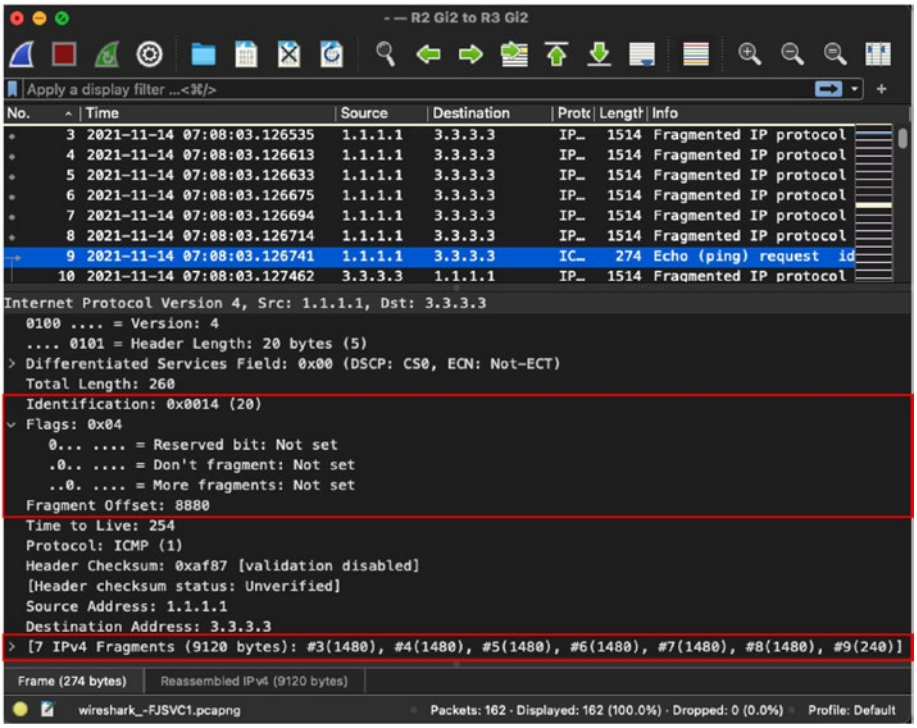


Figure 3-18. *Wireshark capture of last fragmented packet*

For simplicity and ease of remembering, the IP packets can be categorized into three types:

- Nonfragmented
- Initial fragment
- Noninitial fragment

Based on these packet types, the more fragments bit and Fragment Offset field are set as shown in Table 3-5.

Table 3-5. *Fragment Settings in IP Header*

Packet Type	More Fragments Flag	Fragment Offset Field
Nonfragmented	0	0
Initial fragment	1	0
Noninitial fragment (not last)	1	Nonzero
Noninitial fragment (last)	0	Nonzero

In today's networks, we rarely notice fragmentation as most networks these days are designed with support for jumbo MTU (MTU values higher than 1500 bytes). All the Gigabit Ethernet networks support up to 9216 bytes to be set as the MTU value and to leverage the benefit of higher MTU sizes, most applications send the packets with the don't fragment (do not fragment or DF) bit set. When this bit is set, the router processing the packet will not fragment the packet and will try to send the packet out as is. When a packet is forwarded across a network segment with the DF bit set and the router encounters a next hop interface having lower MTU or IP MTU settings, the router will send an ICMP Destination Unreachable message back with type 3 and code 4, which basically means fragmentation is needed and the DF bit is set. Along with the ICMP Destination Unreachable message, the router also sends the MTU settings of the next hop device to let the source know that it needs to send the packet with a smaller data payload if it wants to send the packet with the DF bit set. Figure 3-19 displays the Wireshark capture of the ICMP Destination Unreachable message along with the MTU settings of the next hop device.

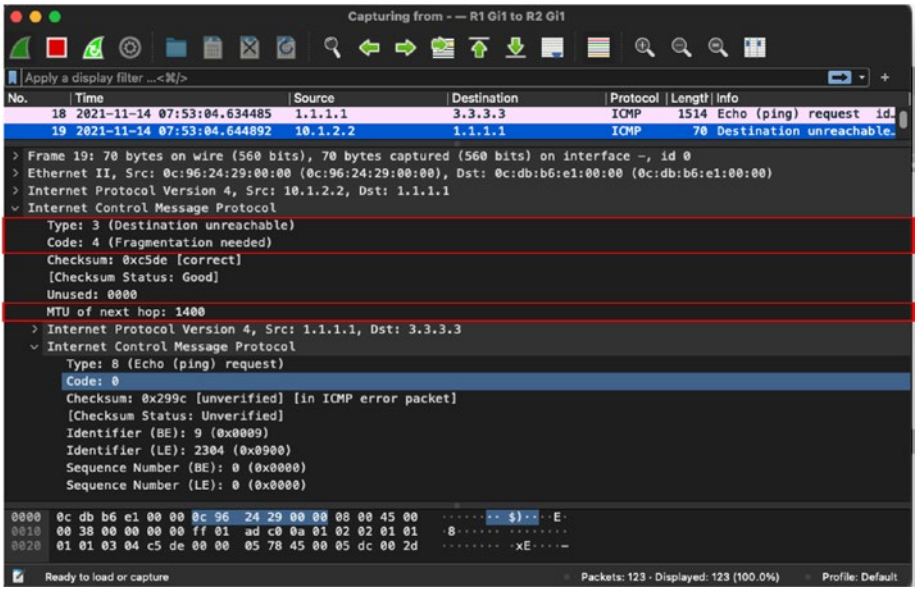


Figure 3-19. *Wireshark capture of ICMP Destination Unreachable message*

IPv6 Packets

When IPv4 was introduced by DARPA, the 32-bit address space would allow for 2^{32} (4.3 billion) addresses, which seemed sufficient at the time. As networks and the number of devices grew, however, it became evident that the IPv4 address space would not be sufficient based on the number of devices per person. Scientists then developed the IPv6 addressing scheme and increased the address size limit from 32 bits to 128 bits. In 1998, IPv6 addressing was standardized in RFC 2460. IPv6 addressing had a number of enhancements over IPv4:

- *Streamlined header:* Although the IPv6 header is much larger than the IPv4 header, several fields from the IPv4 header were removed in the IPv6 header, making it more streamlined.

- *Revised fields:* Some of the fields in the IPv6 header were revised when compared to Ipv4:
  - The TTL field in IPv4 was converted to the Hop Limit field in IPv6.
  - The Precedence and ToS fields were moved to the Traffic Class field.
  - The Protocol field was covered under the Next Header field.
  - The 32-bit Source Address and Destination Address fields were now converted to 128-bit Source Address and Destination Address fields.
- *Flow label:* Flow label was introduced in the IPv6 header for identifying streams such as real-time traffic that required special treatment in the network.

Figure 3-20 shows the comparison between an IPv4 and an IPv6 header. Notice that even though the the IPv4 header has more field, the IPv6 header looks more streamlined.

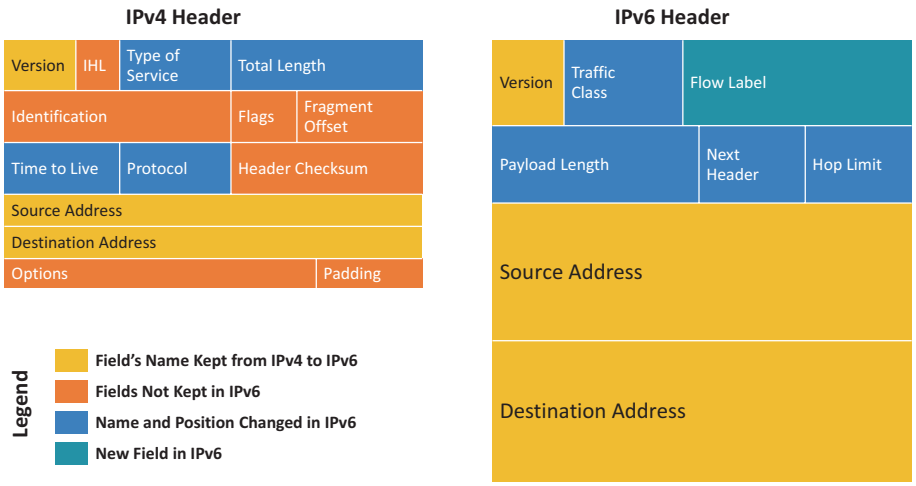


Figure 3-20. *IPv4 and IPv6 headers*

Figure 3-21 shows the Wireshark capture of an IPv6 packet exchange between two devices. The highlighted section of the Wireshark capture shows the IPv6 header. Let's now examine the fields of an IPv6 header in detail:

- *Version:* This 4-bit field indicates the IP version that is in use. For IPv6 packets, you will see the value set to 6.
- *Traffic Class:* This 8-bit field is used for allowing special treatment to a packet in the network based on the DSCP values assigned to an IPv6 packet. It is a combination of two fields:
  - TOS: The first 6 bits are used to set the DSCP value of a packet similar to an IPv4 packet. The DSCP value defaults to 0.
  - ECN: The last 2 bits of this field are used for congestion notification similar to how it is done in an IPv4 packet.
- *Flow Label:* The 20-bit Flow Label field is used by a source to group a set of packets belonging to the same flow. It is usually used for QoS and to ensure the packets of same flow take the same path.
- *Payload Length:* This 16-bit field represents the packet's payload. The payload may not exceed 2^{16} (65,535) bytes of data except in situations where extension headers are being used. When extension headers are used, the field value is set to 0.
- *Next Header:* This 8-bit field indicates the higher layer protocol that follows the IPv6 header.

- *Hop Limit:* The 8-bit Hop Limit field is similar to the TTL field in an IPv4 header. The value of this field represents the number of routed hops a packet can traverse before getting dropped or reaching the destination. The maximum value of this field is 255. At every routed hop, the value of the Hop Limit field is decreased by 1 and when the value reaches 0, the packet is dropped.
- *Source Address:* The 128-bit Source Address field represents the IPv6 address of the sender from which the packet originated.
- *Destination Address:* The 128-bit Destination Address field represents the IPv6 address of the packet's destination.

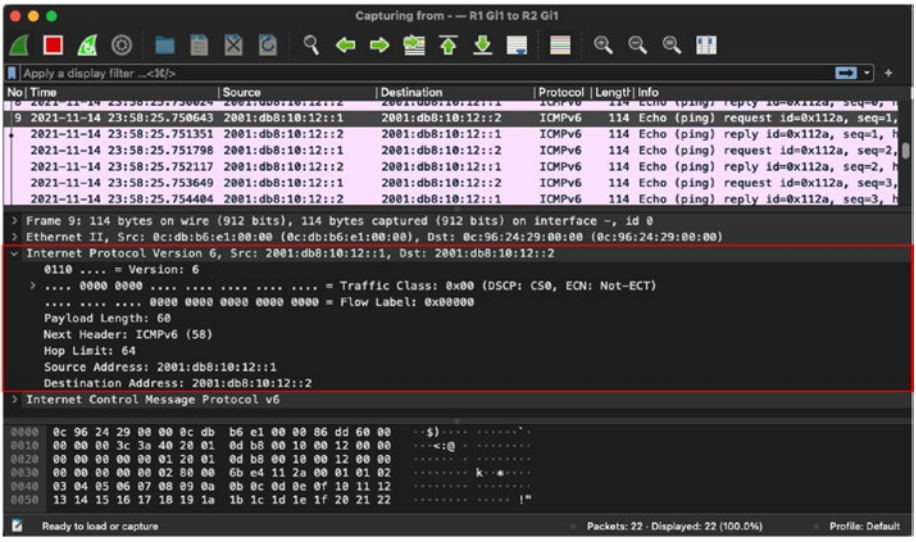


Figure 3-21. *Wireshark capture of IPv6 header*

Note There is no option for fragmentation in the IPv6 header similar to the Flags or Fragment Offset fields in the IPv4 header. If fragmentation is required on Ipv6 packets, the Extensions header is used.

IPv6 Addressing

Because IPv6 addresses support 128-bit addressing, the protocol supports undecillions (a trillion trillion trillion) addressable spaces, or simply put, 3.4×10^{38} unique IPv6 addresses can be allocated. IPv6 addresses are represented in hexadecimal format written in eight groups of 2 bytes each, with each group separated by a colon. For instance, an IPv6 address would look something like the following address, making it virtually impossible to memorize.

```
1111:2222:3333:4444:aaaa:bbbb:cccc:0abc
```

One of the best features of IPv6 address notation is that some groups of zeros can be collapsed using a double colon to form a shorter address. Note, however, that in any given address, only a single collapsible group is allowed. Consider the IPv6 address shown here:

```
1111:2222:3333:0000:aaaa:0000:cccc:0abc
```

This address can either be written as:

```
1111:2222:3333::aaaa:0000:cccc:0abc
```

or as:

```
1111:2222:3333:0000:aaaa::cccc:0abc
```

Note A deep dive on IPv6 addressing is outside the scope of this book. To learn more about IPv6 addressing, refer to RFC 4291.

There are several types of IPv6 addresses that are supported:

- *Link local address:* Link local addresses are automatically assigned to the interfaces on which IPv6 is enabled. These addresses are used to communicate with hosts on the same subnet. This address always starts with FE80.
- *Global unicast:* These addresses are public IPv6 addresses that are uniquely recognized and are routable over the Internet.
- *Unicast address:* A unicast address is used for a single host on a network.
- *Unique local:* These addresses are routable within the administrative domain.
- *Multicast address:* Multicast addresses are used to send data to multiple receivers who are subscribed to the multicast group address.
- *Anycast address:* Anycast addresses are used to send data to multiple locations using the same IPv6 address. An anycast address is allocated for a set of interfaces that typically belong to different nodes.

Following are the IPv6 address subnet ranges for different IPv6 addresses:

- *Global unicast:* 2000::/3
- *Unique local:* FC00::/7
- *Link local:* FE80::/10
- *Multicast:* FF00::/8

Extension Headers

In IPv4, a lot of capabilities were added using the Options field and similar functionalities and those capabilities had to be preserved in IPv6. Having an additional Options field added a bit of an overhead in processing IPv4 headers, though. Taking that into consideration, IPv6 Extension Headers (EH) was introduced, as defined in RFC 2460. The EH are added to the IPv6 headers as needed and the main header of 40 bytes remains as is. Figure 3-22 illustrates what an IPv6 header with EH would look like.

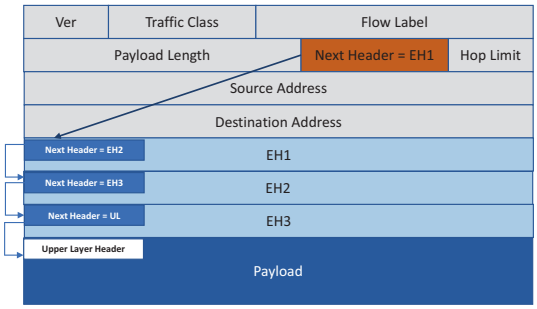


Figure 3-22. *IPv6 header with Extension Headers*

Table 3-6 lists the various Extension Headers and their mapping for next header values.

Table 3-6. *IPv6 Extension Headers and Next Header Values*

Order	Header Type	Next Header Code
1	Basic IPv6 header	-
2	Hop-by-Hop options	0
3	Destination options (with Routing options)	60
4	Routing header	43
5	Fragment header	44
6	Authentication header	51
7	Encapsulation Security Payload header	50
8	Destination options	60
9	Mobility header	135
	No next header	59
Upper layer	TCP	6
Upper layer	UDP	17
Upper layer	ICMPv6	58

ICMPv6

ICMPv6 is a very crucial protocol when it comes to the working of IPv6. Although IPv4 and IPv6 are similar in terms of their overall functionality, ICMPv6 has multiple use cases when it comes to the functioning of IPv6. ICMPv6 provides additional benefits such as these:

- Improved multicast routing
- Extensions
- Stateless Autoconfiguration (SLAAC)

The ICMPv6 header is similar an IPv4 header. It contains the Type, Code, and Checksum fields, followed by ICMPv6 options and contents that are based on type and code values.

In the previous section, you likely noticed that when we were talking about IPv6 addressing, we did not talk about broadcast. That is because there is no concept of broadcast in IPv6, as it is considered an inefficient mechanism. Because there is no broadcast, ARP cannot work for IPv6. This is where ICMPv6 comes into play. We talk about the IPv6 neighbor discovery process in the next section, but for now, let's focus on the different ICMPv6 messages and their Type and Code values. The ICMPv6 messages are divided into two categories, error messages and informational messages.

Table 3-7 displays a list of the most commonly used error messages in ICMPv6.

Table 3-7. *ICMPv6 Error Messages*

Type	Header Type	Code	Definition
1	Destination unreachable	0	No route to destination
		1	Communication with destination administratively prohibited
		2	Beyond scope of source address
		3	Address unreachable
		4	Port unreachable
		5	Source address failed ingress/egress policy
		6	Reject route to destination
		7	Error in source routing header
2	Packet too big	0	
3	Time exceeded	0	Hop limit exceeded in transit
		1	Fragment reassembly time exceeded
4	Parameter problem	0	Erroneous header field encountered
		1	Unrecognized next header type encountered
		2	Unrecognized IPv6 option encountered

Table 3-8 displays a list of most commonly used informational messages in ICMPv6.

Table 3-8. *ICMPv6 Informational Messages*

Type	Header Type	Code	Definition
128	Echo Request	0	
129	Echo Reply	0	
130	Multicast Listener Query (MLD)	0	• General query: Used to learn which multicast addresses have listeners on an attached link • Multicast-address-specific query: Used to learn if a particular multicast address has any listeners on an attached link
131	Multicast Listener Report (MLD)	0	
132	Multicast Listener Done (MLD)	0	
133	Router Solicitation (NDP)	0	
134	Router Advertisement (NDP)	0	
135	Neighbor Solicitation (NDP)	0	
136	Neighbor Advertisement (NDP)	0	
137	Redirect Message (NDP)	0	
138	Router Renumbering	0	Router Renumbering command
		1	Router Renumbering result
		255	Sequence number reset

(*continued*)

Table 3-8. (*continued*)

Type	Header Type	Code	Definition
139	ICMP Node Information Query	0	The Data field contains an IPv6 address that is the subject of this query.
		1	The Data field contains a name that is the subject of this query, or is empty, as in the case of a NOOP.
		2	The Data field contains an IPv4 address that is the subject of this query.
140	ICMP Node Information Response	0	A successful reply. The Reply Data field may or may not be empty.
		1	The responder refuses to supply the answer; the Reply Data field will be empty.
		2	The Qtype of the query is unknown to the responder. The Reply Data field will be empty.

Note There are other ICMPv6 informational messages, too. Table 3-8 does not provide an exhaustive list.

IPv6 Neighbor Discovery

To learn about the connected neighbor in IPv6, we have Neighbor Discovery Protocol (NDP). NDP uses the link local address (fe80::/64) as its source and the hop limit is set to 255. IPv6 neighbor discovery relies primarily on two functions, neighbor solicitation and neighbor advertisement.

A Neighbor Solicitation (NS) message is primarily used for three purposes:

- Determining the link-layer address of a neighbor.
- Checking the validity of an already defined address.
- Validating if an IPv6 address generated via auto-config is unique.

Let's assume we have two hosts or two devices, say A and B, connected to each other and interested in communicating with each other. When Host A wants to form an IPv6 neighborship with Host B, it will send an NS packet. An NS packet is basically a Type 135 ICMPv6 packet. The originating device sends the NS packet to every device on the network via multicast, which basically communicates to the receiver, "What is the MAC address of 2001:db8:12:2? My MAC address is 01:01:01:01:01:01." When the device that is assigned the destination IPv6 address receives the multicast NS packet, it responds back with a Neighbor Advertisement (NA) packet, which is an ICMPv6 Type 136 packet. This packet basically tells the source, "Hi. My network address is 2001:db8:12:2 and my MAC address is 02:02:02:02:02:02." This neighbor discovery process is explained visually in Figure 3-23.

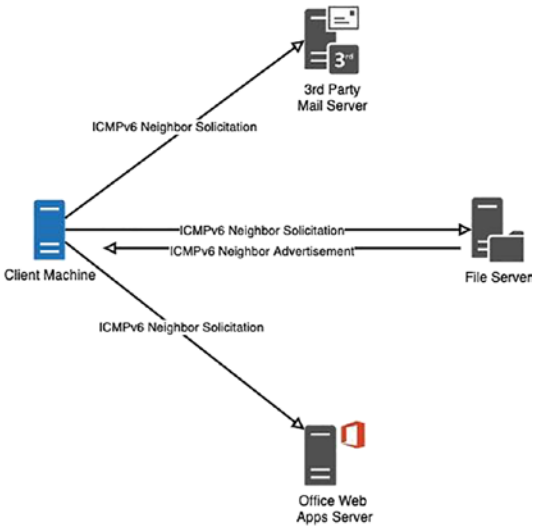


Figure 3-23. *IPv6 neighbor discovery*

Figures 3-24 and 3-25 demonstrate the NDP though the Wireshark capture. Notice that in Figure 3-24 the NS packet has generated sourcing the client machine's unicast address with the destination set to the solicited node multicast address. The solicited node multicast address is created by taking the least significant 24 bits of the unicast or anycast address and appending it to the FF02::1:FF00:0/104 address.

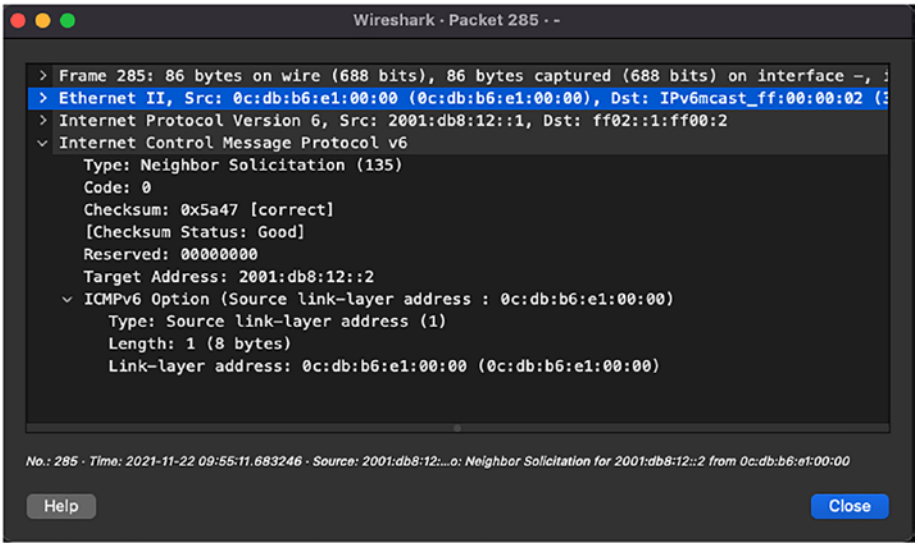


Figure 3-24. *ICMPv6 Neighbor Solicitation packet*

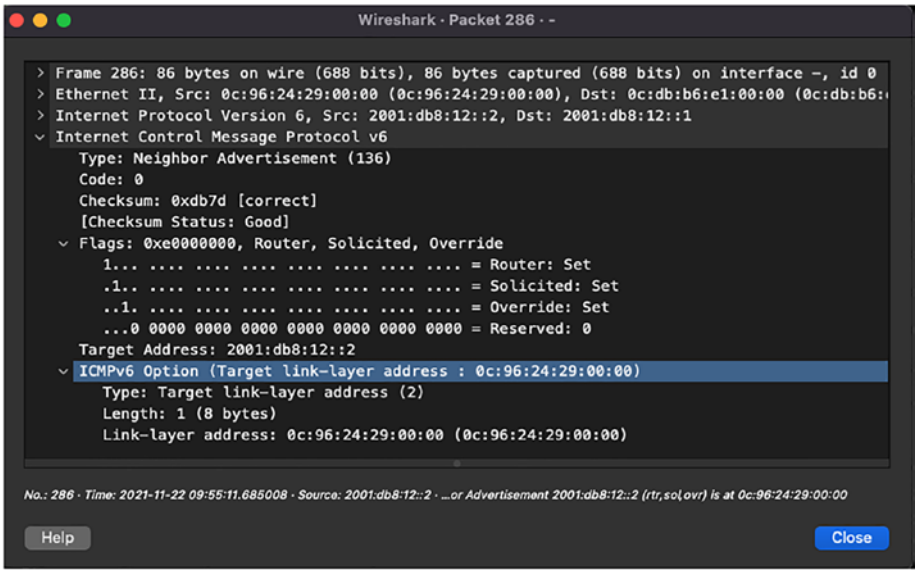


Figure 3-25. *ICMPv6 Neighbor Advertisement packet*

Apart from the NS and NA messages, other ICMPv6 messages are used as part of the NDP:

- *Router Solicitation (ICMPv6 Type 133):* The Router Solicitation (RS) messages are sent by nodes at bootup to find a router in the local segment. These messages are sent by the hosts to the All Router Multicast Address (FF02::2). On receiving this message, an IPv6 router will generate an RA message immediately rather than waiting for the next scheduled interval. Because the destination address is a multicast address, the corresponding Layer 2 address will be in the format 33:33:xx:xx:xx:xx, where xx:xx:xx:xx:xx is the last 24 bits of the destination IPv6 address.
- *Router Advertisement (ICMPv6 Type 134):* The RA messages are sent in response to the RS messages or periodically. The RA messages are sent to All Nodes Multicast Address. These messages consist of certain flags and options that contain the information that the interfaces on the links use to configure themselves. IPv6 routers send RA messages periodically at random intervals to reduce synchronization issues when there are multiple IPv6 routers on the segment.
- *Redirect (ICMPv6 Type 137):* Redirects are used by IPv6 routers to inform the hosts of a better first hop for a destination.

Analyzing QoS Markings

Almost every organization utilizes time-sensitive applications such as VoIP or streaming media, routing protocols, and so on. Because the global Internet is unpredictable, there are chances that such critical and time-sensitive applications could be dropped. The traffic for such applications should be given higher priority and treated differently in the network than the usual data traffic. To do that, the IPv4 header has the Type of Service (DSCP + ECN) field and the IPv6 header has the Traffic Class field, which allow the user to set DSCP values that will categorize different application traffic. The network devices such as routers and switches can then be configured to treat the traffic based on their DSCP values.

Although we deal with QoS settings mostly at Layer 3, there is still the possibility of frame prioritization at the Layer 2 level. This is done using Class of Service (CoS) bits in Layer 2 frames. If we talk about a regular packet, at the outermost layer we have the Layer 2 header, then the IP header, and then the data or payload. From the Layer 2 frame perspective, we usually have an 802.1Q or 802.1p frame. Figure 3-26 highlights the Tag field in the 802.1Q header, which is used for setting CoS bits.

Figure 3-26. *QoS at Layer 2*

The 802.1Q frame header has a 16-bit Tag control information field that has the following subfields:

- *Priority code point (PCP):* 3-bit
- *Drop Eligible Indicator (DEI):* 1-bit
- *VLAN Identifier (VID):* 12-bit

The PCP field refers to the IEEE 802.1p CoS and maps to the frame priority level. The values of this field are used to prioritize different classes of traffic.

When talking about QoS at Layer 3, ToS (DiffServ field) or Traffic Class is an 8-bit field, out of which the initial 6 bits make up the DSCP field and the last 2 bits are for ECN as shown in Figure 3-27.

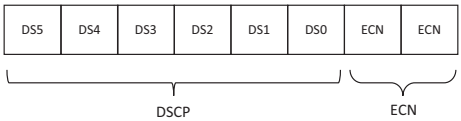


Figure 3-27. *DiffServ field*

Based on the 6-bit DSCP values, the traffic is given different treatments in the network and these values are categorized as explained here:

- *Default Forwarding (DF):* Any traffic that does not meet the criteria of any of the defined classes falls under the category of Default Forwarding. The default and recommended DSCP value of this class is 0.
- *Expedited Forwarding (EF):* RFC 3246 defines the EF per-hop behavior (PHB) for traffic that has low delay, low loss, and low jitter requirements. This class is suitable for voice, video, and real-time service traffic. The recommended DSCP value of EF is 46.
- *Assured Forwarding (AF):* RFC 2597 and RFC 3260 define the behavior for the AF class. This class assures delivery of traffic if the traffic does not exceed some subscribed rate. Within AF, four separate classes are defined and packets within each class are given drop precedence (low, medium, and high). Note that the traffic within one class has the same priority. Table 3-9 shows the different AF classes categorized based on their drop probability.

Table 3-9. *Assured Forwarding Classes Based on Drop Probability*

Drop Probability	Class 1	Class 2	Class 3	Class 4
Low	AF11 (DSCP 10)	AF21 (DSCP 18)	AF31 (DSCP 26)	AF41 (DSCP 34)
Med	AF12 (DSCP 12)	AF22 (DSCP 20)	AF32 (DSCP 28)	AF42 (DSCP 36)
High	AF13 (DSCP 14)	AF23 (DSCP 22)	AF33 (DSCP 30)	AF43 (DSCP 38)

- *Class Selector:* Before DiffServ, IP networks used the IP Precedence field in the ToS byte to prioritize the traffic to maintain backward compatibility with devices that still use IP Precedence, and the Class Selector PHB was defined. Table 3-10 lists all the IP Precedence values.

Table 3-10. *IP Precedence Values*

Value	IP Precedence Bits	IP Precedence Name
0	000	Routine
1	001	Priority
2	010	Immediate
3	011	Flash
4	100	Flash Override
5	101	Critical
6	110	Internetwork Control
7	111	Network Control

To sum up, the different DSCP values and their corresponding IP Precedence values are shown in Table 3-11.

Table 3-11. *DSCP and IP Precedence Values*

DSCP Value	Decimal Value	Meaning	IP Precedence Value
101 110	46	Expedited Forwarding (EF)	101 – Critical
000 000	0	Best Effort/Default	000 – Routine
001 010	10	AF11	001 – Priority
001 100	12	AF12	
001 110	14	AF13	
010 010	18	AF21	010 – Immediate
010 100	20	AF22	
010 110	22	AF23	
011 010	26	AF31	011 – Flash
011 100	28	AF32	
011 110	30	AF33	
100 010	34	AF41	100 – Flash Override
100 100	36	AF42	
100 110	38	AF43	
001 000	8	CS1	1
010 000	16	CS2	2
011 000	24	CS3	3
100 000	32	CS4	4
101 000	40	CS5	5
110 000	48	CS6	6
111 000	56	CS7	7

Based on Table 3-11, it can be understood that traffic such as voice, video, and network protocols are given a higher priority in the network. All the routing protocols traffic is sent with the DSCP value of CS6. To deal with all other traffic such as voice, video, or application traffic, QoS needs to be configured on a per-hop basis so that each device can treat the traffic accordingly. Figure 3-28 shows the Wireshark capture of a BGP keepalive message that is marked with a DSCP value of CS6 under the DiffServ field in the IP header.

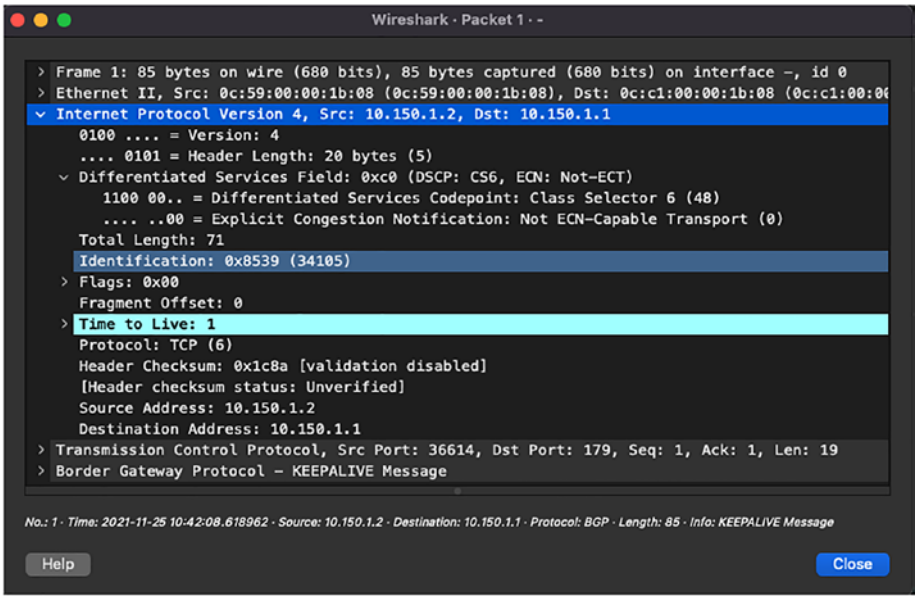


Figure 3-28. *Wireshark capture of BGP packet*

When performing network QoS testing and validation, network operators can simply use the PING tool to mimic different application traffic. Example 3-2 demonstrates how to use this tool on Cisco IOS-XE software and Mac OS to simulate traffic with different ToS and DSCP settings. Although the traffic can be initiated, it is important to note that the network devices should be configured accordingly to perform further classification of the traffic. Note that when initiating the ping on Mac OS,

use the -z option to specify the ToS value. Note that the binary for 184 is 1011 1000. This means that the packet will be sent with the DSCP value of 46 (101 110 in binary) and with the ECN field set to 0 (00 in binary).

Example 3-2. Using PING for Traffic Simulation with DSCP Settings

```
! Ping on Cisco IOS-XE Software
R1#ping
Protocol [ip]:
Target IP address: 2.2.2.2
Repeat count [5]: 100
Datagram size [100]: 1400
Timeout in seconds [2]:
Extended commands [n]: y
Ingress ping [n]:
Source address or interface: 1.1.1.1
DSCP Value [0]: 46
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0x0000ABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 100, 1400-byte ICMP Echos to 2.2.2.2, timeout is 2
seconds:
Packet sent with a source address of 1.1.1.1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (100/100), round-trip min/avg/max =
1/1/46 ms
! Ping on MAC OS with DSCP value 46
```

```
vinit@Hackers-Box ~ % ping -z 184 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=64 time=125.217 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=1.200 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=2.703 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=64 time=49.220 ms
64 bytes from 192.168.0.1: icmp_seq=4 ttl=64 time=1.386 ms
64 bytes from 192.168.0.1: icmp_seq=5 ttl=64 time=1.347 ms
```

Note ECN bits in the DiffServ field will be covered in the next chapter.

Summary

By now, you should understand the Layer 2 and Layer 3 concepts as well as have a solid foundation about the different fields in Ethernet, IPv4, and IPv6 headers. In this chapter, we covered in detail the Layer 2 header, specifically the Ethernet header, and learned about various EtherTypes. We also learned about the IPv4 header, including how the packets get encapsulated inside the IP header and uses of various fields in the IP header. We also covered the ICMP header and how it can be used for troubleshooting purposes, and how ICMP messages can be used to notify the network about incorrect network MTU settings.

We then moved on to IPv6 headers, which helped network operators transition from 32-bit addressing to 128-bit addressing. We discovered some of the benefits of IPv6 over IPv4 headers and how they reduce the need for having broadcast packets by performing neighbor discovery using ICMPv6 headers. We learned that in IPv6, NDP leverages different ICMPv6 messages such as Router Solicitation, Router Advertisement, Neighbor Solicitation, Neighbor Advertisement, and Redirect message.

We also learned that the ICMPv6 messages are sent to different IPv6 multicast addresses. Finally, we ended this chapter learning about how QoS can be used in the network and how the DSCP values can be used to treat each type of application traffic differently.

Reference in This Chapter

- RFC 1918: Address Allocation for Private Internets, by Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. https://datatracker.ietf.org/doc/html/rfc1918

CHAPTER 4

Analyzing Layer 4 Traffic

This chapter covers the following topics:

- Understanding the TCP/IP model
- Transmission Control Protocol
- User Datagram Protocol

Understanding the TCP/IP Model

We have already covered the OSI model, in which we learned about Layer 2 frames and Layer 3 packets and their importance when exchanging packets between two endpoints. This chapter focuses on the Transport layer (Layer 4) of the OSI model, which is responsible for transporting the data between the source and destination either via a connection-oriented or a connectionless mechanism. There are various Transport layer protocols that are used to transmit the data, including these:

- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)
- Stream Control Transmission Protocol (SCTP)
- Reliable User Datagram Protocol (RUDP)

V. Jain, *Wireshark Fundamentals*, https://doi.org/10.1007/978-1-4842-8002-7_4

In this chapter, we focus primarily on the TCP and UDP modes of transmission.

In the 1970s, two Defense Advanced Research Projects Agency (DARPA) scientists, Vint Cerf and Bob Kahn, often known as the fathers of the Internet, started researching reliable data communications across packet radio networks. From the lessons learned from the Networking Control Protocol (NCP), a set of protocols forming a part of Point-to-Point Protocol (PPP), Cerf and Kahn created the Transmission Control Program. Transmission Control Program was a huge success, and in 1974 it was initially standardized in RFC 675, Specification of Internet Transmission Control Program.

Initially, the Transmission Control Program managed both the routing and datagram transmission, but over time, collaborators suggested dividing the functionality into layers. In 1978, the Transmission Control Program was split into two distinct Protocols, the Internet Protocol (IP) and the Transmission Control Protocol (TCP). Both the protocols combined to form the Internet Protocol suite, commonly known as TCP/IP. The TCP/IP model only has four layers:

- *Application layer:* This layer allows for process-to-process communication on the same host or different hosts. This layer leverages the lower layer protocols to transmit the information. The Application layer introduces different communication models such as the client/server model or peer-to-peer networking model. Some of the examples of aHTTP, FTP, and SSH.
- *Transport layer:* This layer takes care of performing host-to-host communication that is either in a local LAN segment or remote network segments separated by routers. Two of the primary protocols in the Transport layer are TCP and UDP.

- *Internet layer:* This layer defines the addressing and routing structures used by the TCP/IP protocols. IP defines the addressing that will be used by the hosts and network elements in the same or different segments and provides a function for hop-by-hop routing by sending datagrams to the next hop that holds the information to the next network segment.
- *Link layer:* This layer is a combination of the Data Link layer and Physical layer of the OSI model. This layer provides information about hardware addresses (MAC address) and ensures physical transmission of data.

When distributed applications or client/server applications that are separated across network segments communicate using a router, they leverage the TCP/IP model to establish the communication and exchange information. As mentioned before, the Application layer of the TCP/IP model only focuses on process-to-process communication; it leverages the underlying layers to transmit the data. Figure 4-1 illustrates how a client/server application communicates using the TCP/IP model. When the client wants to send a request to the server, it creates a packet data unit (PDU) for the data that it wants to send to the remote server. The first level of encapsulation is provided by the Transport layer, in which it is decided based on the application requirements if the communication is to be established using a connectionless architecture (via UDP) or using a connection-oriented architecture (via TCP). The packet is then encapsulated with an IP header and then with the protocols at the Link layer. Once the final encapsulation is completed, the packet is sent across the network segments, processed, and routed accordingly toward the destination host, where the decapsulation process starts from Link layer all the way to the Transport layer, after which the final PDU is received by the remote server and processed accordingly.

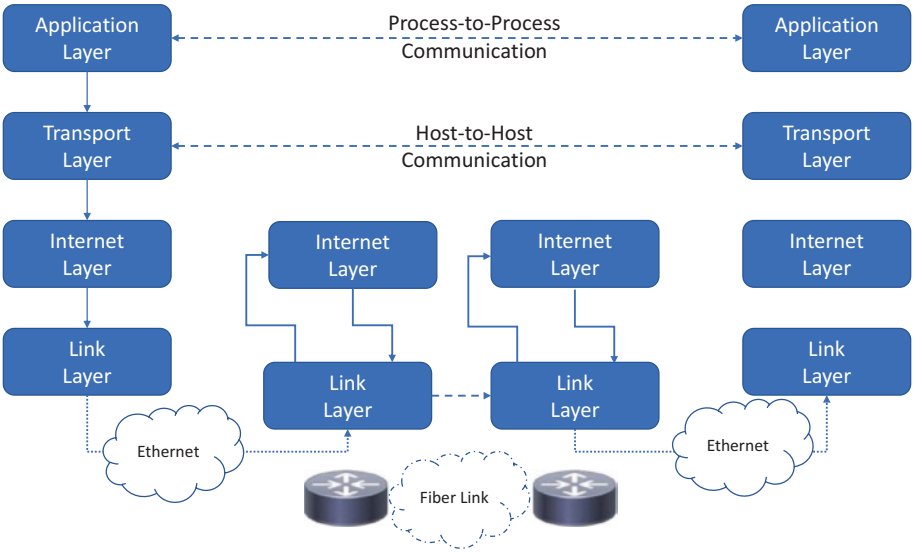


Figure 4-1. *Data flow across the TCP/IP model*

The TCP/IP model is so widely used that it can be safely said that the Internet today depends on it. The TCP/IP model contains a suite of protocols that allows for host-to-host communication across multiple network segments. Figure 4-2 displays the TCP/IP models and the protocols being used at each layer.

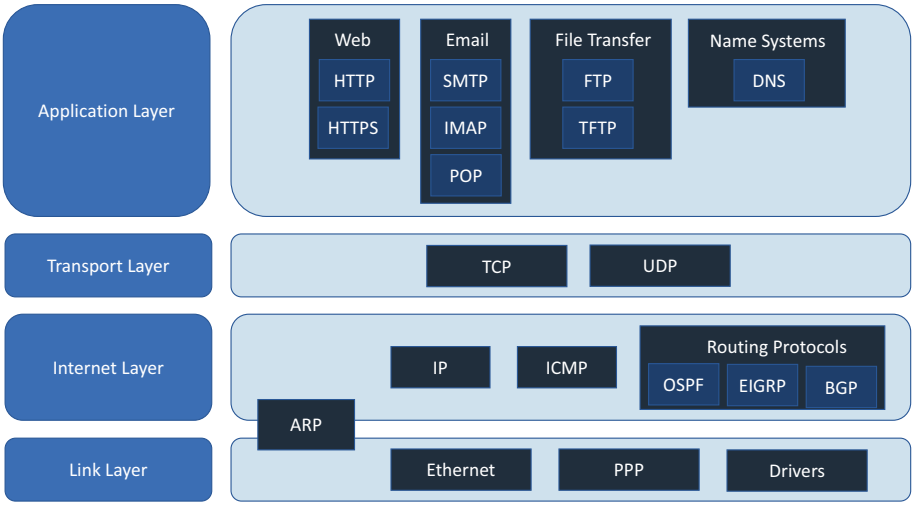


Figure 4-2. *TCP/IP protocol suite*

If we take a closer look at the TCP/IP model, it is not very different from the OSI model, but it was developed to solve a different problem than the OSI model. The Application layer, Presentation layer, and Session layer of the OSI model are categorized all under the Application layer in the TCP/IP model. The Transport layer and Network layer remain the same in the TCP/IP model. The Data Link layer and Physical layer of the OSI model are categorized under the Link layer in the TCP/IP model. Apart from the number of layers and layer mapping, there are some other key differences between the two models, listed in Table 4-1.

Table 4-1. *OSI Model vs. TCP/IP Model*

OSI Model	TCP/IP Model
Transport layer is only connection-oriented.	Transport layer is both connection-oriented and connectionless.
Allows users to standardize router, switch, motherboard, and other hardware.	Focuses only on establishing connection between different types of computers.
Provides clear distinction between interfaces, services, and protocols.	Doesn't provide clear distinction points.

Problem of Ownership

Today, almost every IT-enabled organization depends on mission-critical applications to successfully run its business. Sometimes these applications are an important aspect of a company's revenue generation model (e.g., an ecommerce portal). When those mission-critical applications stop working and the software developers are sure that it is not because of their code, they escalate the problem to network engineers or network administrators. The software developers usually own the Application layer, Presentation layer, and Session layer of the OSI model or just the Application layer of the TCP/IP model.

In 90 percent of the cases, if not more, network engineers verify the routing and reachability, which falls under the Link layer and Internet layer of the TCP/IP model (Physical layer, Data Link layer, and Network layer of the OSI model) and escalate it back to the application team, saying it is not their problem. This is where the finger-pointing game starts between the application developers and network engineers. If we take a close look at both the OSI model and the TCP/IP model, you will realize that both the application developers and the network engineers do not demonstrate

ownership for an important layer—the Transport layer. Nobody wants to go the extra mile and check the information presented at the Transport layer. Instead of posing this as a challenge, we should be seeing this as an opportunity. Knowing how the data are transmitted and how host-to-host communication will happen can quickly help isolate the problem and help both the application developers and network engineers solve the problem quickly.

Transmission Control Protocol

When the initial research was being done by DARPA in 1973, the focus was on developing a protocol that would ensure secure transmission between two hosts while maintaining the integrity of the data, regardless of the amount of data being sent. DARPA and the University of Southern California collaborated and standardized the protocol specification of TCP, a protocol that provided a connection-oriented data transmission mechanism while ensuring data integrity, in RFC 793. RFC 793 was later updated by RFC 1122, RFC 3168, RFC 6093, and RFC 6528.

The current version of TCP allows two nodes or endpoints to establish a connection that enables two-way transmission of data; that is, a device can send and receive data at the same time. Each connection in TCP works in a client/server model, irrespective of which node assumes the role of the server or client, and each endpoint connection is uniquely identified using an ordered pair of IP address and port number. This ordered pair is known as a tuple or a socket. Thus, a TCP connection is often referred to as a socket connection. Now, before moving onto understanding how a TCP connection is established, let's take a closer look at the TCP header, displayed in Figure 4-3.

A TCP header is a 20-byte header that consists of the following fields:

- *Source port (16-bit):* Specifies the port number of the sender.
- *Destination port (16-bit):* Specifies the port number of the receiver.
- *Sequence number (32-bit):* Used to keep track of the data (in bytes) sent out by the host during a TCP session. During a new TCP connection, the initial sequence number sent is a random 32-bit value. The receiver will use the sequence number and reply with an acknowledgment. When it comes to troubleshooting TCP issues, protocol analyzers often use a relative sequence number of 0 as it is easier to remember than some high-value random number. The sequence number is also used for validating the segments after transmission.
- *Acknowledgment number (32-bit):* Used to keep track of every byte received by the receiver. An acknowledgment is sent in response to every packet that is received by the host.
- *Offset (4-bit):* Specifies the length of the TCP header, which allows us to know where the actual data begins.
- *Reserved (6-bit):* Reserved for further use as per RFC 793.

- *Flags (6-bit):* Enables various TCP actions for data processing and communication. The TCP software will perform specific actions when one or more flags are set in the TCP header. The following are the various flags that are set in TCP:
    - Urgent Pointer (URG)
    - Acknowledgment (ACK)
    - Push (PSH)
    - Reset (RST)
    - Synchronization (SYN)
    - Finish (FIN)
- *Window size (16-bit):* Specifies the number of bytes the receiver is willing to receive. Using this field, the receiver tells the sender the amount of data that it is willing to receive.
- *Checksum (16-bit):* Used to verify if the TCP header is okay or not. Using this field, TCP is able to reliably detect any transmission issues.
- *Urgent pointer (16-bit):* This field is used to indicate how many bytes of packet data starting from the first byte are considered to be the urgent data by the sender. This field is only used when the URG flag is set under the Flags field.

- *Options (0–320 bits):* The TCP Options field is at the end of the header and is always specified in multiples of 8 bits. If any of the bits are not filled, they are padded with zeros. This field is used to include various TCP functions that do not belong to the general TCP header. The following is a list of various TCP functions available as part of the TCP Options field:
  - Maximum Segment Size (MSS)
  - Window Scaling
  - Selective Acknowledgments (SACK)
  - Timestamps
  - Nop

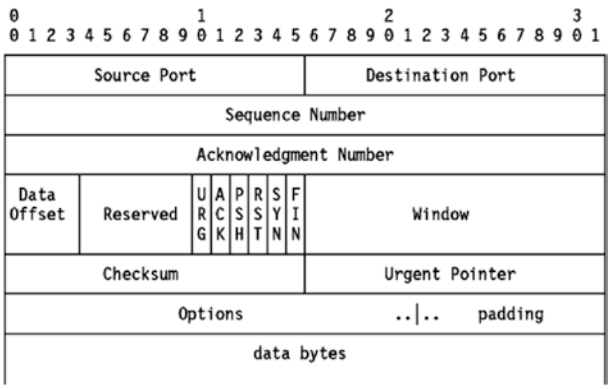


Figure 4-3. *TCP header*

Figure 4-4 shows the Wireshark capture of the first packet of an SSH session. Notice that this packet displays both the raw sequence number, which is a randomly generated number, and a relative sequence number that can be used for troubleshooting purposes. This packet also has multiple TCP options such as MSS and Timestamps as part of the TCP Options field.

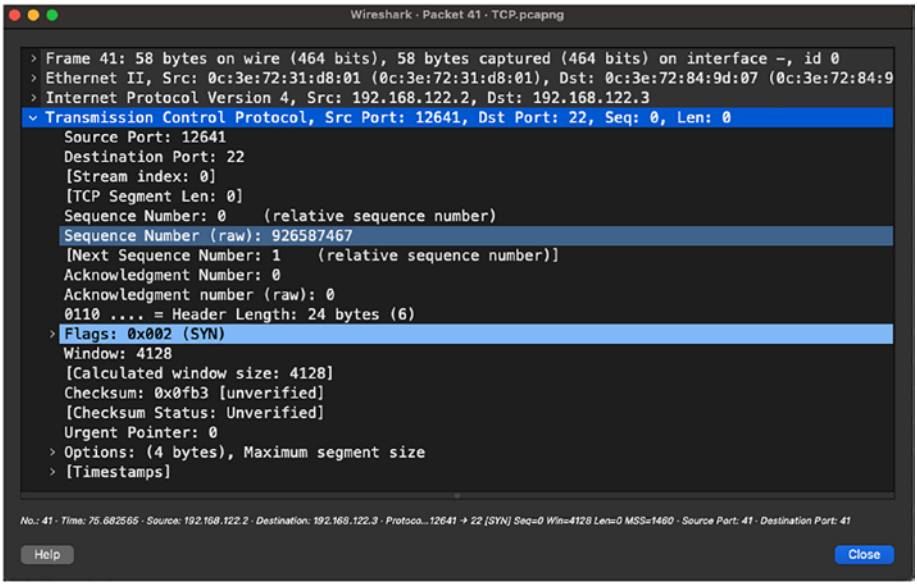


Figure 4-4. *Wireshark capture of TCP packet*

When talking about port numbers in the TCP header, there are a maximum of 65,535 port numbers that are allowed on a system. There are some well-known port numbers, too, that we use knowingly or unknowingly on a daily basis through various applications:

- HTTP: Port 80
- HTTPS: Port 443
- Telnet: Port 23
- SSH: Port 22
- FTP: Port 21
- DNS: Port 53
- IMAP: Port 143
- POP3: Port 110

Port numbers can be categorized into three types:

- *Well-known ports:* These port numbers range from 0 to 1023.
- *Registered ports:* These port numbers range from 1024 to 49151. They are not assigned or controlled but can be registered to avoid any duplication.
- *Dynamic ports:* These ports range from 49152 to 65535 and cannot be assigned, controlled, or registered.

When troubleshooting, TCP-related issues such as HTTP/TCP-based IP Service Level Agreement (SLA) probes that are deployed on network devices might stop working. As network engineer you might focus on validating the configuration or checking why the remote end has stopped responding, but it is equally important to keep an eye on the ports that are locally open on the box. It could be that even though the TCP session gets established, the connection does not terminate from time to time, and the device might run out of available ports to establish further TCP sessions.

TCP Flags

In the previous section we learned about the six flags in the TCP header. Each flag plays an important role during various phases of a TCP session. Figure 4-5 shows all the different flags in the TCP header seen in a Wireshark capture.

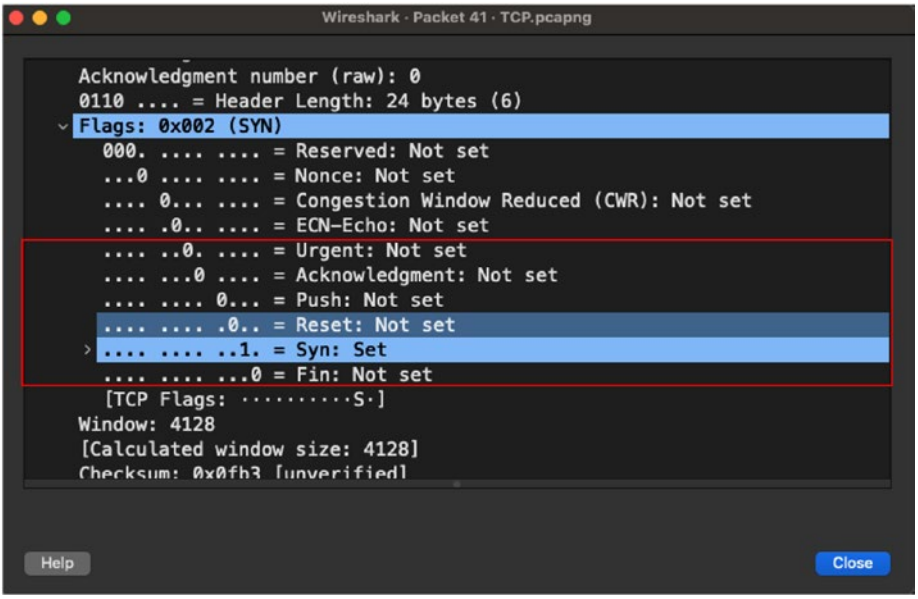


Figure 4-5. *TCP flags in Wireshark capture*

Each TCP flag is set to perform the following actions:

- *URG:* The Urgent flag is set to signal the TCP application that the payload data must be processed urgently up to the set pointer in the Urgent Pointer field. Note that the Urgent Pointer field is relevant only when the URG flag is set in the TCP header.
- *ACK:* The Acknowledgment flag is set in combination with the Acknowledgment Number field. This flag indicates the acknowledgment by the receiver that it has received the TCP packets that were previously sent.
- *PSH:* The Transport layer, by default, stores the application data in a buffer for some time so that it can transmit data equal to the MSS size to ensure faster convergence and better performance in the network for

TCP applications. Such behavior is not desirable, though, for certain applications, such as chat applications. Similar issues apply on the receiving end as well. The PSH flag in the TCP header solves this problem by telling the TCP software to immediately send the payload to the Network layer as soon as it receives the payload from the Application layer. In simple words, it tells the receiver and sender to immediately process the packets instead of buffering them.

- *RST:* If the TCP software identifies an error during transmission, it sends an RST flag to reset the connection.
- *SYN:* The SYN flag is the first step to initiate a TCP connection via the three-way handshake process.
- *FIN:* – The FIN flag signals the receiver that the sender is ending the transmission.

TCP Three-Way Handshake

The TCP three-way handshake is a three-step process that is required to establish a secure and reliable TCP connection between a client and a server.

1. *SYN:* In the first step, the client initiates a TCP connection toward the remote server. When it does that, the SYN flag is set to 1 in the TCP header and a random sequence number is used for this TCP connection. In this case, it is 926587467, as shown in Figure 4-6. Because this is the first packet, the ACK flag is set to 0. There are other fields that are also set in the TCP header such as Window size and MSS TCP options.

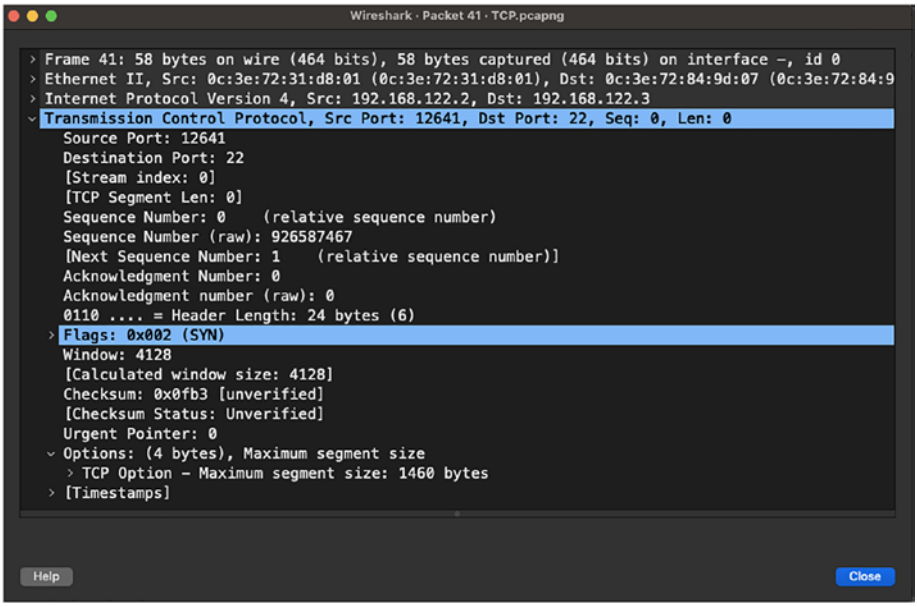


Figure 4-6. *TCP SYN Wireshark capture*

2. *SYN-ACK:* When the server receives the SYN packet for the TCP connection, it responds back with an acknowledgment by setting the ACK flag bit to 1. Also, when the ACK flag is set, the Acknowledgment Number field is set to the value of one more than the received SYN packet. So, in this case, the Acknowledgment number field will have the value 926587468. Also, because TCP allows for two-way communication, the server also sets the SYN flag to 1 and sets a random sequence number in the TCP header. Note that the sequence number used in the SYN-ACK packet will be different than the one received from the client. In this case, it is set to 4227540456. Figure 4-7 displays the TCP SYN-ACK packet from server to client.

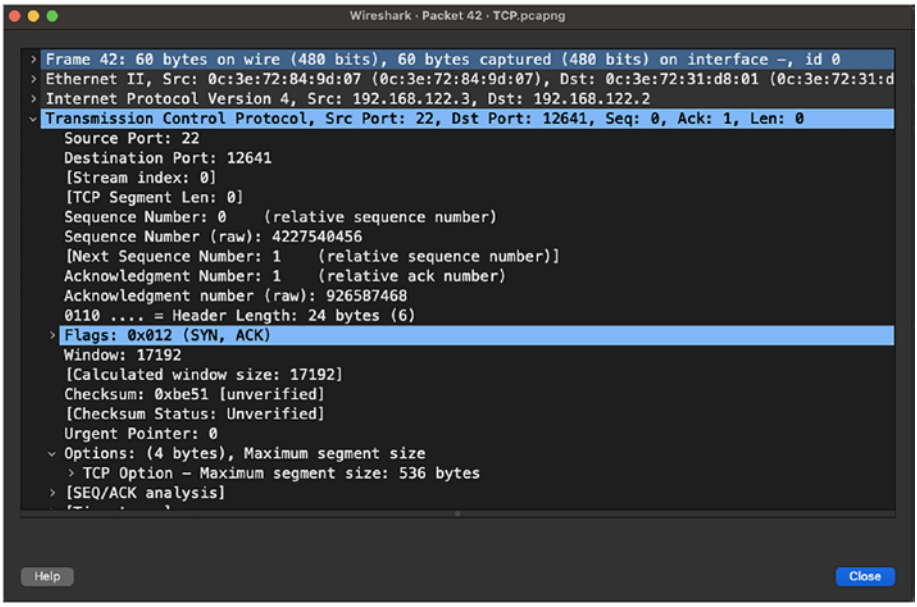


Figure 4-7. *TCP SYN-ACK Wireshark capture*

3. *ACK:* On receiving the SYN from the server, the client now must respond back with an acknowledgment. For that, the client sends another TCP packet to the server with the ACK flag set, and the Acknowledgment Number field value set to the value of the sequence number plus 1. In this packet the SYN flag is set to 0. Figure 4-8 displays the ACK from the client to the server with the Acknowledgment Number field set to 4227540457. Note that after the ACK is received by the server, the minimum of the client or server's MSS value is taken into consideration for data transmission.

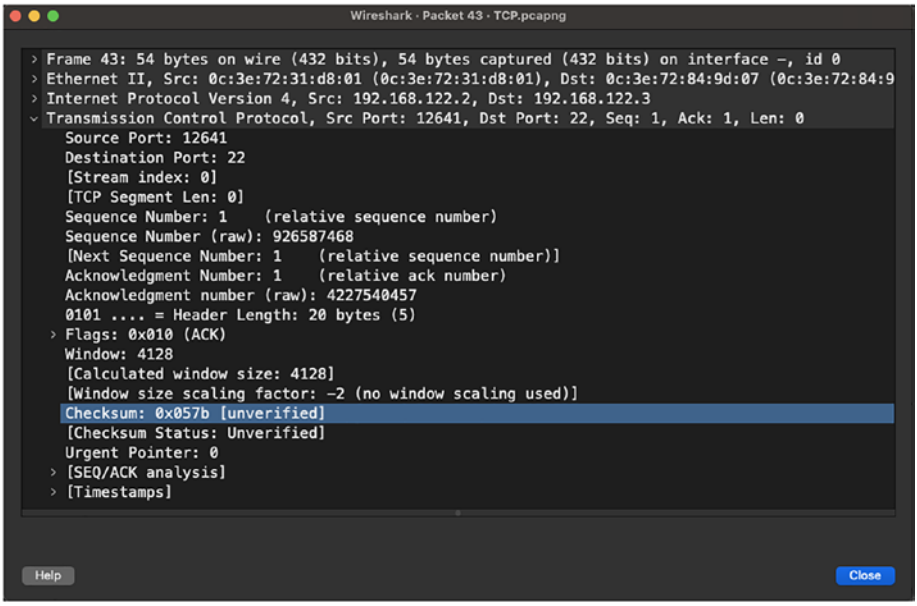


Figure 4-8. *TCP ACK Wireshark capture*

Because we are looking at an example of an SSH session between the server and client, after a TCP session has been established, all the SSH protocol exchanges are performed. If you look at the packets that are exchanged by the SSH protocol, you will notice that they mostly have the PSH flag set, which indicates that the SSH protocol is telling the TCP software not to buffer the data and transmit the packet to the remote end. Figure 4-9 displays the Wireshark capture of an SSH control packet with the PSH flag set in the TCP header.

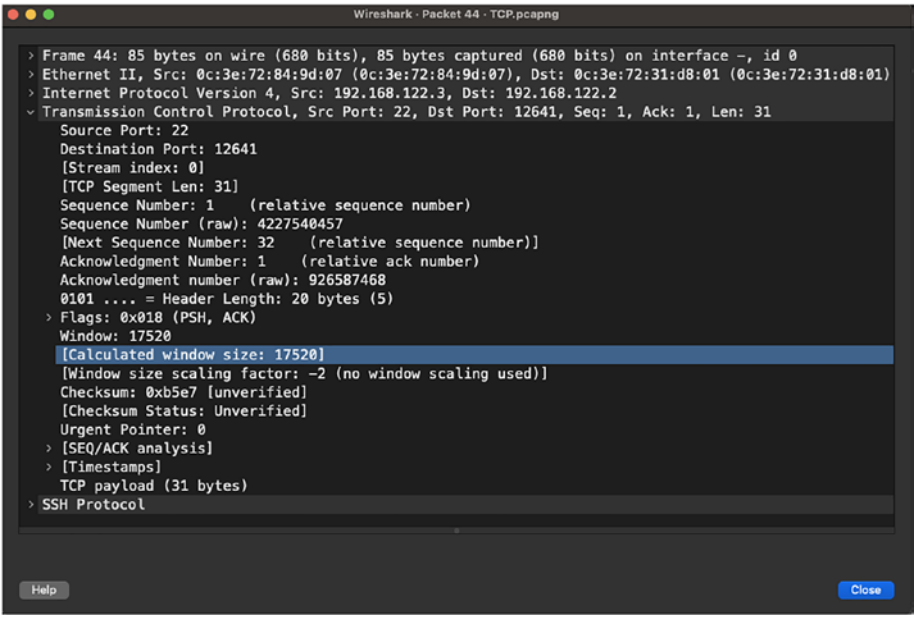


Figure 4-9. *SSH packet with PSH flag*

Like the connection initiation process, there is a three-way handshake process for connection termination. The client and the server exchange the following TCP packets when they wish to close the connection:

- *FIN:* When the client wishes to terminate the connection, it sends a TCP packet with the FIN flag set to 1 and sends it with a random sequence number. Note that at this point, the SYN flag will be set to 0 and the ACK is set to 0. If the client is supposed to send an acknowledgment to the server for the previously received TCP packet, the ACK flag can be set to 1 along with the acknowledgment number, but this ACK has no relation to the FIN flag that is set on the packet. TCP does this to reduce the number of packets being exchanged. Figure 4-10 shows the Wireshark

capture of SSH connection being terminated when an '*exit*' command is typed on the terminal by the client. Figure 4-10 shows the Wireshark capture of TCP FIN packet when the client wishes to terminate the connection. Notice that in this packet, there is an ACK flag set as well, but this acknowledgment is for another sequence number that the client received.

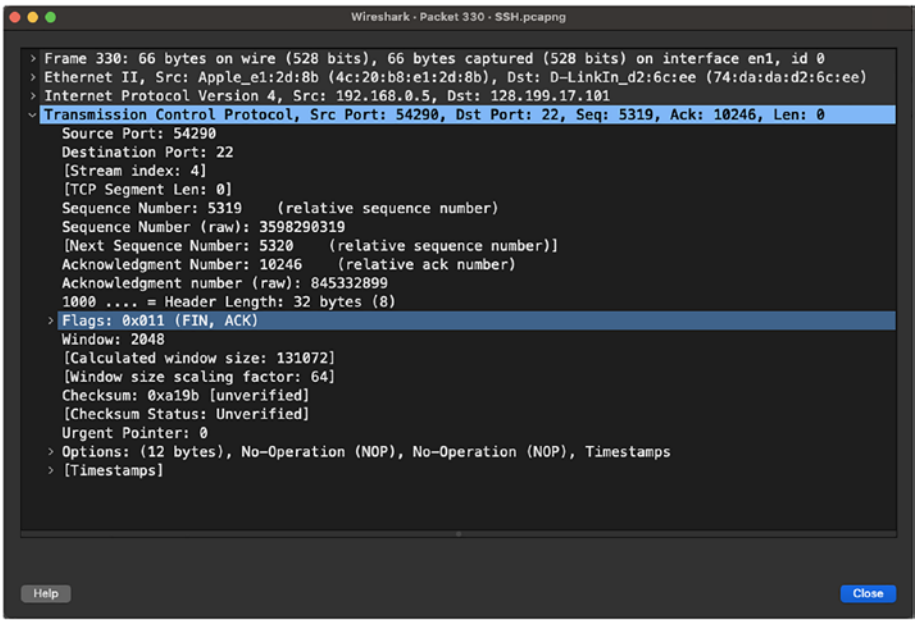


Figure 4-10. *TCP FIN Wireshark capture*

- *FIN-ACK:* On receiving the client's termination request with TCP FIN, the server acknowledges the request by replying to the client with ACK. The server also sets the FIN flag to 1 and sends it to the client with a random sequence number different than that of the received FIN. Once this step is completed, the connection is terminated from the client to the server

side. Figure 4-11 shows the Wireshark capture of a TCP FIN-ACK packet sent by the server toward the client in response to the FIN request received from the client.

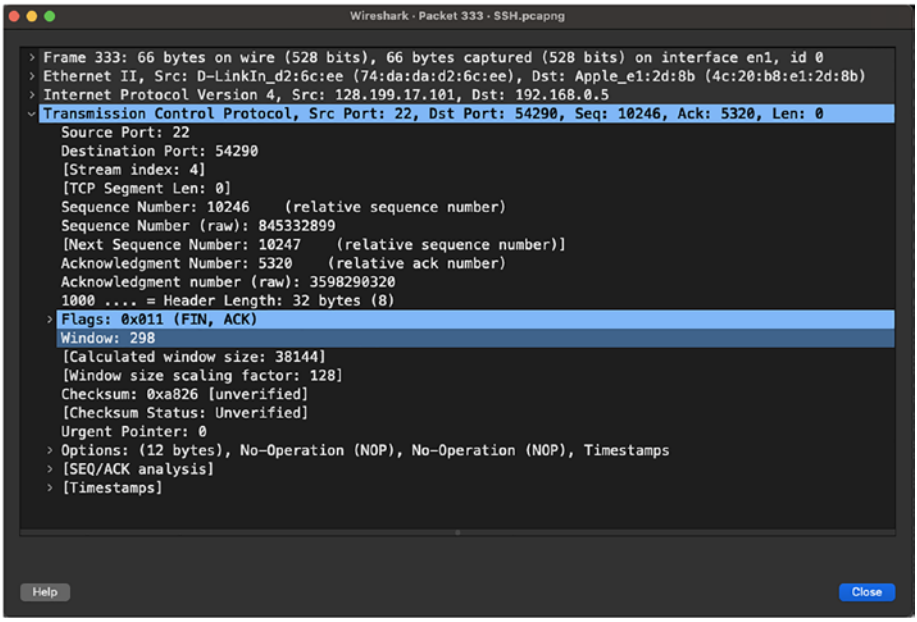


Figure 4-11. *TCP FIN-ACK Wireshark capture*

- *ACK:* This is the last step where the client sends the acknowledgment back to the server for the received FIN from the server. It sets the ACK flag to 1 and sets the Acknowledgment Number value to the Sequence Number of FIN plus 1. After this step is completed, the connection is terminated from the server to the client side. Figure 4-12 displays the Wireshark capture of the TCP ACK packet sent by the client to the server in response to the FIN packet it received from the server in the previous step. Note that the Sequence Number and Acknowledgment Number fields work in the same manner as they did during the session initiation process.

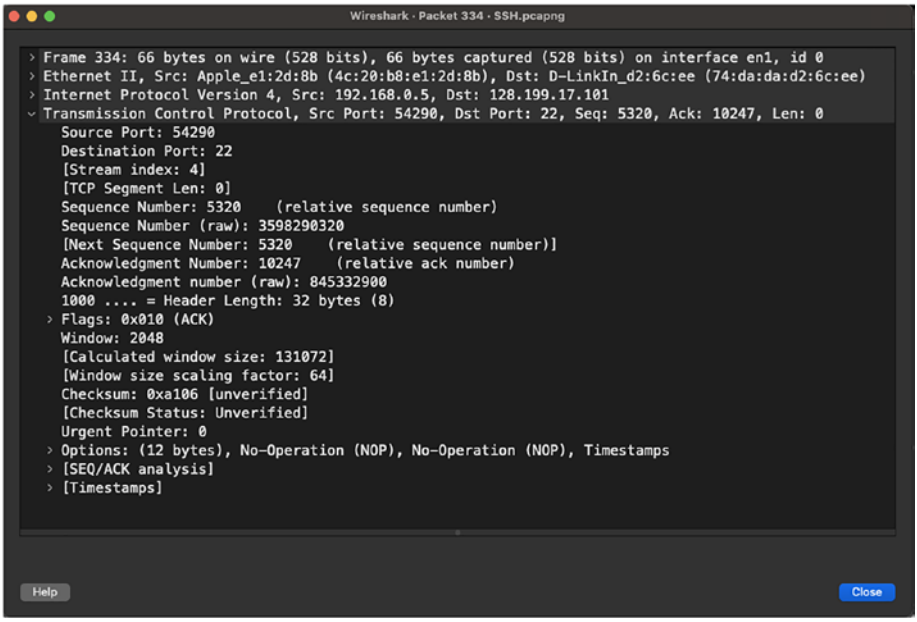


Figure 4-12. *TCP ACK Wireshark capture*

Every TCP connection goes through different connection states during its session lifetime. The following are the possible TCP connection states:

- *LISTEN:* A TCP application is awaiting an inbound connection request.
- *SYN-SENT:* A connection request has been sent but no acknowledgment has been received from the remote end.
- *SYN-RECEIVED:* A connection request has been received and an acknowledgment has been sent to the remote host, but the host is awaiting an acknowledgment of the connection request sent out as a response to the original connection request.

- *ESTABLISHED:* All SYN and ACK have been received and the connection has now been established. Both end hosts can start sharing the data.
- *FIN_WAIT_1:* A session or connection termination request has been sent, but no acknowledgment has been received.
- *FIN_WAIT_2:* An acknowledgment has been received from the remote host, but no corresponding termination request has been received from the remote host.
- *CLOSING:* A session termination request has been sent and a corresponding session termination request has been received and acknowledged but no acknowledgment has been received from the remote host for the original session termination request.
- *CLOSE_WAIT:* A session termination request was received and acknowledged but no corresponding session termination request has been sent out yet.
- *TIME_WAIT:* The host waits for a reasonable amount of time to ensure the remote host receives the final acknowledgment of a session termination request.
- *LAST_ACK:* Host awaits a final acknowledgment after sending an end of connection message in response to having received a session termination request.

At times it gets hard to remember the state transitions and the corresponding TCP flags set during those state transitions. To remember this, you can simply follow the TCP finite state machine as shown in Figure 4-13.

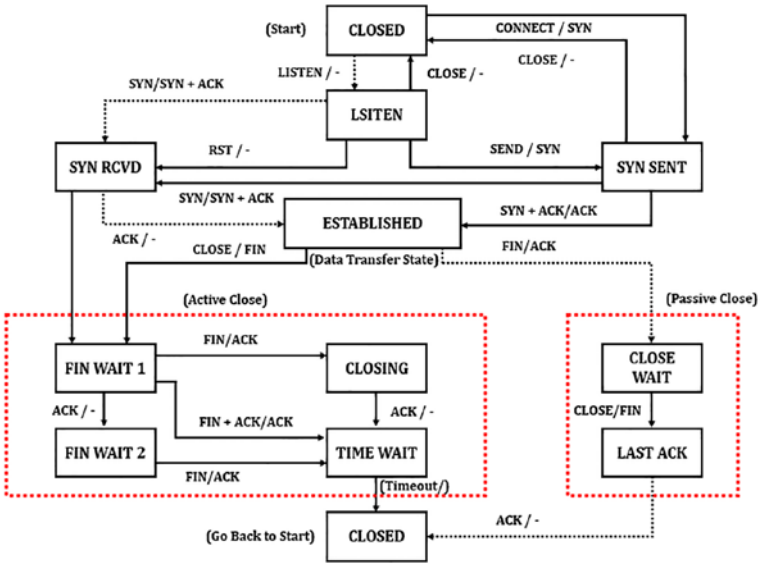


Figure 4-13. *TCP finite state machine*

Port Scanning

Many network and security analysts perform port scanning to find out about network and host vulnerabilities and services running on the network that can be exploited. Port scanning is a technique of determining which ports in a network or host are open to send or receive traffic. An open port indicates that a service such as HTTP/HTTPS or FTP is offered on the destination network or host. If attackers know what services are offered, they might be able to use other tools to identify security vulnerabilities to exploit those services.

NMAP is a freely available scanner that runs on the UNIX OS and has options for various port scanning techniques. It also has options to detect any scans that might be running on the network. Some of the port scanning techniques are listed here:

- *Connect requests:* In this technique an active connection is attempted using the three-way handshake. If the port is open, the three-way handshake is completed, and the scanner gracefully closes the connection by sending an active close request. If the port is closed, the destination responds back with an RST flag set. Note that this is not a safe scanning method, as these connection attempts are logged on the target host.
- *Half-open scan:* In this technique, the three-way handshake is not completed and thus the name half-open scan. A SYN is sent by the scanner and it waits for a response. If the target port is open, it returns a SYN-ACK and the connection will be immediately torn down by the scanning host because it did not issue the connection request. Because the handshake never completed, the target host might not log these TCP SYN packet scans.
- *Non-SYN-ACK-RST scans:* As per RFC 793, segments containing an RST flag are always discarded and segments containing an ACK always generate an RST flag. So, non-SYN packets that do not contain an RST or ACK could be used for port scanning. Note that this method of port scanning is only useful if the target host or network follows the RFC specifications. OSs that do not follow the RFC send RSTs from both open as well as closed ports, thus making it difficult for scanners to return accurate results.

As part of network security best practices, it is equally important to detect any impossible packet types that might have the following TCP flag combinations:

- SYN RST
- SYN FIN
- RST FIN
- FIN
- No flags

Network operators can perform filtering of various types of flags in Wireshark using the following filters:

- SYN Flag set: `tcp && tcp.flags == 0x02`
- ACK flag set: `tcp && tcp.flags == 0x10`
- RST flag set: `tcp && tcp.flags == 0x04`
- FIN flag set: `tcp && tcp.flags == 0x01`
- No flags set: `tcp && tcp.flags == 0x3f`

Investigating Packet Loss

Packet loss in a network can happen for two main reasons:

- Link errors/Layer 2 errors
- Network congestion

Most of the time, once a network is set up, it runs smoothly. It could demonstrate transient or complete packet loss only when the hardware fails or the link has issues. Detecting hardware failures is not very complex, as multiple links and protocols running on the network hardware or host will start showing symptoms of hardware failure and can easily be fixed by replacing the complete hardware or a particular part that is causing

the symptoms. When it comes to link issues, there could be several things to troubleshoot, some within our control and some outside. With a link, the issue could be with the unidirectional failures, Small Form-Factor Pluggables (SFPs), fiber or Ethernet cables, duplex settings, telco provider in the middle, and so on. The challenge with link issues is that even though the link might have errors, it will still forward some traffic and drop the remaining traffic, so network operators might not even know unless there is a notification of an event or a complaint from an end customer. With link issues, the data transmitted may also get corrupted and get eventually dropped. In most cases, an error counter on the network or host interface will increment to indicate an issue with the link, which then helps to identify and resolve the problem.

Traffic congestion, on the other hand, can cause a great deal of service disruption and is seen especially when transitioning between link speeds within the network (from 10 Gbps to 1 Gbps). If the higher speed link sends traffic at a rate the egress interface might not be able to keep up with, then it will start dropping the packets. In such cases, with TCP, the sender determines that the loss occurred in transit and will retransmit the packets. This scenario is also known as discards. Because TCP is a reliable connection-oriented protocol, it provides a mechanism to track data that have been sent and receive an acknowledgment of what has been received. If for any given packet the mapping ACK is not received, the TCP software resends the data assuming the packet has gone missing and ensuring reliable transmission of data. You might wonder why, after so much progress and innovation in the field of networking and development of 100 Gbps fiber links, we still have to deal with issues such as network congestion.

TCP Retransmission

As we already know, for every byte of data sent across a TCP connection, there is an associated sequence number. When a sender sends a TCP segment, it starts a retransmission timer of variable length. Let's assume

that the TCP segment gets lost in transit before reaching the receiver. Due to the packet being lost in transit, the receiver never sends the ACK back to the sender. After the retransmission timer expires, the sender assumes that the segment has been lost and it retransmits the data again to the receiver. Figure 4-14 demonstrates the segment loss and data retransmission between server and client. So, if a Wireshark capture shows a lot of retransmitted TCP segments, it simply means that there is packet loss in the network.

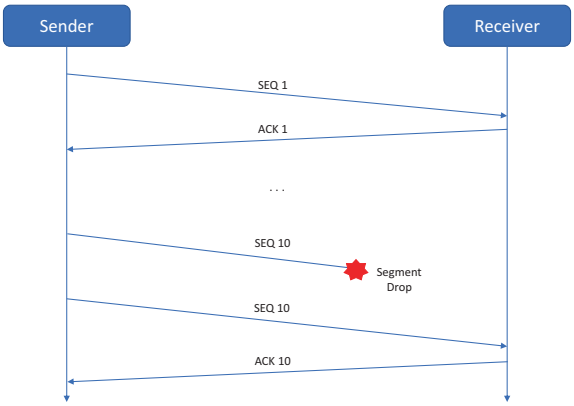


Figure 4-14. *TCP retransmission*

To analyze retransmissions in a network, network operators might have to place multiple taps in the network. For instance, examine the simple topology shown in Figure 4-15. In this topology host H1 (IP 10.1.2.1) sitting behind router R1 is trying to send traffic to host H3 (IP 3.3.3.3), which is sitting behind router R3.

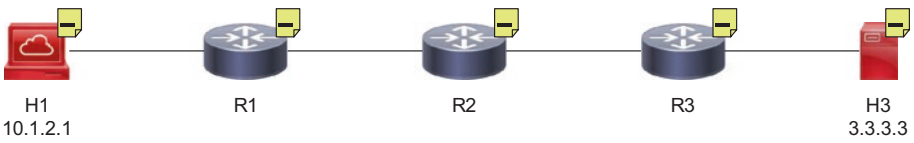


Figure 4-15. *Topology*

If there is packet loss happening in the network segment between R2 and R3, you will notice in the Wireshark capture that there are multiple retransmissions between the source and destination. In Figure 4-16, the segment for which the sender did not receive the acknowledgment retransmitted the segment back. In this case the sequence number of that segment was 3546854380 (relative sequence number 213) and the acknowledgment number was 597044005 (relative acknowledgment number 501).

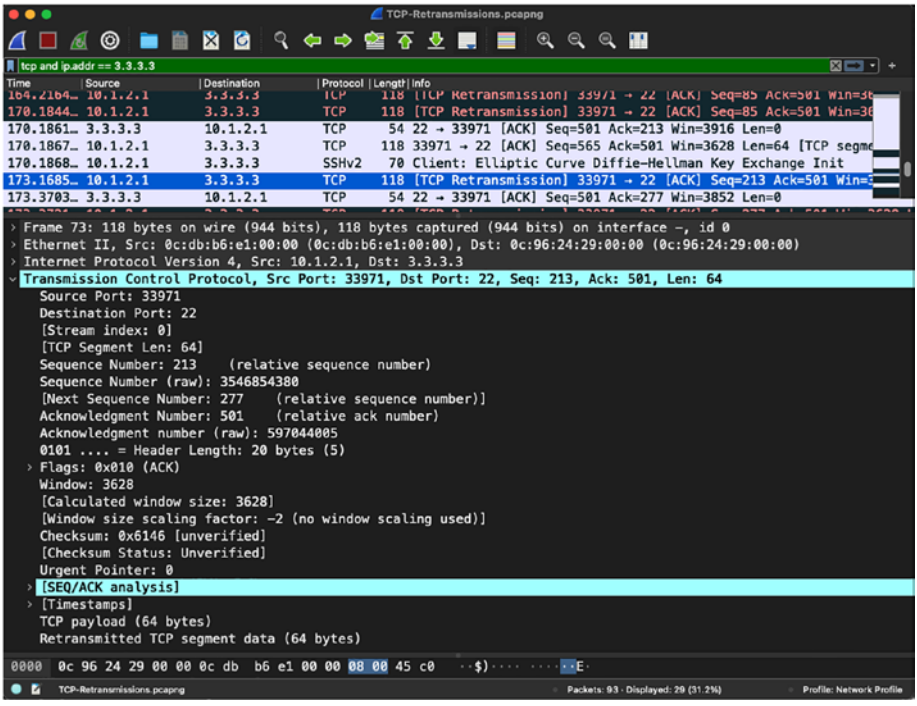


Figure 4-16. *Wireshark capture with TCP retransmissions*

TCP Out-of-Order Packets

In networks, users might also encounter TCP out-of-order (OOO) packets. TCP OOO packets simply mean that the packets arrive at the destination in a different order from that in which they were sent. This could happen for several reasons:

- *Multiple paths:* If the TCP segments are following multiple paths (ECMP paths to the destination) or via parallel processing paths within a router or a network equipment (e.g., per-packet load balancing), and either of the systems are not designed to ensure the ordering of the packets, this could lead to OOO packets in the network. Note that it is TCP's job to deliver the packets in the right order.
- *QoS:* Poorly configured QoS, especially a queueing mechanism, can cause OOO packets in the network. If the QoS settings do not forward the packet in a first in, first out (FIFO) manner or if the QoS settings drop the TCP packets along the path, this could lead to retransmission of those dropped TCP segments and eventually to OOO packets.
- *Oversubscription:* Oversubscribed links in the network can cause OOO packets. The traffic will end up getting dropped, causing retransmission, slowdowns, and OOO packets.
- *Microbursts:* A microburst is a behavior seen in networks when rapid bursts of data packets are seen in quick succession, leading to time intervals of full line-rate transmission. This can cause packets to get dropped due to buffer overflows on the interface. When such bursts occur in the network, links would end up dropping packets, causing retransmissions, slowness, and OOO packets.

When OOO segments are received by the TCP software, one of the main functions that it performs is reassembling packets in order or requesting retransmission of OOO packets. If the Wireshark capture shows that there are OOO packets, then as part of the troubleshooting process you might want to look at the possible causes listed earlier.

Tip If you see a lot of TCP OOO packets, there is packet loss between the capture point and the sender. If you see a lot of TCP retransmissions, though, there is packet loss happening between the capture point and the receiver.

Troubleshooting with Wireshark Graphs

When troubleshooting TCP or any network issues in a large-scale environment, where there is large amount of data to be analyzed, it becomes challenging and time consuming to identify the root cause. In such a scenario, a quick peek at graphical data can give us a better understanding of what is happening in the network. Wireshark provides you with numerous graph options that can be used for investigating various types of issues. There are graphs in Wireshark that are specific for TCP and can come in handy for day-to-day network analysis and troubleshooting tasks. This section focuses on some of the various graphs that can be used.

TCP Stream Graphs

TCP Stream Graphs can be used to provide visual insights about TCP streams. The Wireshark tool gives user options to select between all packets and TCP packets. The graphs are part of the Wireshark profile and can also be imported from another profile. Within TCP Stream Graphs, there are different graph and analysis options that network analysts can use:

- Time Sequence (Stevens)
- Time Sequence (tcptrace)
- Throughput
- Round Trip Time
- Window Scaling

All these options can be accessed in Wireshark on the Statistics menu by selecting TCP Stream Graphs, as shown in Figure 4-17.

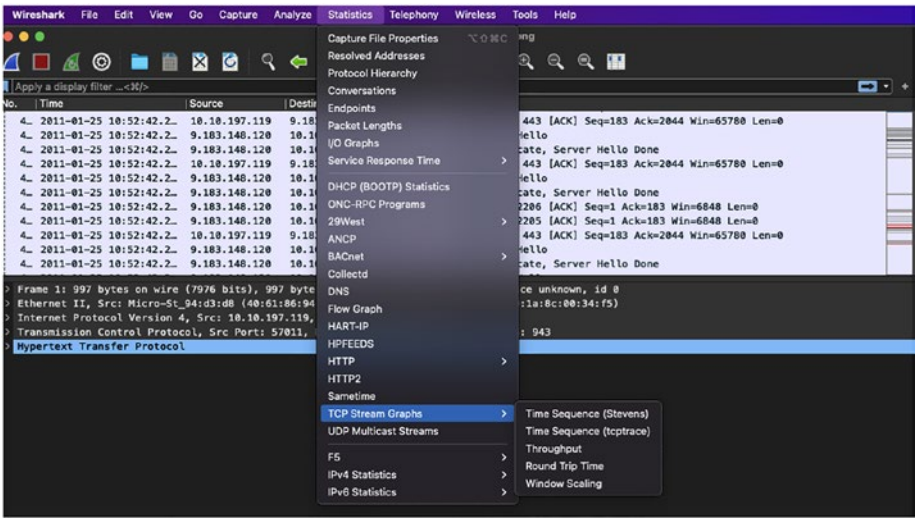


Figure 4-17. *TCP Stream Graphs options*

Time Sequence (Stevens)

This TCP stream time sequence graph shows TCP sequence numbers plotted against time in any single direction. You do have options to switch between the direction of the TCP stream, but only one direction can be analyzed at any given point in time. If the captured traffic is only TCP

traffic, then you can simply select Time Sequence (Stevens) graphs from the menu. This will display the graph based on sequence numbers vs. time in seconds as shown in Figure 4-18.

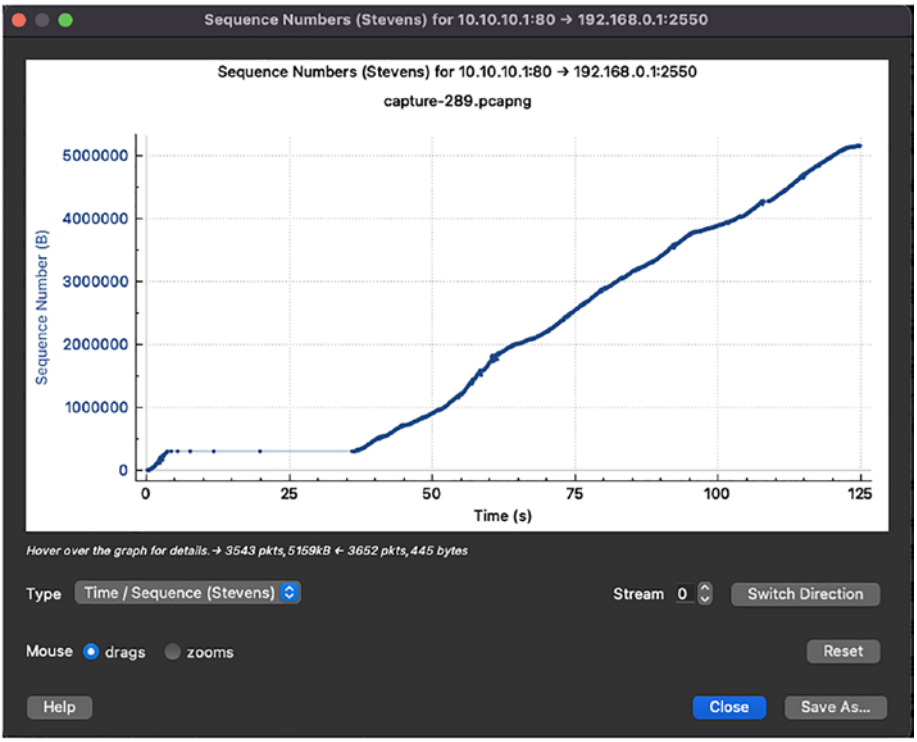


Figure 4-18. *TCP Stream Graphs: Time Sequence (Stevens)*

In an ideal situation you might want to see a smooth line from the bottom left corner to the top right corner of the graph. Notice that in the graph in Figure 4-18, the graph is mostly incremental and has a smooth line, but there are flat periods in the graph. The flat periods in this graph are bad in that they indicate that the sequence number in that direction is not increasing. When you click on these flat periods, you will notice that there are TCP errors seen during those periods in the Wireshark capture. In this case, when we click on one of the flat periods, we can see TCP

ZeroWindow error shown in Figure 4-19, which indicates that the window size is 0 in the TCP header. A TCP window size of 0 usually indicates the client (or server, but in most scenarios it is the client) has advertised the value of 0 for its window size, indicating that the TCP receive buffer is full and cannot receive any more data.

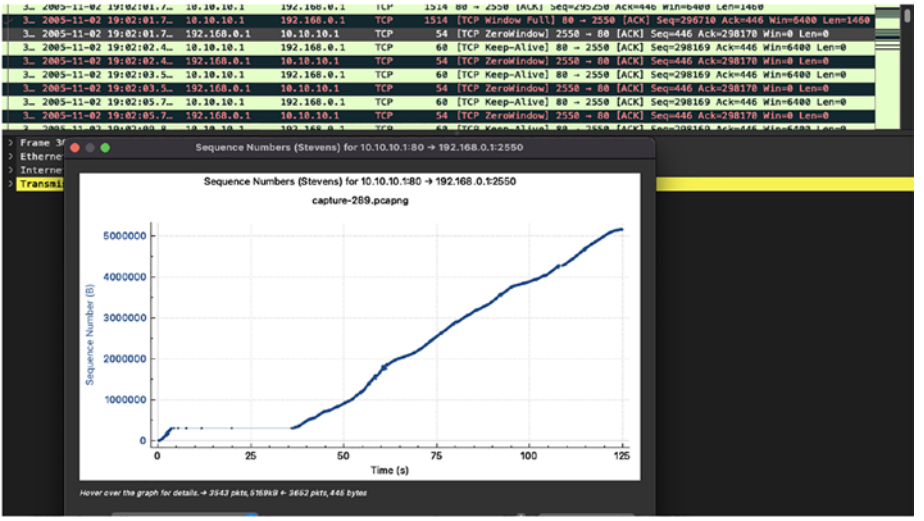


Figure 4-19. *TCP ZeroWindow Error seen in Stevens Time Sequence graphs*

If there are dips in the graph, it would usually indicate TCP retransmissions or OOO packets. Figure 4-20 displays the dip in graphs indicating TCP retransmissions as well as OOO packets in the network.

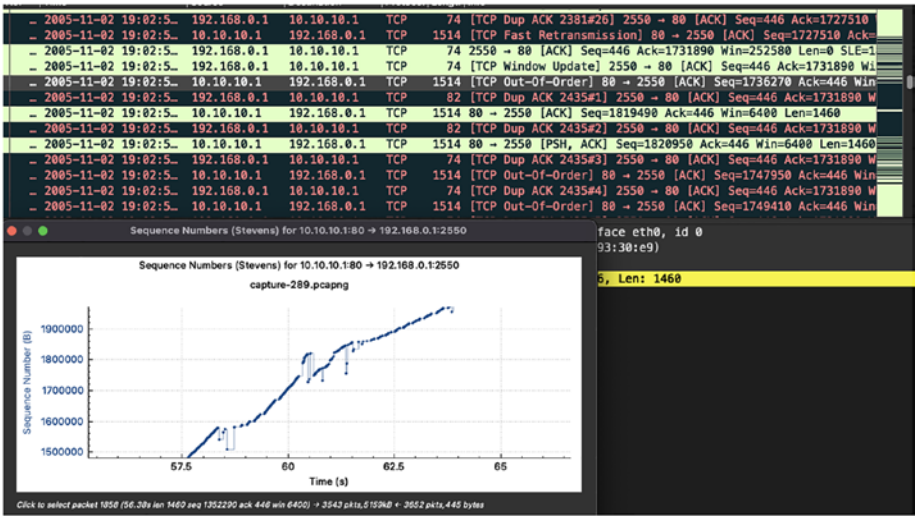


Figure 4-20. *TCP retransmissions and OOO packets in Stevens graph*

Time Sequence (tcptrace)

The tcptrace Time Sequence graph is similar to the Stevens graph, but on steroids. It shows the bytes in flight as well as the receive window information, which is highlighted. This graph also shows other information such as acknowledgments and selective acknowledgments (SACKs) received. Figure 4-21 displays the tcptrace Time Sequence graph. Notice the green line in the graph; this indicates the receive window (rwnd) received from the destination host. The blue sections or blue dots in the graph indicate the packets in transit. The red lines in the graph indicate SACKs.

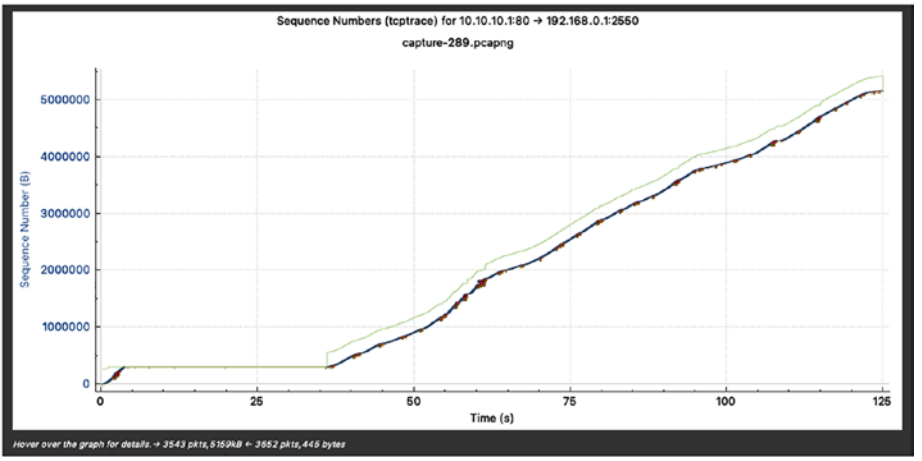


Figure 4-21. *Tcptrace Time Sequence graph*

When we further expand the graph as seen in Figure 4-22, notice the brown lines in the graph. These brown lines indicate acknowledgments received from the receiver end. The red lines in the graph indicate SACKs.

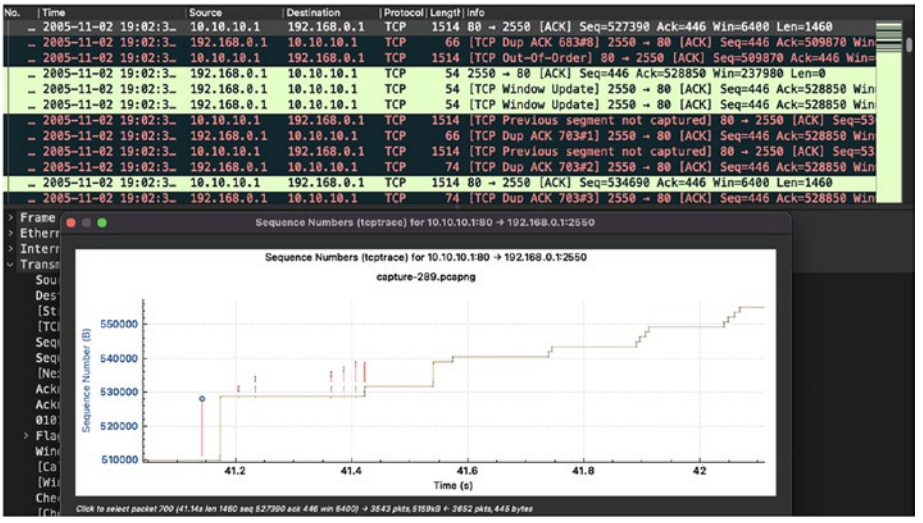


Figure 4-22. *ACKs and SACKs in tcptrace Time Sequence graphs*

When looking at these graphs, there are two things that we do not want to see:

1. The bytes in flight (blue lines or dots) touching the receive window (graph lines).
2. Steps (these denote that the sender is not sending the data fast enough or it could be related to a receive window size issue).

Figure 4-23 shows a pretty big step in the graph and when a user clicks on that step, they can see TCP ZeroWindow errors in the Wireshark capture indicating the receiver's TCP receive buffer is full and it cannot process any further packets at the moment.

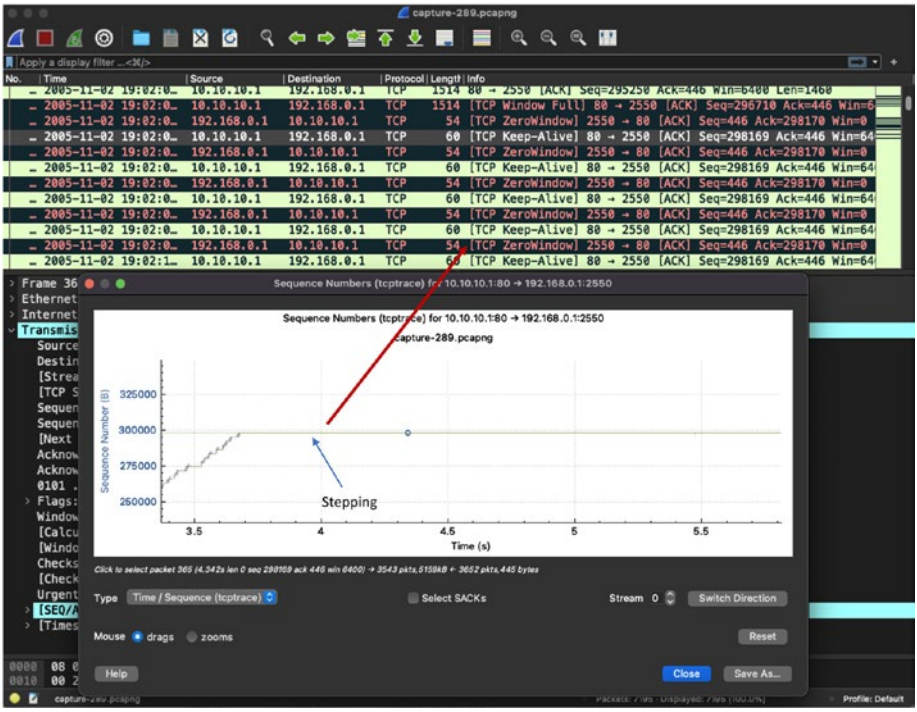


Figure 4-23. *Steps in tcptrace Time Sequence graphs*

Throughput Graph

The Throughput graph is very useful during throughput testing in a greenfield deployment or during migration testing in the network. This graph shows the segment length (packet size) and average throughput vs. bytes per second (bps) over time. It also has options to show both the throughput and the goodput in the graph. Figure 4-24 shows the Throughput graph. Notice that in this graph, the segment length is stable during the capture but there is also a gap in the segment length section that indicates that the sender is not sending anything.

Note In computer networks, goodput means the application-level throughput of a communication. It simply indicates good throughput of an application.

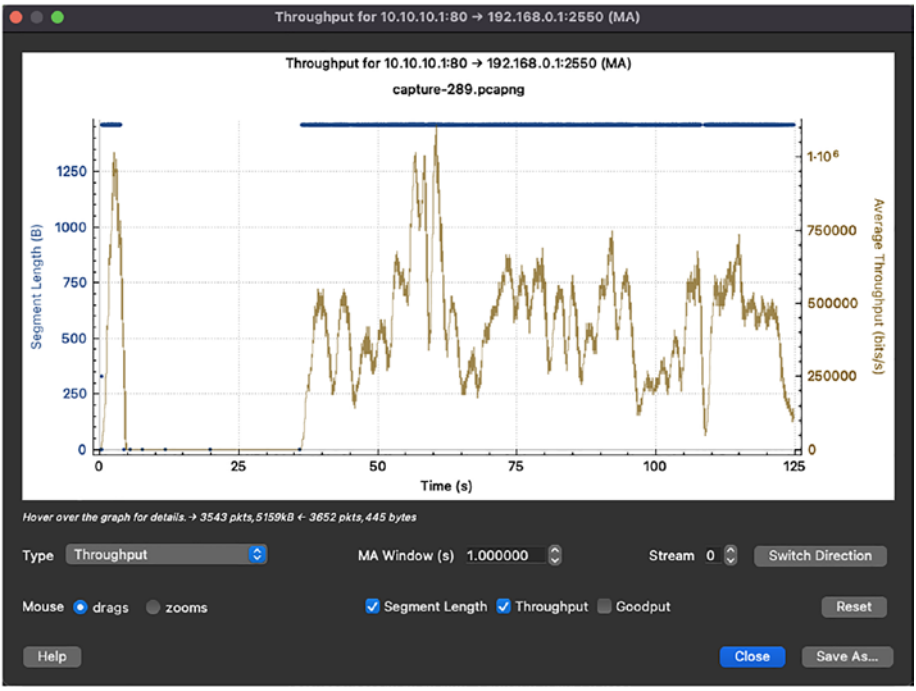


Figure 4-24. *Throughput graph*

If the graph shows sporadic segments (dotted lines), it indicates that the device is sending sporadically as shown in Figure 4-25, and it usually indicates that there is packet loss in the path. If users click those sporadic segments, they might be able to see TCP retransmissions or OOO packets.

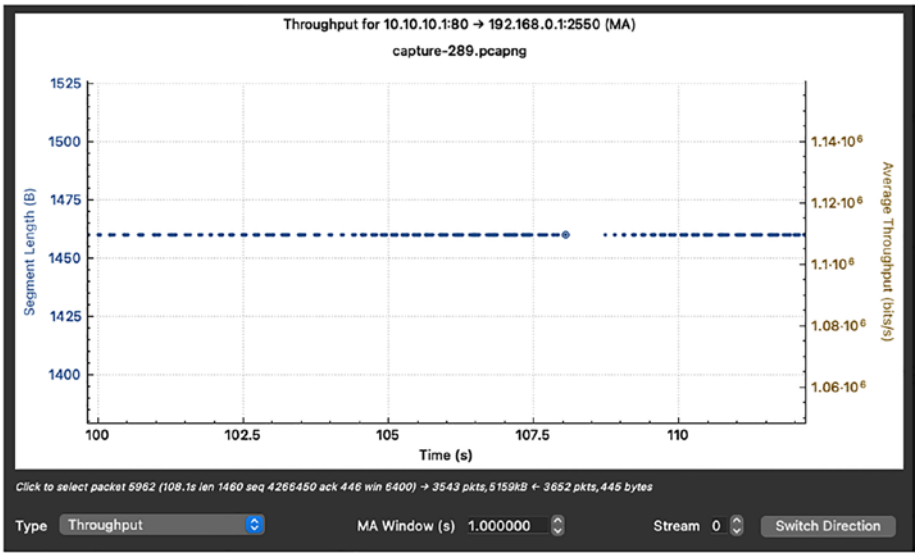


Figure 4-25. *Sporadic segments*

Window Scaling Graph

The Window Scaling graph can be very useful when troubleshooting TCP window issues. These issues usually occur when one end is sending more traffic than the other end can handle, or the receiving end has no buffer left in the TCP receive window (as seen in some previous examples). This graph, displayed in Figure 4-26, shows the TCP receive window (in green) vs. bytes in flight (in blue). Note that in an ideal situation, the bytes in flight should never be more than the receive window size.

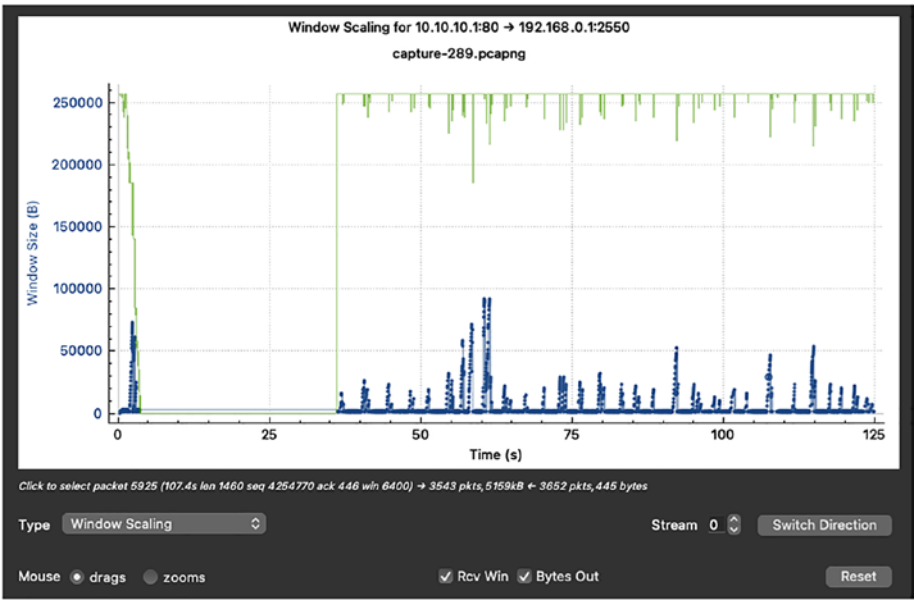


Figure 4-26. *Window Scaling graph*

Further zooming into the graph, if we see the flat lines (steps) in the graph, it usually represents round trip time (RTT). An RTT is the difference between the time when the packet was sent out and an ACK was received for that packet. Figure 4-27 displays the flat lines in the Window Scaling graphs indicating the RTT.

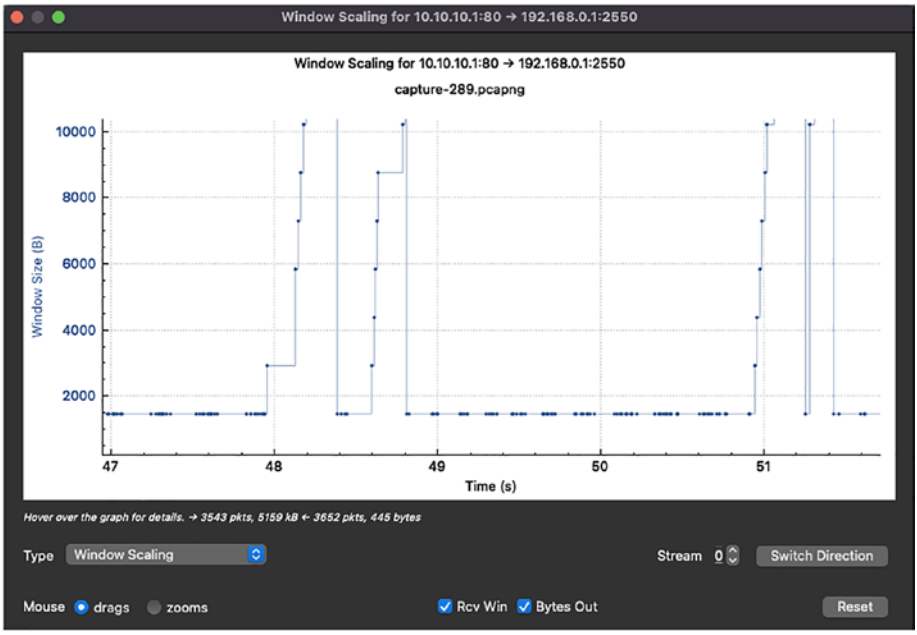


Figure 4-27. *RTT in Window Scaling graph*

Note that if the bytes in flight (the blue dots and line) start from the bottom (i.e., at 0), it indicates that all the previous segments that were transmitted have been acknowledged and there are no packets in flight. If the bytes in flight start above the 0 value (baseline) it indicates that there are segments and bytes that have not yet been acknowledged. Figure 4-28 shows the bytes in flight starting above the baseline.

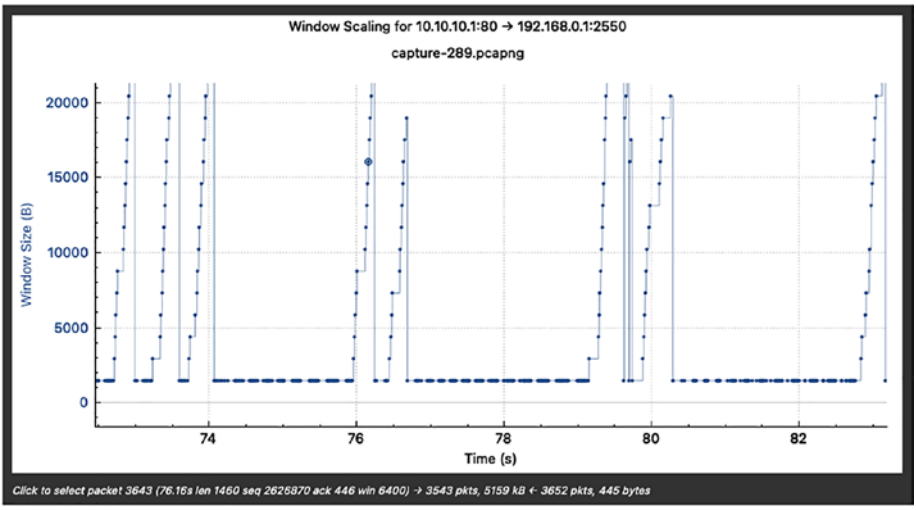


Figure 4-28. *Unacknowledged bytes in flight in Window Scaling graph*

RTT Graph

If there are jitters in the network, you might want to leverage the help of the TCP Streams RTT graph. The RTT graph measures the RTT of all TCP packets. If the graph shows big spikes, it usually indicates there is either packet loss in the network or network congestion. Figure 4-29 shows the RTT graph for all the captured packets in Wireshark. Notice that initially in the graph, there are very large spikes, but the later part of the graph shows consistent RTT. This indicates that initially there was either congestion or packet loss that increased the RTT, but after it was resolved the RTT was fairly stable.

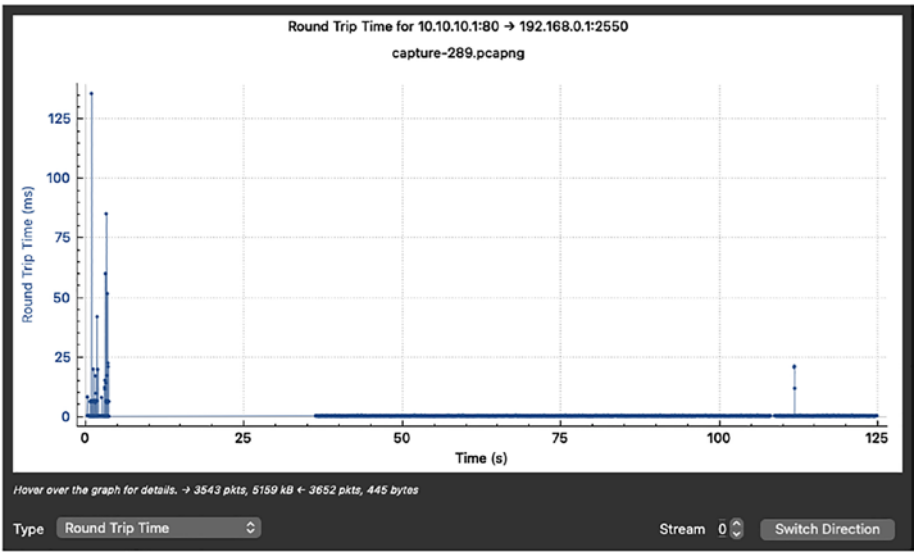


Figure 4-29. *RTT graph*

I/O Graphs

The I/O graphs provide a customizable list of graphs allowing users to compare different types of traffic and correlate the events with the application traffic based on errors seen in the network quickly and easily. The I/O graphs allow users to customize the different graphs they wish to see simultaneously, which makes it easier to correlate the data with network events. For instance, you might observe a dip in the requests coming in on a web server on HTTP as well as HTTPS using the I/O graphs while comparing it with any TCP errors seen during that instance. Figure 4-30 displays the traffic pattern of both HTTP and HTTPS traffic from the Wireshark capture. The green lines highlight the HTTP traffic, whereas the red line indicates the HTTPS traffic. Note that the HTTP and HTTPS graph filters are not present by default. These can be added by clicking the + icon, assigning the graph name, and under Display Filters,

setting the filter to `tcp.port == 80` for HTTP or `tcp.port == 443` for HTTPS. Once the filters are set, users can customize these graphs with the colors of their choice.

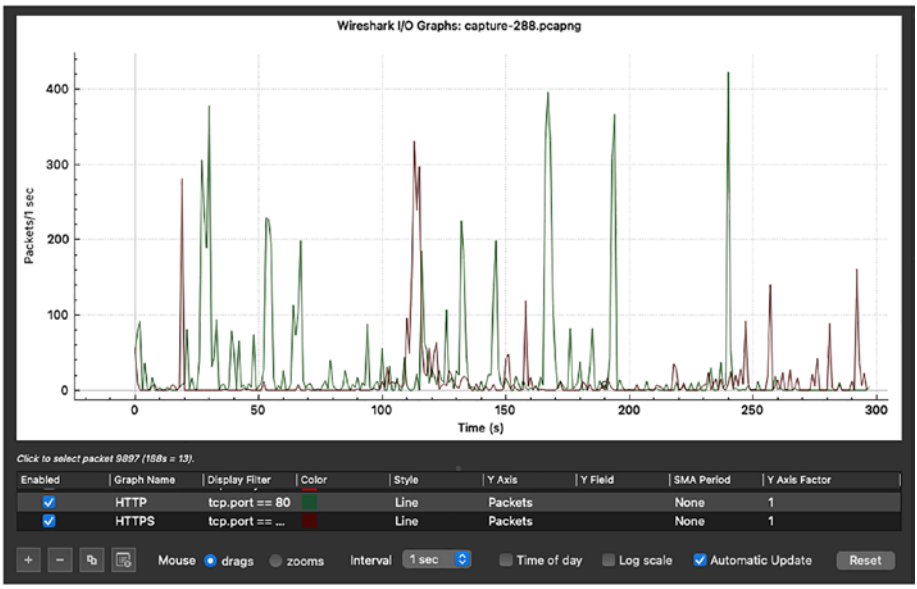


Figure 4-30. *HTTP vs. HTTPS traffic in I/O graphs*

If we look at another Wireshark capture where we only have HTTP traffic, but a lot of TCP errors, we can easily correlate the dip in the traffic with a high number of TCP errors. In Figure 4-31, the traffic pattern of HTTP traffic is shown along with TCP errors. When there are major dips seen in the HTTP traffic, we can also see the spike in the TCP errors. When looking at the Wireshark capture around that time, we will be able to determine that there was packet loss happening around that time.

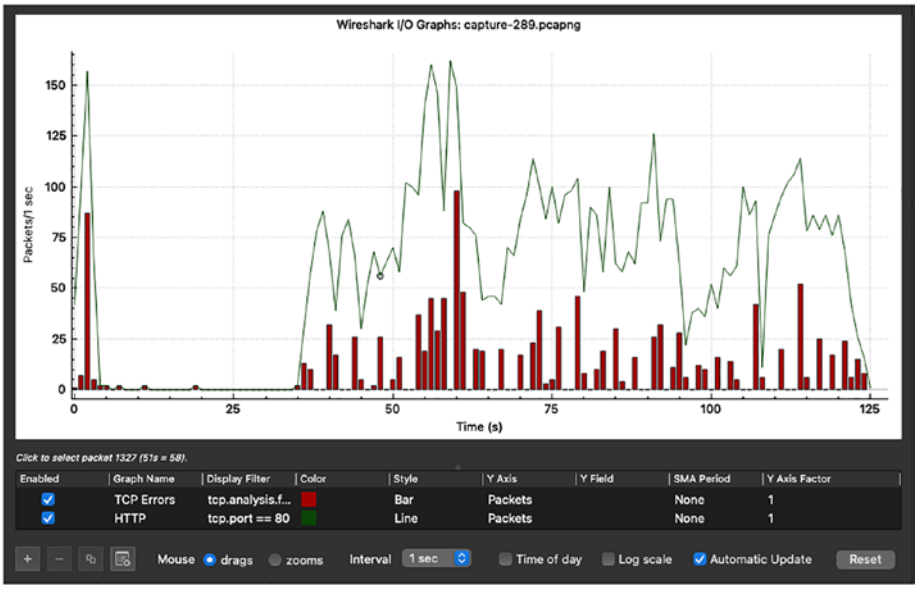


Figure 4-31. *HTTP traffic and TCP errors in I/O graphs*

The I/O graphs can also be used to analyze any type of microbursts happening in the network. The I/O graphs give options to plot the graph, not just at a 1-s time interval, but also to the millisecond level (performance could be affected based on the amount of data being analyzed by the Wireshark capture). Figure 4-32 shows the I/O graphs adjusted to a 100-ms time interval. The graph in this scenario shows spikes from time to time. The traffic spikes in these scenarios might not be relevant because there are not too many packets being sent within the 100-ms time interval, but if there were more packets sent during the 100-ms time interval, it would have been a concern.

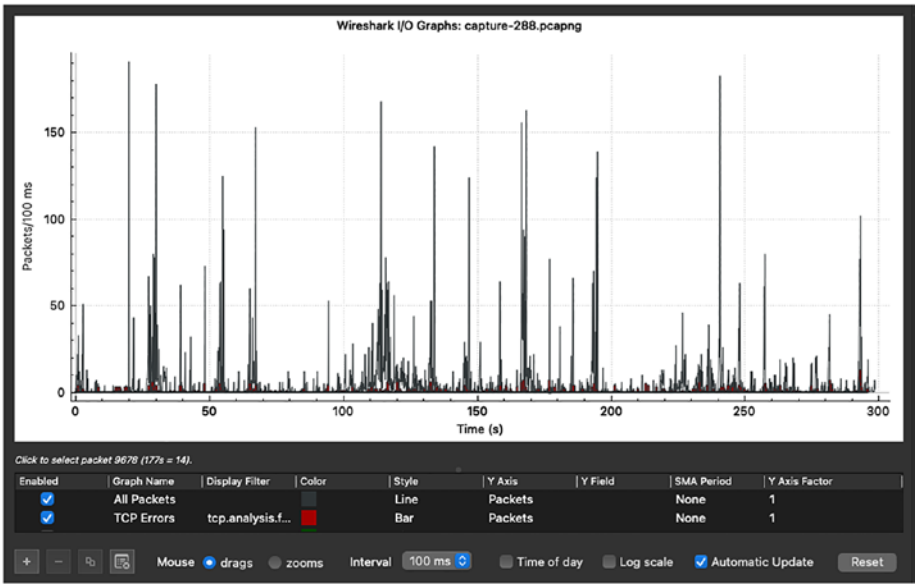


Figure 4-32. *Microbursts in I/O graphs*

Flow Graphs

When troubleshooting TCP-related network problems, it is necessary to track the flow such as the three-way handshake, data flow and acknowledgments, and so on. Just looking at Wireshark it might be difficult to identify the flow unless you are using the option to follow the TCP stream, in which case it will give you the complete flow of that packet. However, it might still be difficult to understand the direction of each packet, as you will have to keep track of the source and destination IP addresses. This challenge is solved by another Wireshark graph known as Flow graphs. Flow graphs provide you with a graphical representation of all the TCP flows from the Wireshark capture and help you visualize the TCP flow along with its direction. Figure 4-33 shows the Flow graph of the TCP packets from the Wireshark capture.

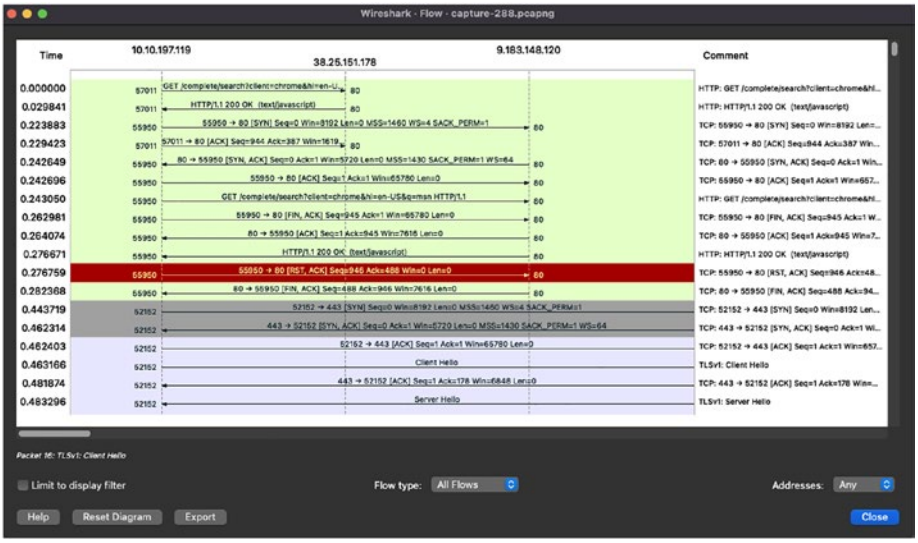


Figure 4-33. *Flow graph*

One of the benefits of using Flow graphs is that they preserve the colors from the Wireshark profile and allow you to apply filters. The Flow graph comes in very handy when troubleshooting VoIP-related issues. It shows all the conversations related to DNS, TCP, HTTP, and so on, for the specified traffic. These are a few of the most common use cases of Flow graphs:

- Tracking any malicious host or application that is trying to access multiple servers on the network.
- Tracking TCP retransmissions
- Tracking connection resets

For applying filters in Flow graphs, you can simply apply a display filter in the Wireshark tool using filter expressions and then use Limit To Display Filter check box. When this check box is selected, it will automatically change the Flow graph to only the flow that is being targeted in the display filter. Figure 4-34 shows the Flow graph of an HTTP flow that has been

filtered on the Wireshark display filter. Next to the Limit To Display Filter check box, there is also a drop-down list that allows users to further trim down the visual Flow graph to particular types of flows such as ICMP, TCP, and so on.

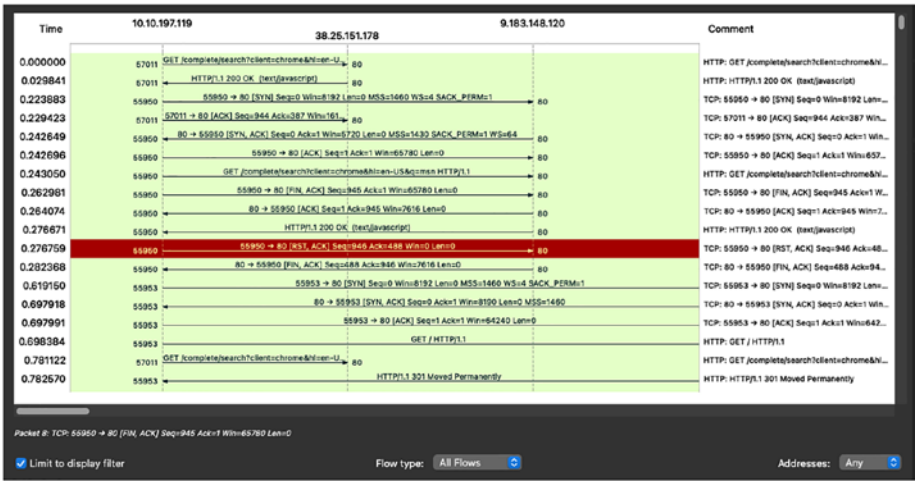


Figure 4-34. *Flow graph filter*

TCP Expert

When working on a complex problem, you must know the right filters, use the right options, and have your own profile in Wireshark for different protocols to be able to analyze and identify the problem as quickly as possible. Knowing and using various display filters for troubleshooting different types of TCP issues can save you a lot of time. Table 4-2 displays a list of common TCP-based display filters and what they do.

Table 4-2. *TCP Display Filters and Their Functions*

Display Filter	Function
`tcp.flags == 0x2` `tcp.flags.syn == 1`	Capture all TCP SYN packets
`tcp.flags.reset == 1`	Capture TCP Resets

`(tcp.flags == 0x10) &&`
`(tcp.seq == 1) &&`
`(tcp.ack == 1)`
	Capture only third packet of the TCP three-way handshake

`tcp.time_delta > 1`	Filter TCP delays greater than t seconds; in this example, t = 1

`tcp.time_delta > 1 &&`
`tcp.flags.fin == 0 &&`
`tcp.flags.reset == 0`
	Identifying TCP delays but ignoring delays from the TCP connection termination process (during the connection termination process, TCP FIN is sent to the remote end, or the TCP reset flag is set)

`tcp.window_size >= 0 &&` `tcp.window_size < 500`	Identifying small TCP window sizes
`tcp.analysis.out_of_order`	Filtering TCP OOO packets
`(tcp.flags.syn == 1) &&` `(!tcp.len == 0)`	Filtering TCP SYN or SYN-ACK packets that contain data

Wireshark Profile for TCP

Wireshark allows users to create custom profiles that can come in very handy based on the type of issue being investigated. Every Wireshark application comes with a Default profile that has the following fields:

- No.
- Time
- Source
- Destination
- Protocol
- Length
- Info

The Default profile is good for beginners and yields a lot of useful information, but troubleshooting TCP issues is a complex process and requires more specific fields related to TCP to quickly analyze TCP packets. To create a new profile, follow these steps:

1. Right-click Profile: Default at the bottom right corner of the Wireshark application (Figure 4-35).
2. Select the New option. This will open the profile modally.
3. Create a profile named TCP and click OK.
4. Right-click again at the bottom right corner of the Wireshark application and select the TCP profile from the Switch To submenu.

5. Once selected, the new TCP profile will become your active profile. Note that at this point, this new profile will have the same columns and settings as the Default profile.

To change the settings of the new TCP profile, select Preferences from the Wireshark Menu then go to Appearance | Columns and then add the following columns with the types and settings as shown in Figure 4-36.

- No.
- Time
- Delta
- Source
- Destination
- TCP Delta
- SEQ
- ACK
- Window
- Bytes in flight
- Info

Figure 4-35. *Selecting the Default profile*

Once the columns are added, the TCP profile UI yields more granular information about TCP as shown in Figure 4-36. You can see how easy it looks to point out packets with window size 0. Not just for TCP, but in general, network and security analysts should always create and use custom profiles and use custom fields in their UI based on their style of troubleshooting.

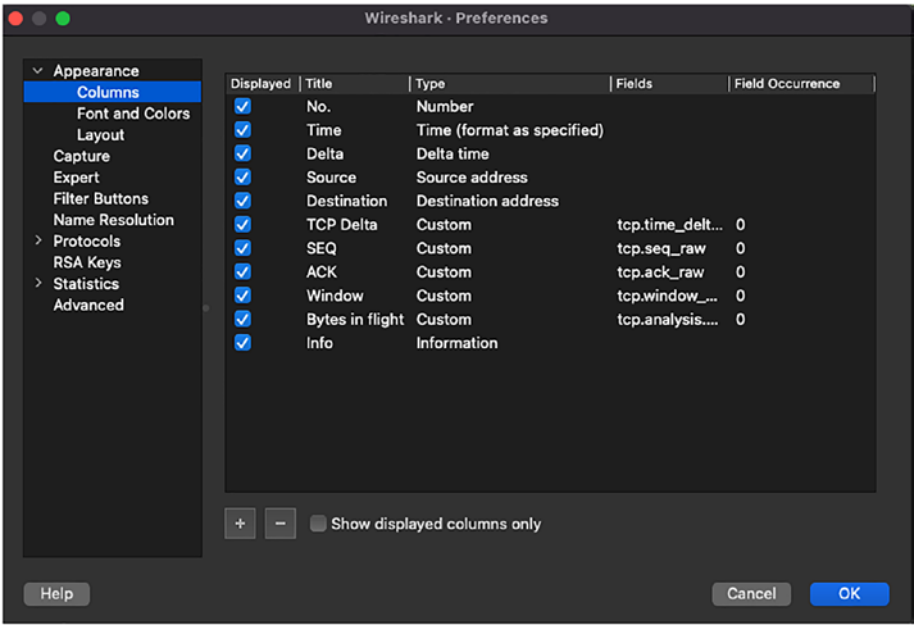


Figure 4-36. *Custom columns for TCP profile*

Most of the fields in the Columns list are self-explanatory. The only field that needs some explanation is the TCP Delta field. The TCP Delta simply means the time since previous frame in the given TCP stream. This field helps identify if there have been delays in the network in turn causing delays in receiving the TCP stream. The information in the TCP Delta field is available in the Timestamp section of the TCP header but let's not get confused with every delta that you see. Some delays are normal, such as these:

- *SYN packets:* There might be a delay before the initial SYN packet. For instance, once the Wireshark capture is started, you might ask the user to connect to a web server. There will be a delay in such a case before the first packet is seen on the wire.
- *Connection termination packets:* TCP connection termination packets are basically FIN, FIN-ACK, RST, and RST-ACK. These packets are explicitly sent to close or terminate a connection. These packets could be sent when a user opens a new tab on the browser, or the session gets automatically closed after a page is loaded.
- *GET requests:* GET requests can be generated in HTTP when a user clicks a link to open a new page or to request new data from the back end of the web application. Some GET requests are instant, but there might be GET requests initiated by background processes that might not have any priority, for instance, a GET request for `.ico` files.
- *DNS queries:* DNS queries during a web browsing session are common and could lead to unexpected delays in response.
- *Image files:* When a browser application requests image files or **.ico** files, there might be delays for such requests based on the web server settings or file size of the image.

User Datagram Protocol

Unlike TCP, UDP is a lightweight connectionless protocol that is used to transfer data in the network. UDP is different than TCP in several ways:

- No handshake mechanism
- No session teardown
- Smaller header size
- Unreliable data delivery
- No mechanism to manage OOO packets
- No protection from data duplication

UDP as a transport protocol thus seems useful in scenarios where error checking and correction mechanisms are either not necessary, or these functions are performed by the end applications. The UDP protocol was designed by David P. Reed in 1980 and was standardized in RFC 768. It is a simple message-oriented Transport layer protocol that primarily consists of four fields of 2 bytes each, as shown in Figure 4-37. The UDP header is always 8 bytes in length, as it does not have any Options field in the header.

- *Source port:* Identifies the sender's port number.
- *Destination port:* Identifies the receiver's port number.
- *Length:* Specifies the length in bytes of the UDP header and payload; minimum length is 8 bytes.
- *Checksum:* This field is used to ensure the integrity of the data. This field stores the 16-bit words summed using 1's complement arithmetic, which is calculated based on the IP header, the UDP header, and the payload.

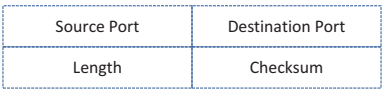


Figure 4-37. *UDP header*

Knowingly or unknowingly, you use UDP in various applications on your network computer. Applications such as DHCP, DNS, Trivial File Transfer Protocol (TFTP), and more, all use UDP as their transport protocol. If you are interested in checking which UDP ports are in use on your system, use the command `netstat -anp udp`. Example 4-1 displays the output of this command on a Mac OS and Windows OS.

Example 4-1. Netstat Command for Verifying UDP

```
genie@VinJ ~ % netstat -anp udp
udp4       0      0  10.65.55.185.*         8.8.8.8.53
udp4       0      0  10.65.55.185.*         8.8.8.8.53
udp4       0      0  10.65.55.185.*         8.8.8.8.53
udp4       0      0  10.65.55.185.*         8.8.8.8.53
! Output omitted for brevity
C:\Users\Administrator>netstat -anp udp

Active Connections

  Proto  Local Address          Foreign Address        State
  UDP    127.0.0.1:1900         *:*
  UDP    127.0.0.1:56629        *:*
  UDP    127.0.0.1:57233        *:*
  UDP    127.0.0.1:65272        *:*
  UDP    192.168.0.3:137        *:*
  UDP    192.168.0.3:138        *:*
  UDP    192.168.0.3:1900       *:*
  UDP    192.168.0.3:5353       *:*
  UDP    192.168.0.3:56527      *:*
  UDP    192.168.0.3:57232      *:*
  UDP    192.168.0.3:65271      *:*
```

You can follow these simple steps to capture UDP traffic on your computer system:

1. Start the Wireshark application and start a capture on your computer's NIC.
2. Open a command prompt.
3. Clear your DNS cache using the `ipconfig /flushdns` command.
4. Try initiating a ping to a remote server or website from the command prompt.
5. Close the command prompt.
6. Stop the Wireshark capture.

Figure 4-38 shows the Wireshark capture of a DNS query for `www.apple.com`. The destination UDP port of 53 indicates that this is a DNS packet. If there are too many packets in the Wireshark capture file, you can simply filter the DNS packets using the display filter `udp.port == 53`. If we look at the UDP packet, we can see the source port is 51053, the destination port is 53, which is used for DNS, the length of the packet is 39 bytes, and the checksum value is set to 0x22e5. Note that at the end of the UDP header, you can see that the UDP payload is 31 bytes and adding 8 bytes of UDP header it equates to 39 bytes.

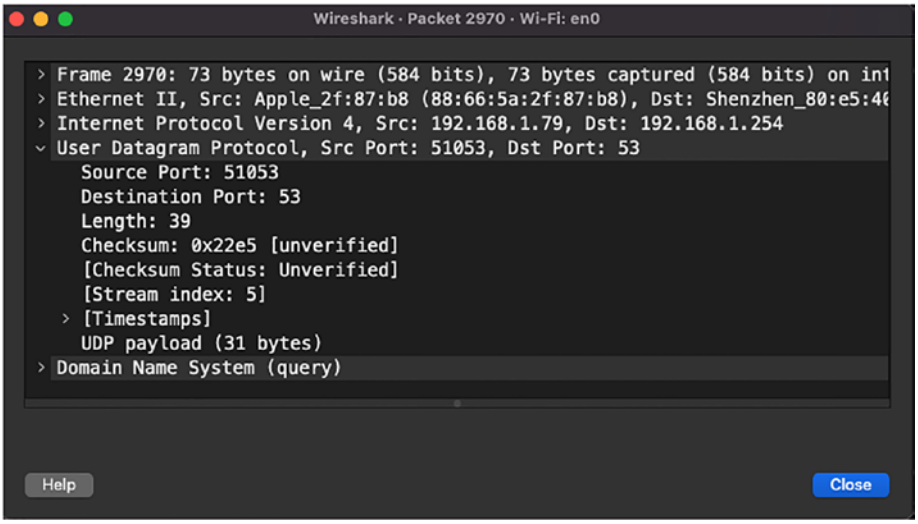


Figure 4-38. *Wireshark capture of DNS query*

Once the Wireshark capture has been performed, users can also follow the UDP streams by selecting one of the flows in Wireshark, right-clicking, and from Follow menu, selecting UDP Stream. This will show both the DNS query and the DNS response in the Wireshark window with the filter being set to `udp.stream eq stream-number`. Figure 4-39 displays the complete UDP stream for a DNS query to `www.apple.com`.

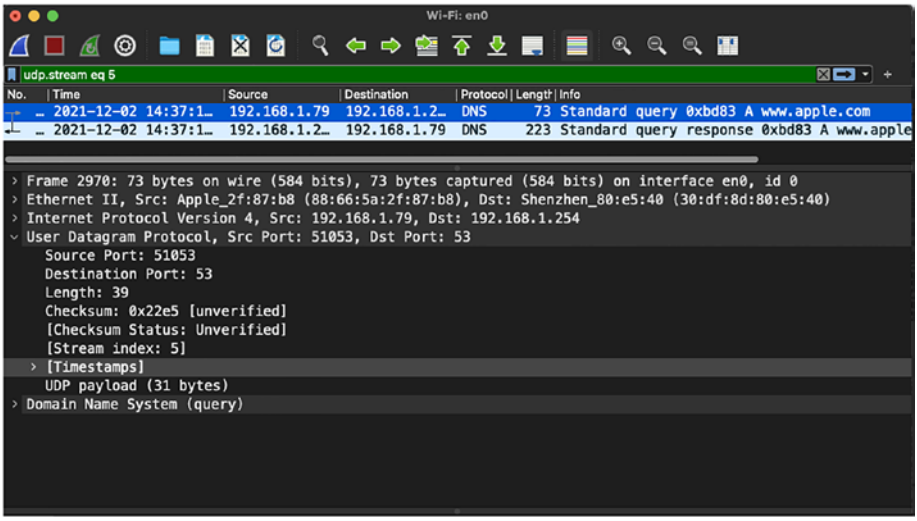


Figure 4-39. *Filtered UDP stream in Wireshark*

There isn't much as user can do when it comes to troubleshooting UDP. If there is a packet loss in the network, the application can simply request the data again, but the UDP software itself does not track any sequence number. A UDP data packet lost is data lost. There is some analysis that can still be done in Wireshark using I/O graphs, as these are not specific to just TCP, but any kind of stream captured in Wireshark. Users can filter a UDP stream on Wireshark and then select the I/O Graph option from the Statistics menu. The I/O graph will display options such as All Packets and TCP Errors, but also Filtered Packets with the filter set to UDP packets that was used as the display filter in Wireshark. Figure 4-40 displays the I/O graph of filtered UDP packets on Wireshark.

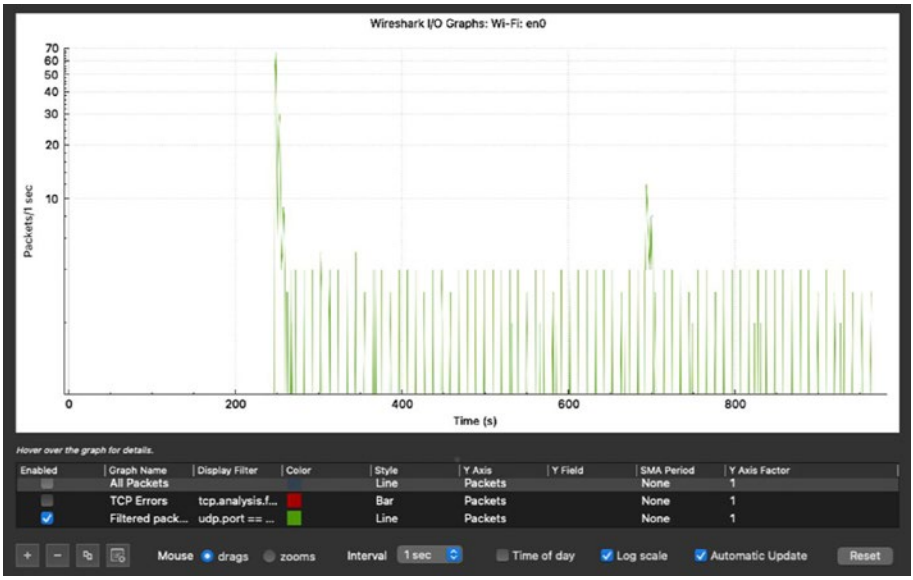


Figure 4-40. *I/O graph for filtered UDP packets*

Summary

Any network engineer or security analyst should have a deep and solid understanding of protocols working at different layers, but one of the layers that most engineers least on least is the Transport layer. The Transport layer protocols are crucial for ensuring end-to-end communication and transporting data between sender and receiver. To transport the data, the Transport layer has protocols that follow either connectionless or connection-oriented architecture with both having their respective use cases. In this chapter, we focused on the two key Transport layer protocols, TCP and UDP.

In this chapter, we explored the working of TCP and how it helps solves several problems such as data reliability, data integrity, and so on. We learned about the TCP connection process using three-way handshake, how port scanning is done by attackers, and how network engineers can

investigate packet loss issues in the network. We learned that packet loss in the network can lead to issues such as TCP retransmissions and TCP OOO packets. We then saw how quick analysis and troubleshooting can be performed for network traffic using Wireshark graphs, including TCP Stream graphs, I/O graphs, and Flow graphs. As a network engineer or security analyst it is important to have custom profiles in Wireshark to analyze different types of traffic. We covered how users can create custom profiles for TCP and can quickly identify issues such as ZeroWindow by simply looking at the capture.

At the end of the chapter, we looked at UDP and in which scenarios UDP is used by different applications. We then learned how to filter UDP traffic and how we can leverage I/O graphs to learn about the UDP traffic pattern.

References in This Chapter

- RFC 793: Transmission Control Protocol, DARPA, Information Sciences Institute University of Southern California, IETF, September 1981. http://tools.ietf.org/html/rfc793.

RFC 768: User Datagram Protocol, J. Postel, IETF, August 1980. http://tools.ietf.org/html/rfc768.

CHAPTER 5

Analyzing Control Plane Traffic

This chapter covers the following topics:

- Analyzing routing protocol traffic
- Analyzing overlay traffic

Analyzing Routing Protocol Traffic

So far, we have learned how to set up Wireshark, perform packet captures, and analyze Layer 2 to Layer 4 traffic. Most of the traffic that we have looked at so far is data traffic. When we are dealing with packet loss in the network, we usually try to understand the problem based on what is happening in the network: Is there an errored link in the network dropping the traffic? Is network congestion leading to data loss? When the data loss is happening in the network, chances are high that the data might also be control plane traffic. Although we can give separate treatment to the control plane traffic from the data traffic using QoS, that only helps prioritizing packets on the device, not on the wire. So, a packet loss can simply drop data traffic as well as control plane traffic. Thus, a control plane flap due to any amount of packet loss can still be analyzed using the methods that we have seen so far. It could also be the case, though, that control plane protocols misbehave even when there is no packet

V. Jain, *Wireshark Fundamentals*, https://doi.org/10.1007/978-1-4842-8002-7_5

loss or congestion in the network. This chapter is focused on analyzing control plane traffic and understanding the headers and functionality of various routing protocols, diving deeper into certain cases and how we can troubleshoot them using Wireshark. Note that this chapter does not focus on teaching any control plane or data plane traffic, but just analyzing different control plane and data plane traffic, which can prove useful for network engineers. It is assumed that network engineers are well aware of the protocols discussed in this chapter.

OSPF

Open Shortest Path First (OSPF), defined in RFC 2328, is one of the well-known and most widely adopted interior gateway protocols (IGPs). It is a dynamic routing protocol that operates within a single autonomous system (AS) and is suitable for large heterogeneous networks. OSPF uses the Dijkstra algorithm, also known as the shortest path first (SPF) algorithm, to calculate the shortest path to the destination. In OSPF, the shortest path to a destination is calculated based on the cost of the route, which considers variables such as bandwidth, delay, and load.

OSPF allows network administrators to break large networks into smaller segments known as OSPF areas. This allows network administrators to reduce the OSPF areas, which are basically a collection of networks that support multiple area types:

- *Backbone area:* Network segment that belongs to area 0.0.0.0. All other areas are either physically or virtually connected to the backbone area. Exchanging routing information between multiple nonzero or nonbackbone areas is only possible through the backbone area.

- *Standard nonzero area:* In this area, OSPF packets are normally transmitted. This area is directly or virtually connected to the backbone area.
- *Stub area:* This area does not allow and accept routes from external sources such as routes learned by other routing protocols and redistributed into OSPF.
- *Totally stubby area:* This area does not accept routes from external sources and link information from other areas. Instead, a default route is advertised in this area for allowing the router in this area to reach a destination in other areas or even external sources.
- *Not so stubby area (NSSA):* An NSSA is derived from a stub area with the difference that this area also has an Autonomous System Boundary Router (ASBR) router attached to it and learns the external routes from the redistribution happening on the ASBR.

In an OSPF area, based on the placement of the router, each router assumes different responsibilities and performs various functions. OSPF has four router types:

- *Backbone router:* A backbone router runs OSPF and has at least one interface part of the backbone area or area 0.0.0.0.
- *Internal router:* An internal router has OSPF adjacency only with the devices in the same area. These routers do not form adjacency across multiple areas.

- *Area Border Router (ABR):* An ABR router forms OSPF neighbor adjacency with multiple devices in multiple areas. Because it has adjacency in multiple areas, it maintains a copy of the link-state topology database of multiple areas and distributes it to the backbone area.
- *ASBR:* An ASBR router participates in other routing protocols apart from OSPF and exchanges the routing information learned from other protocols into OSPF and vice versa.

The OSPF routing protocol uses a link-state database (LSDB) that is formed using the information exchanged between all the routers within the area. This information exchange between the routers within the area is done using link-state advertisement (LSA). Instead of exchanging all the network and link information in a single LSA, OSPF uses different types of LSA for different network types. The following is a list of the commonly used LSAs used in OSPF for exchanging various routing updates:

- Router LSA (Type 1)
- Network LSA (Type 2)
- Summary LSA (Type 3)
- Summary ASBR LSA (Type 4)
- AS External LSA (Type 5)
- NSSA LSA (Type 7)

Based on the information in the LSDB, every router in an OSPF area runs the SPF algorithm on all the destination prefixes and installs the route in the routing table. Note that every router in the OSPF area has an identical copy of the LSDB. Based on the understanding of different LSA types, each area type allows for only specific type of LSAs. Table 5-1 displays different LSAs allowed in different area types in OSPF.

Table 5-1. *OSPF Area to LSA Mapping*

Area Type	LSAs Allowed
Backbone area	Type 1, 2, 3, 4, 5
Standard or normal area	Type 1, 2, 3, 4, 5
Stub area	Type 1, 2, 3
Totally stubby area	Type 1, 2, and Type 3 default route
NSSA	Type 1, 2, 3, 7
Totally NSSA	Type 1, 2, 7, and Type 3 default route

Now that we have learned about the basics of the OSPF routing protocol, let's examine the most commonly seen issues in OSPF. The majority of the issues seen in OSPF are neighbor adjacency issues. When two devices form an OSPF adjacency, they can either form the adjacency over these types of networks:

- Broadcast
- Nonbroadcast
- Point-to-point
- Point-to-multipoint

We can focus on broadcast and point-to-point networks because broadcast and nonbroadcast methods both require Designated Router (DR) / Border Designated Router (BDR) election, and point-to-multipoint networks works on the same principle as point-to-point networks. To form a neighbor adjacency, there are different kind of OSPF packets that are

exchanged, including Hello packets, link-state requests, link-state (LS) updates, and LSAs. Any two devices participating in forming neighbor adjacency go through the following states in the finite state machine:

- *Down:* This is the initial state of an OSPF router where no information is exchanged between the routers.
- *Attempt:* This state is similar to the down state, with the difference that the router is in the state of initiating a conversation. This state is only applicable for nonbroadcast multiaccess (NBMA) networks.
- *Init:* – In this state, a Hello packet has been received from the neighbor router, but the two-way communication has not yet been established.
- *2-Way:* Indicates that a bidirectional conversation has been established between two routers. After this state, DR/BDR is elected for broadcast and NBMA networks. A router on a broadcast or NBMA network becomes full with the DR/BDR, but remains in 2-way with all the remaining routers.
- *Exstart:* In this state DR/BDR is established as a master–subordinate relationship. The router with the highest router ID is selected as the master and starts exchanging the link-state information.
- *Exchange:* In this state, the OSPF neighbors exchange database description (DBD) packets. The DBD packets contain LSA headers that describe the contents of the LSDB and are compared with the router's LSDB.
- *Loading:* If there is any discrepancy or missing information found by comparing the DBD packets with the LSDB, routers send link-state request packets to the

neighbor routers. In response to the link-state requests, the neighbor router responds with LS Update packets that are acknowledged by the receiving router using LSA packets.

- *Full:* In this state, the router's database is completely synchronized with the LSDB of the neighbor routers and the routers become fully adjacent.

Let's now look at the Wireshark captures based on the different states. Figure 5-1 displays the initial OSPF Hello packet where an OSPF-enabled router sends out a Hello packet on the 224.0.0.5 multicast address. Because the router has not received any hello back from the other end, there is no information available about the active neighbor.

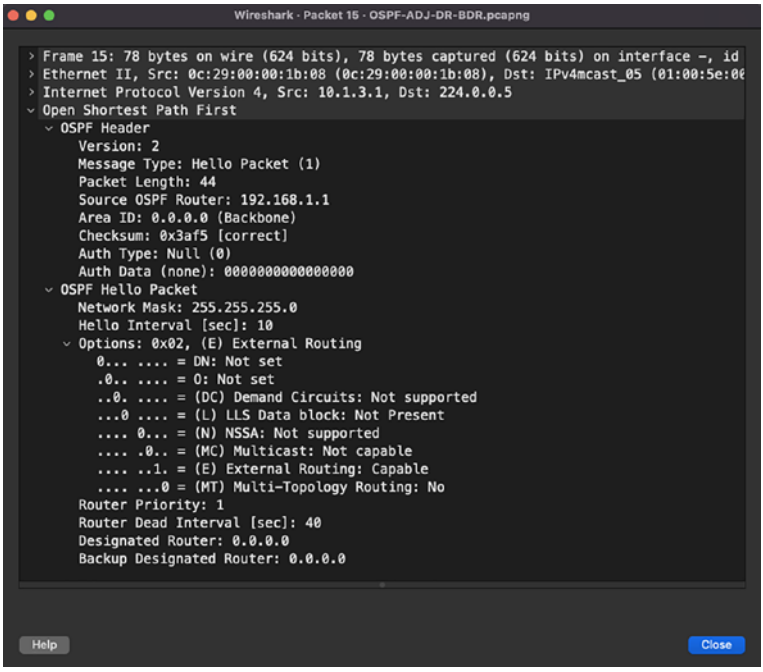


Figure 5-1. *OSPF Hello packet*

Once the OSPF router is able to see the neighbor router, you can then see the Active Neighbor field in the Hello packet. Figure 5-2 displays the active neighbor in the Hello packet for router R3 with OSPF router ID 192.168.3.3. Notice that so far no DR/BDR election has happened in this network segment.

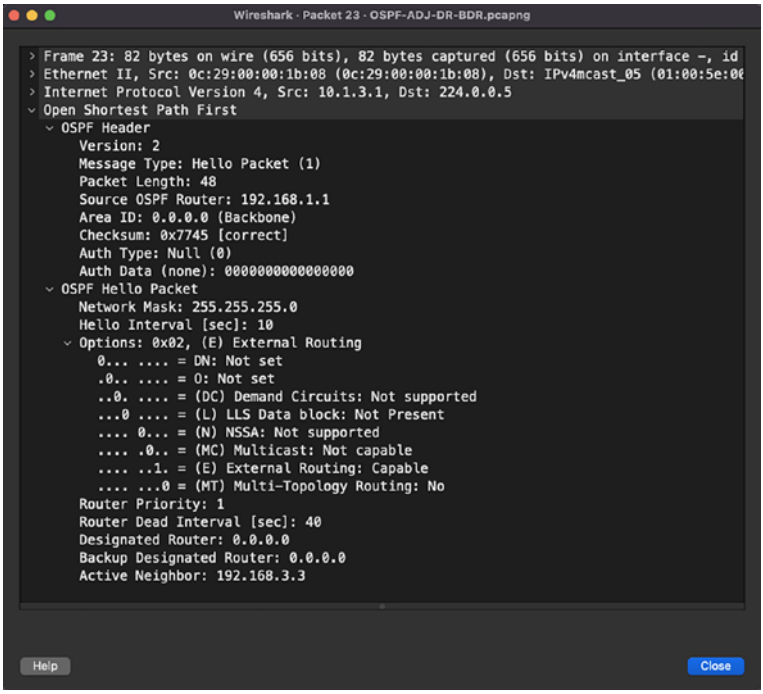


Figure 5-2. *OSPF Hello packet with active neighbor*

Once the OSPF routers negotiate the DR/BDR roles, the Hello packet will then have both the DR and BDR fields populated as shown in Figure 5-3.

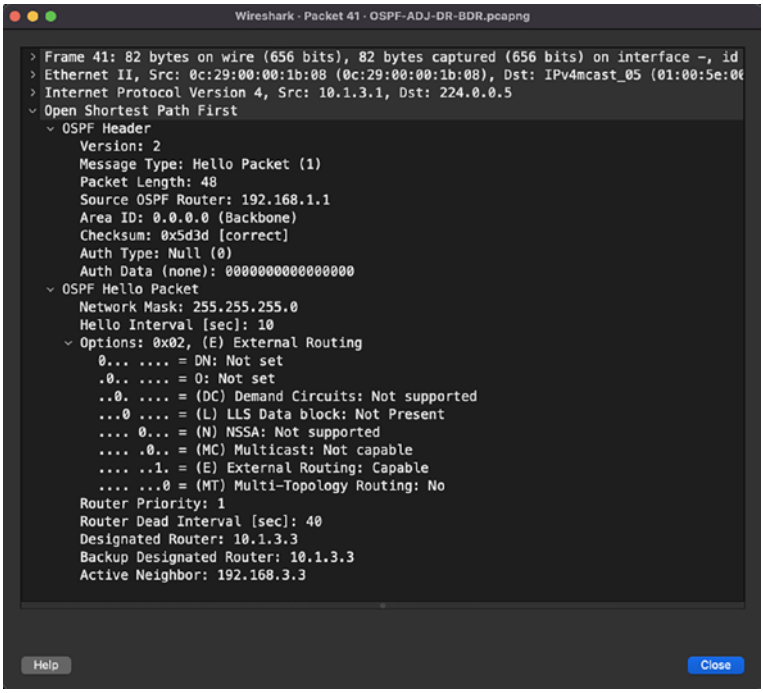


Figure 5-3. *OSPF Hello packet with DR/BDR*

After the DR/BDR election, the routers decide on the master and subordinate election on the segment. Remember that initially both the routers will send the DBD packet with the Master (MS) bit set, but once the OSPF software realizes that the router with the highest router ID is the master, then the router with lowest router ID will not have the MS bit set. Within the OSPF DBD packet, the MTU of the segment is also advertised. Notice that if there is a mismatch of the MTU values, the OSPF neighborship gets stuck in the exstart or exchange state. Once the master and subordinate election is completed, only then will the routers start exchanging the LSA information in the OSPF DBD packets. Figure 5-4 displays the DBD packet with the MS bit set for the packet coming from the router with highest router ID. Also notice the various LSAs being advertised to the neighboring router. Another important thing to

remember is that the Init (I) bit is always set on the initial DBD packet sent by each side of the segment. The More (M) flag is set where there are more DBD packets pending that will be sent by the router.

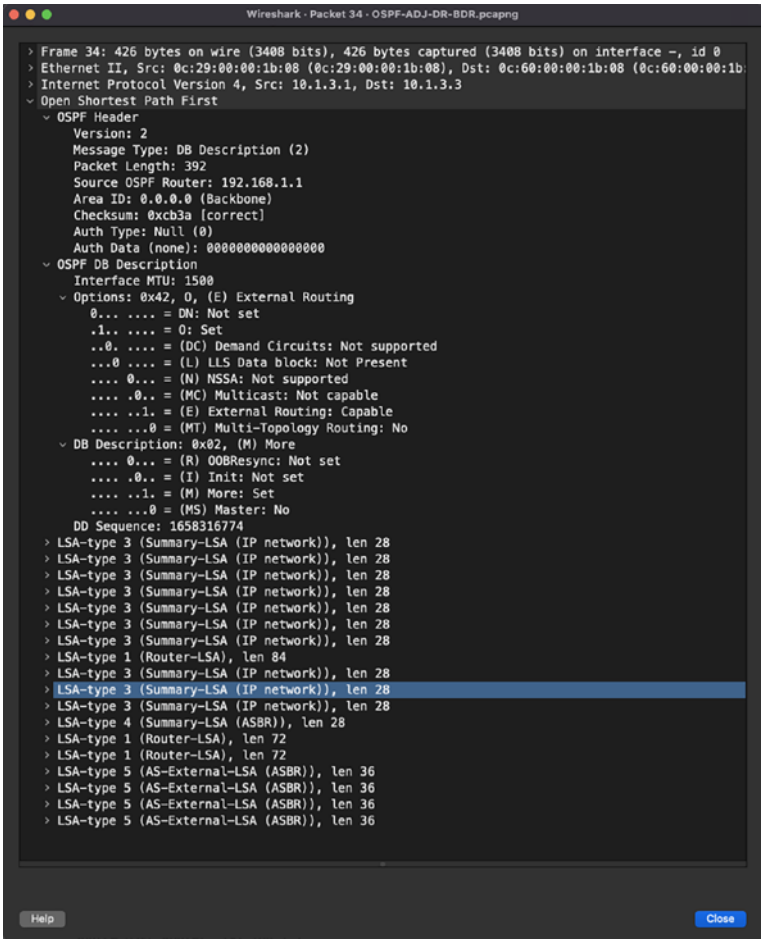


Figure 5-4. *OSPF database description packet*

The LS Update packets are sent between the routers in the segment. When an LS Update is sent by DR, it is sent to destination address 224.0.0.5, whereas BDR sends it over 224.0.0.6. The LS Update packet basically contains the list of LSAs that the OSPF router wants to advertise

to its neighboring device to synchronize the OSPF database. Figure 5-5 displays the Wireshark capture of the OSPF LS Update packet advertising LSA Type 1 and LSA Type 2 to the neighboring router.

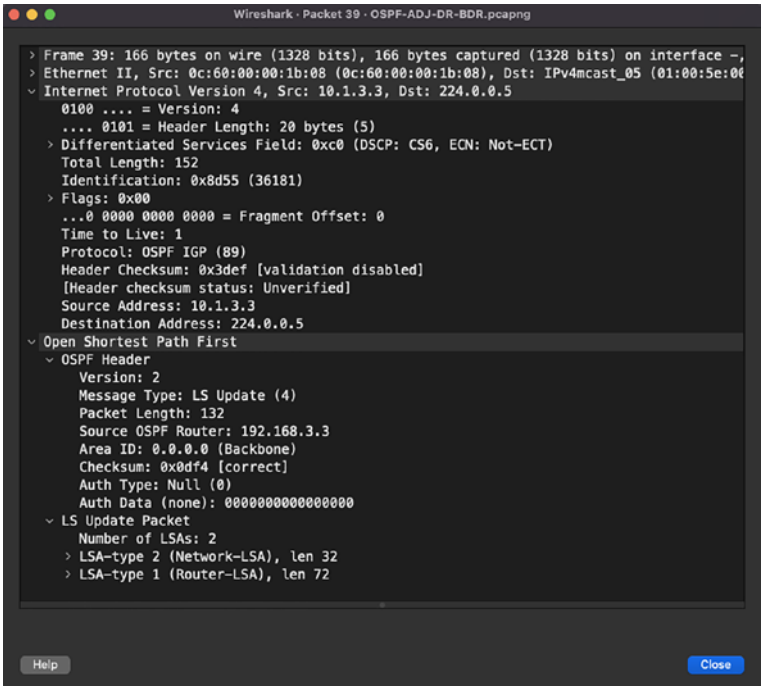


Figure 5-5. *OSPF LS Update packet*

What does an LSA header look like? Each LSA has a common header with 20 bytes followed by a number of additional fields that describe the link. Here are the fields present in the OSPF LSA header:

- *LS Age (2 bytes):* Represents the elapsed time since the LSA was created.
- *Options (1 byte):* Used for advertising OSPF capabilities supported by the router.
- *LS Type (1 byte):* Indicates the type of LSA.

- *Link State ID (4 bytes):* Indicates the link of either the router or the network the link represents.
- *Advertising Router (4 bytes):* Indicates the OSPF router ID of the router originating the LSA.
- *LS Sequence Number (4 bytes):* A sequence number used to detect old or duplicate LSAs.
- *LS Checksum (2 bytes):* Checksum of the LSA, which is used for identifying any data corruption.
- *Length (2 bytes):* Length of the LSA including 20 bytes of the header.

Figure 5-6 displays the Wireshark capture of an LSA Type 3 header within a DBD packet.

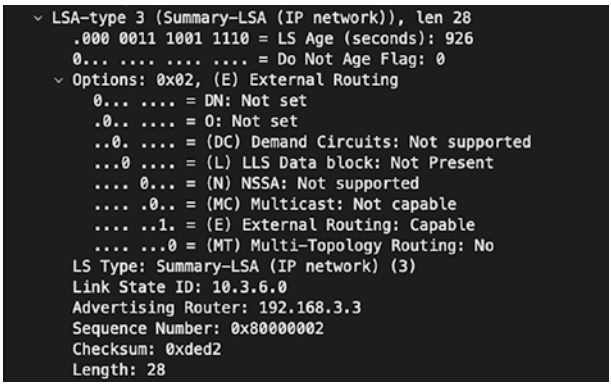


Figure 5-6. *OSPF LSA header*

Based on the LS Update packet, the router sends an LSA packet. Figure 5-7 displays the LSA packet sent by the router R1 in response to the LS Update packet sent by R3 in Figure 5-5.

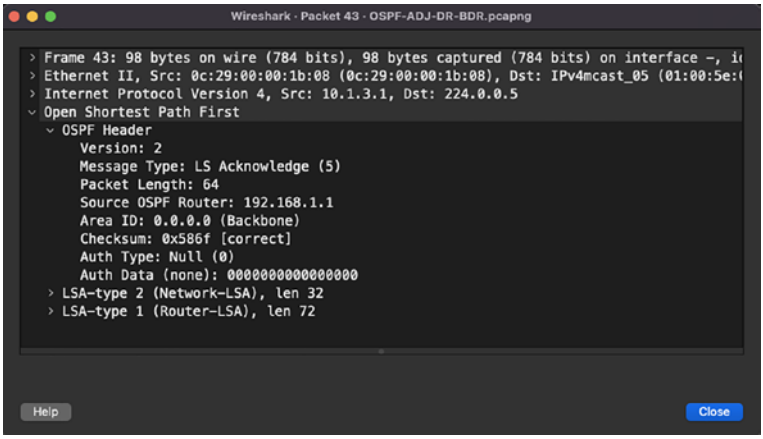


Figure 5-7. *OSPF LSA packet*

Most of the issues in OSPF are usually seen during adjacency bring up, but once the adjacency is up, OSPF remains stable. There might be a bit of difference based on the different OSPF area types, especially with OSPF NSSA. In an OSPF NSSA, the Hello packet has an NSSA (N) bit set, which tells the peering router that it has the NSSA capability enabled on it. Figure 5-8 displays the Hello packet in the NSSA.

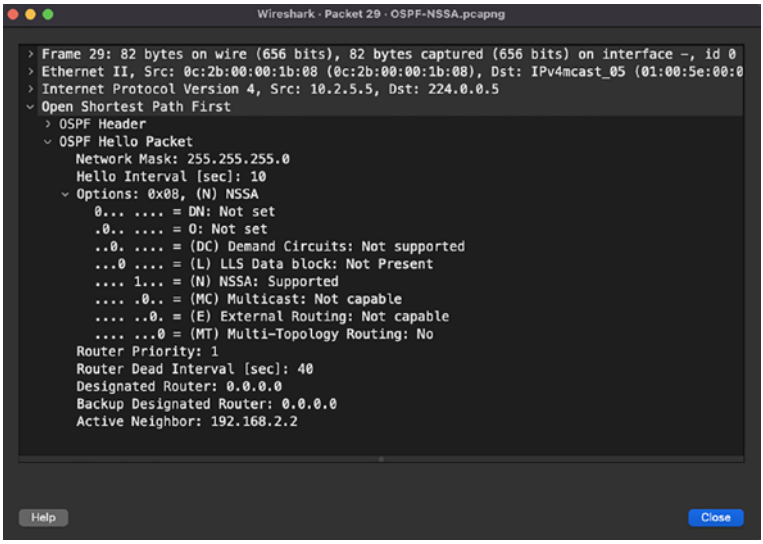


Figure 5-8. *OSPF NSSA Hello packet*

Because NSSA advertises external prefixes as Type 7 LSA and these Type 7 LSAs are converted to Type 5 LSAs at the ABR, the ABR specifically looks for a Propagate (P) bit to ensure the conversion from Type 7 to Type 5 LSA is required. If the P bit is not set, the conversion from Type 7 to Type 5 LSA will not be allowed. Figure 5-9 displays the Wireshark capture of the Type 7 LSA in the LS Update packet with the P bit set.

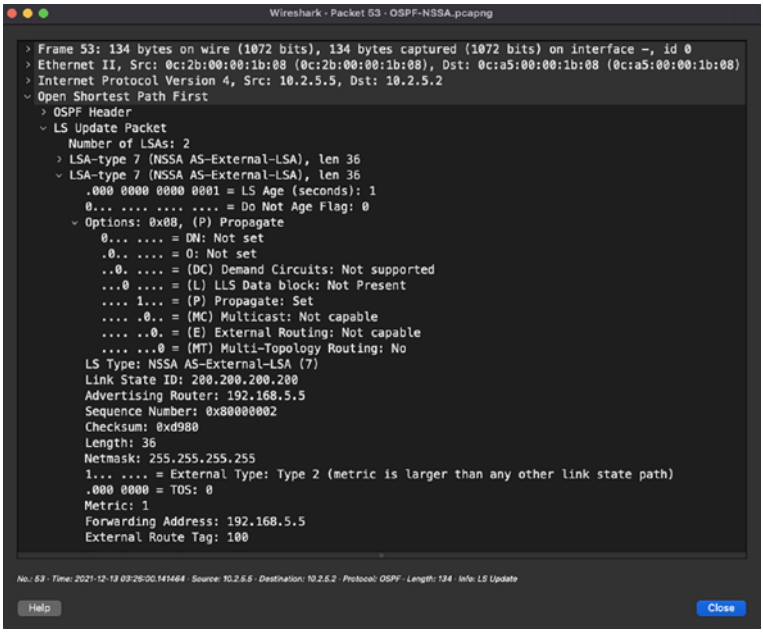


Figure 5-9. *OSPF Type 7 LSA*

Note that most network OSs come with debug capability for various routing protocols that can be enabled on the router for the purpose of troubleshooting, but Wireshark can be helpful in instances when there is a bigger risk of affecting the router in a production environment when running debugs. When working with Wireshark, the filters listed in Table 5-2 can be helpful to filter packets.

Table 5-2. *Wireshark OSPF Filtering*

Filter	Description
`ospf.area_id == 0.0.0.10`	Filters OSPF packets for specified Area ID
`ospf.advrouter == 192.168.5.5`	Filters OSPF packets with the specified router ID of the advertising router
`ospf.hello`	Filters OSPF Hello packets
`ospf.lsa.router`	Filters OSPF router LSA
`ospf.lsa.network`	Filters OSPF network LSA
`ospf.lsa.summary`	Filters OSPF summary LSA
`ospf.lsa.nssa`	Filters for NSSA (Type 7) LSA
`ospf.lsa.asext`	Filters for Type 5 (External) LSA

EIGRP

Enhanced Interior Gateway Routing Protocol (EIGRP), defined in RFC 7868, is another IGP designed and developed by Cisco Systems. It is also known as a distance vector protocol that leverages the Diffusing Update Algorithm (DUAL) to calculate loop-free routing paths using diffusing computations. All routing protocols, including EIGRP, fundamentally work the same way and have similar functions such as these:

- *Establishing communication:* EIGRP uses a three-way handshake for establishing communication.
- *Exchanging routes:* EIGRP uses reliable transport for exchanging routes.
- *Performing path computation:* The procotol leverages the DUAL algorithm to perform path computation.

- Installing routes in the Routing Information Base (RIB): EIGRP only installs loop-free paths in the RIB.

One of the key components of EIGRP is its Topology table. It contains all known paths, locally learned routes, and externally learned routes (learned via redistribution). The information available in the Topology table is used by the DUAL algorithm to calculate the loop-free paths. The EIGRP Topology table not only contains information about the paths, but it also maintains information about when a route was withdrawn by a neighbor.

Most of the computation element resides locally on the router, but EIGRP performs all its tasks using five types of packets:

- Hello
- Update
- Acknowledge
- Query
- Reply

Let's take a closer look at these packets one by one.

Hello Packet

Hello packets are used for peer discovery and maintenance purposes. This packet is the first message sent when the EIGRP process comes up on a router and contains several parameters such as K values, AS numbers that are checked by the peer router on receiving the Hello packet, before forming neighborship. The Hello timer is set to a default of 5-s intervals on high-bandwidth links and 60 s on low-bandwidth links. The Hello packets are usually sent to the multicast address 224.0.0.10 unless the neighbors

are statically configured on a nonbroadcast medium such as a Frame-Relay, in which case they are sent as unicast packets. Figure 5-10 displays the Wireshark capture of EIGRP Hello packets. Note that you can simply filter the EIGRP Hello packets using the filter `(eigrp.opcode == 5) && (eigrp.ack == 0)`. Note that the `eigrp.ack` filter is used to filter out the Acknowledge field because the opcode for both the Hello and Acknowledge fields is the same, but the latter has a nonzero value in the Acknowledge field.

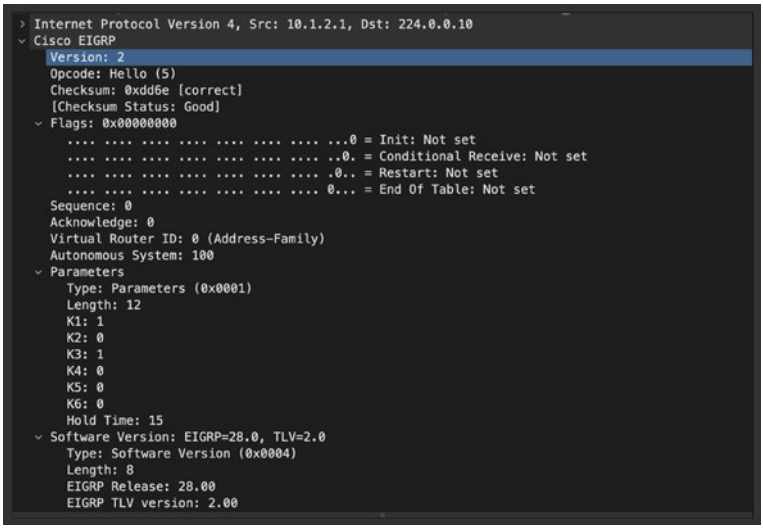


Figure 5-10. *EIGRP Hello packet*

Note The Hello packet also has the Stub flag set when sent by an EIGRP stub router. Users can filter it in Wireshark using the filter `eigrp.stub_flags`.

Update Packet

The Update packets are used by EIGRP to convey reachability information for prefixes to EIGRP neighbors. After an EIGRP neighborship is established, EIGRP routers send Update packets as unicast to the neighbor routers which contains all the routes, also known as the *full updates*. Each route in the update message contains metrics such as bandwidth, delay, load, reliability, and other information such as hop count, MTU, and so on. Once the full updates are exchanged between the EIGRP neighbors, the Update packets are only exchanged when there is a change in topology. For instance, a link flap triggers a withdrawal of multiple routes. This is communicated to EIGRP neighbors via a multicast packet only containing the updates. These updates are called *partial updates*. Figures 5-11 and 5-12 display both the EIGRP full updates and partial updates.

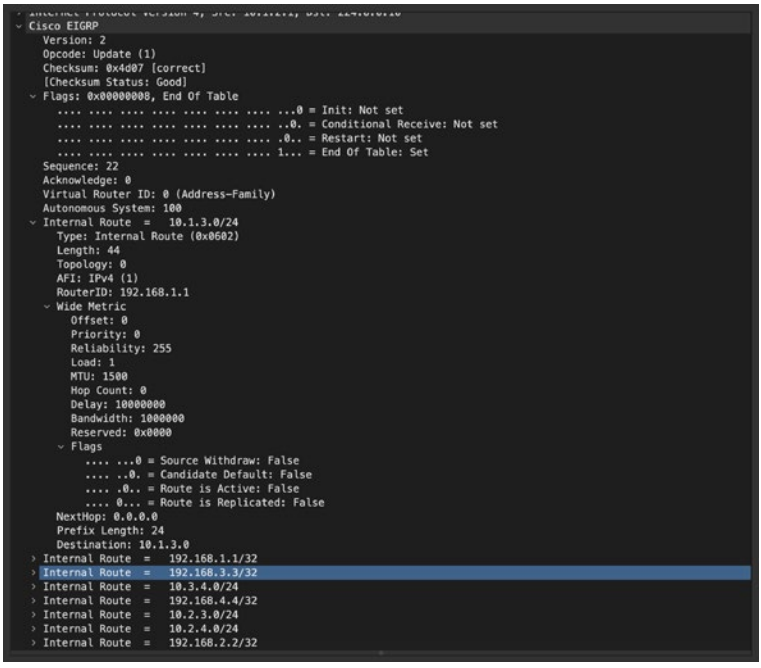


Figure 5-11. *EIGRP full update*

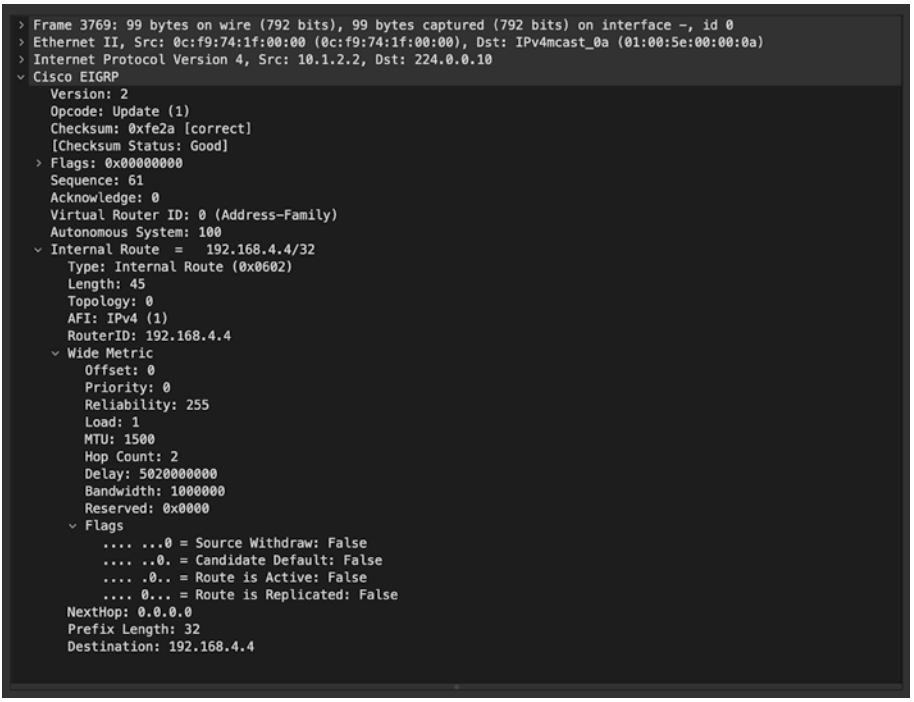


Figure 5-12. *EIGRP partial update*

Users can filter EIGRP update packets in Wireshark using the `eigrp.opcode == 1` filter. This filter displays both full and partial updates in EIGRP.

Acknowledge Packet

EIGRP works similar to the TCP three-way handshake, where the initial packet could be an Update, Query, or Reply packet and in acknowledgment to these packets, an Acknowledge packet is sent by the EIGRP router. The difference between the TCP and EIGRP three-way handshake is that the sequence number in EIGRP is not incremented but rather copied in the Acknowledge field. Also, this whole communication is done by Cisco's proprietary Reliable Transport Protocol (RTP). The EIGRP

Acknowledge packet has the same opcode as the EIGRP Hello packet, but with a nonzero Acknowledge field value. Figure 5-13 displays the Wireshark capture of an EIGRP Acknowledge packet.

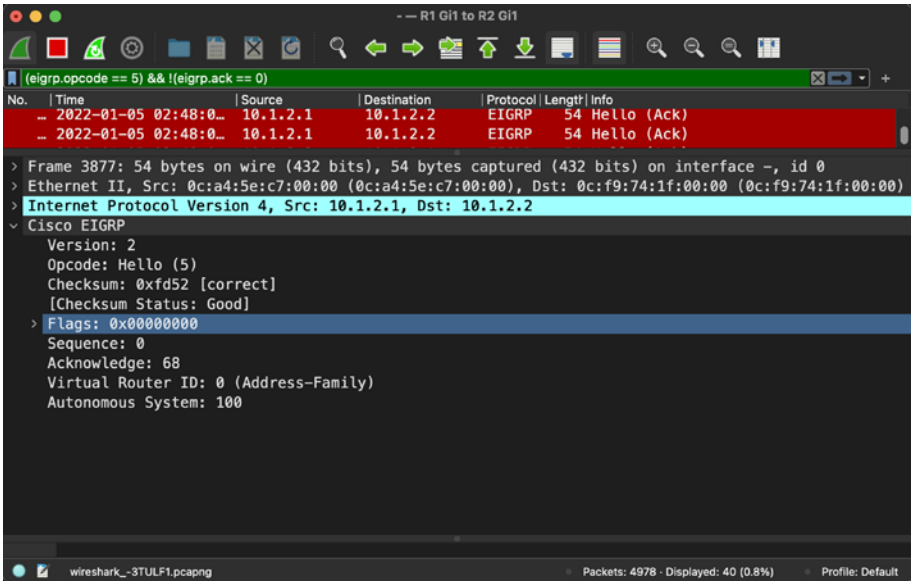


Figure 5-13. *EIGRP Acknowledge packet*

You can filter the EIGRP Acknowledge packet by using the Wireshark filter `(eigrp.opcode == 5_ && !(eigrp.ack == 0)`. The ! operator ensures that we only capture the Acknowledge packet and not the EIGRP Hello packet.

Query Packet

EIGRP queries are sent when a router loses a route to a destination network (the destination prefix goes into active state). Queries are normally sent as multicast to all the neighboring routers to find other paths to the destination prefix. If a receiving router cannot find an alternate path to the destination prefix, it will then query its peers for the

destination prefix. This process goes on until the query has reached the boundary router. Figure 5-14 displays the EIGRP Query packet for the destination prefix.

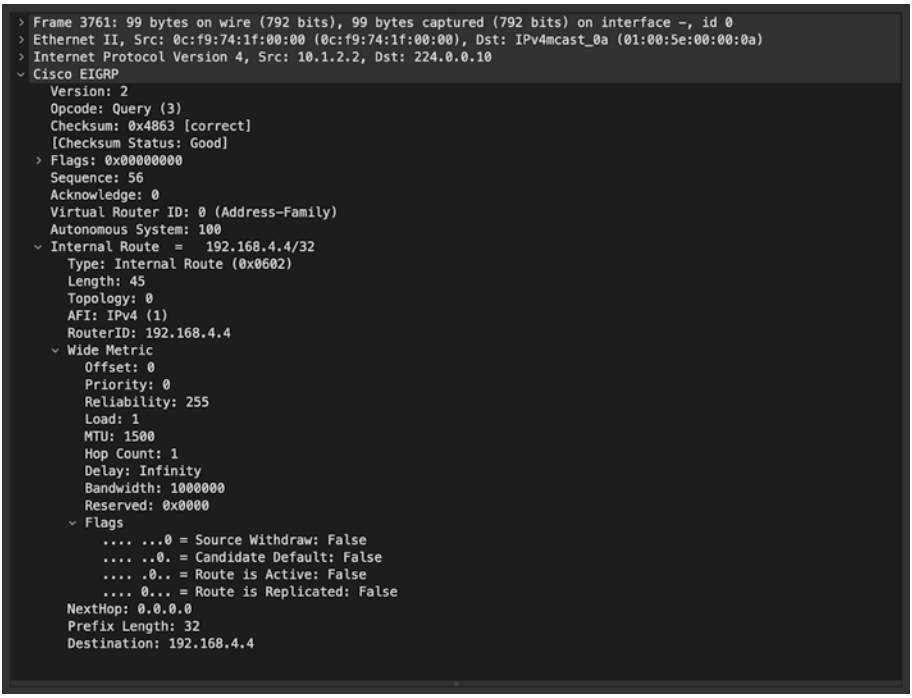


Figure 5-14. *EIGRP Query packet*

You can filter the EIGRP Query packet in Wireshark using the filter `eigrp.opcode == 3`.

Note EIGRP Query packets are not sent to stub routers.

Reply Packet

The EIGRP Reply packet is sent in response to the Query packet. After sending the Query packet, a router waits for a reply from its peer routers. If a router receiving the Query packet knows about an alternate path to the destination prefix, it will respond back to the querying router with the necessary metrics to reach the destination prefix. Figure 5-15 displays the Wireshark capture of an EIGRP Reply packet. You can filter the Reply packet using the Wireshark filter `eigrp.opcode == 4`.

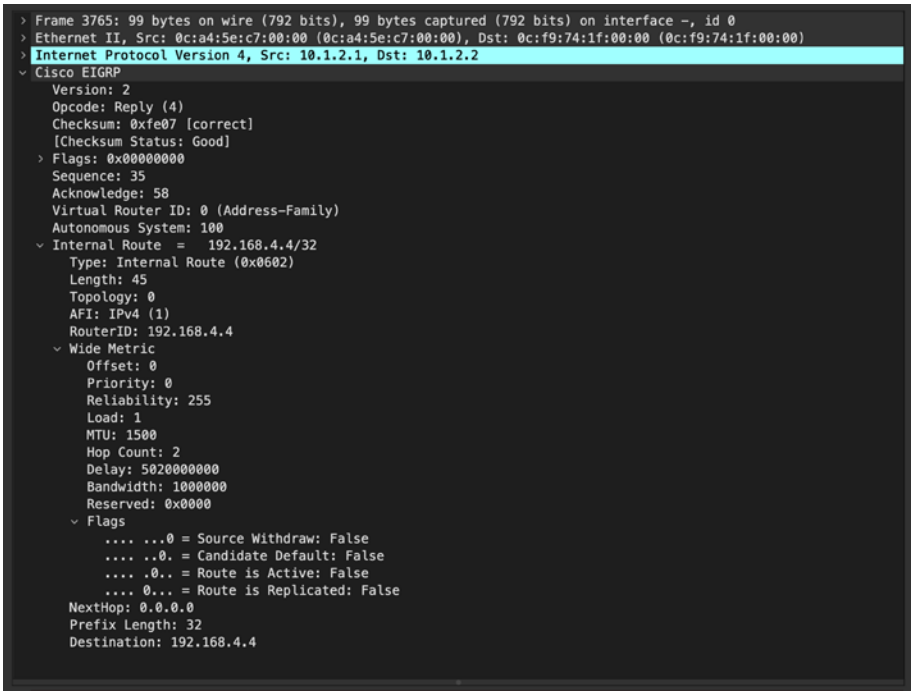


Figure 5-15. *EIGRP Reply packet*

BGP

BGP, often called the routing protocol of the Internet, is an open standard protocol used for connecting network across different AS boundaries. BGP is a highly scalable protocol and has support for multiple address families such as IPv4, IPv6, VPNv4, L2VPN, EVPN, and so on, which allows BGP to be the protocol of choice in enterprise, datacenter, and service provider environments. BGP, in general, cannot route traffic on its own. It leverages the information from IGP to reach to the next hop for the prefix. BGP knows about the prefixes that might be within the same AS boundary or across multiple AS boundaries. BGP only knows about next hops to reach the destination, but it needs IGP to get to that next hop.

Because BGP exchanges information across AS boundaries, it is also important that the information is exchanged via a reliable mechanism. Thus, BGP leverages TCP as its transport mechanism. A BGP session is established on TCP port 179. In BGP, two types of neighborships can be established:

- *Internet BGP (iBGP):* BGP peering established with other routers within the same AS boundary.
- *External BGP (eBGP):* BGP peering established with routers across AS boundaries.

For two routers to establish a BGP peering, they go through a finite state machine as listed here:

- *Idle:* In this state, BGP detects a start event and initializes the BGP resources on the router. The BGP process initiates a TCP connection toward the peer.
- *Connect:* In this state, BGP waits for the three-way handshake to complete. If the three-way handshake is successful, an OPEN message is sent and the BGP

process moves to the OpenSent state. If it is not successful, BGP moves to the Active state, and waits for a ConnectRetry timer.

- *Active:* BGP starts a new three-way handshake. If the connection is successful, it moves to OpenSent state. If it is unsuccessful, the BGP process moves back to the Connect state.
- *OpenSent:* In this state, the BGP process sends an OPEN message to the remote peer and waits for an OPEN message from the peer.
- *OpenConfirm:* In this state, the router has already received the OPEN message from the remote peer and is now waiting for a KEEPALIVE or NOTIFICATION message. On receiving the KEEPALIVE message, the BGP session is established. On receiving a NOTIFICATION message, BGP moves to the Idle state.
- *Established:* This state indicates that the BGP session is established and is now ready to exchange routing updates via the BGP UPDATE message.

From this finite state machine, we have already learned that there are four types of BGP messages:

- *OPEN:* This is the first message exchanged between BGP peers after a three-way handshake has been established between the peers. Once each side confirms the information shared in the BGP OPEN message, other messages are exchanged between them. The following information is compared as part of the OPEN message:
  - BGP version

    - Source IP of the OPEN message should match with configured peer IP
    - Received AS number should match the configured remote AS number of the BGP peer
    - BGP Router ID must be unique
    - Other security parameters such as password, TTL, and so on
- *KEEPALIVE:* The BGP KEEPALIVE message acts like a Hello packet to check whether the BGP peer is alive or not. This message is used to keep sessions from expiring.
- *NOTIFICATION:* BGP NOTIFICATION is sent when the BGP process encounters an error condition. When this message is received, the BGP process closes the active session for which the notification was received. The NOTIFICATION message also contains the information such as error code and suberror code that can be used to determine the cause of the error condition.
- *UPDATE:* This message is used for exchanging routing updates (advertisements and withdrawals) between BGP peers.

We'll now examine these BGP messages in Wireshark. First for the BGP OPEN message, the following fields are present in the header:

- *Marker:* Set to `ffffffffffffffffffffffff`.
- *Length:* Length of the BGP header
- *Type (OPEN message):* Value set to 1.

- *Version:* Specifies the current BGP version used by the router. The current version is 4 as defined in RFC 4271.
- *MS AS:* Specifies the AS number of the router originating the OPEN message.
- *Hold Time:* Specifies the Hold Timer value set on the router sending the OPEN message.
- *BGP Identifier:* Router ID of the router sending the OPEN message.
- *Optional Parameters Length:* Variable length, specifies the combined length of all the parameters included in the Optional Parameters field.
- *Optional Parameters:* This field is used by the router to advertise optional BGP capabilities that are supported in BGP by the OS running on the advertising router. Some of these capabilities include the following:
    - Multiprotocol BGP (MP-BGP) support
    - Route Refresh support
    - 4-octet (4-byte) AS number support

Figure 5-16 displays the BGP OPEN message.

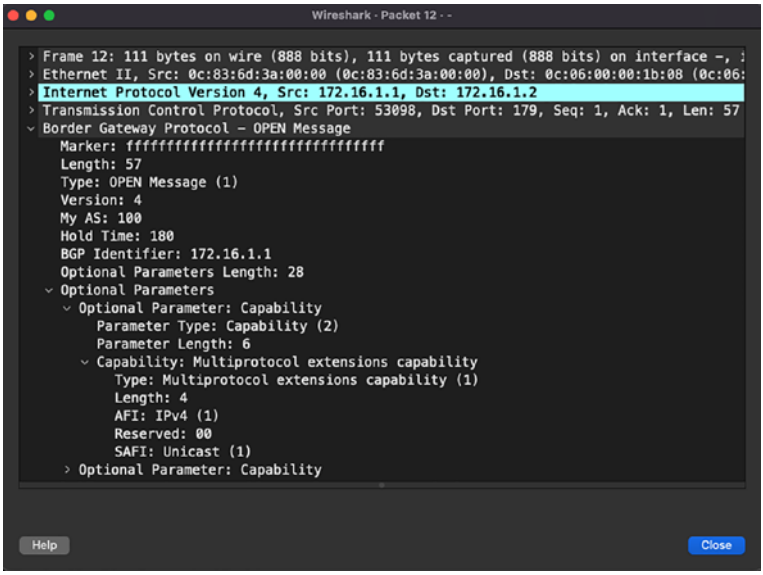


Figure 5-16. *BGP OPEN message*

The BGP KEEPALIVE message, as mentioned before, is used to ensure BGP peers are active. The BGP process does not rely on the TCP connection to validate that the BGP peers are up. BGP KEEPALIVE messages are sent every 60 s by default with the Hold Timer set to 180 s. Figure 5-17 displays the Wireshark capture of a BGP KEEPALIVE message sent between BGP peers. We can see from the Wireshark capture that there are only three fields present in the BGP KEEPALIVE message.

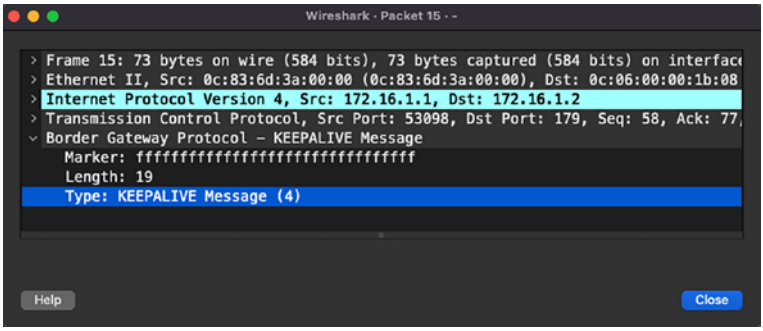


Figure 5-17. *BGP KEEPALIVE message*

A BGP NOTIFICATION message is also a short message that contains the information about error code (major error code) and suberror code (minor error code). Because BGP peering may be established multiple hops away, BGP provides a mechanism to notify other peers about what might have triggered the error condition, causing the BGP peering to flap. Figure 5-18 displays the Wireshark capture of the BGP NOTIFICATION message. In the Wireshark capture we can see that the error code is 6 and the suberror code is 4, which indicates Administratively Reset. Thus, this notification message indicates that the BGP peering was manually reset.

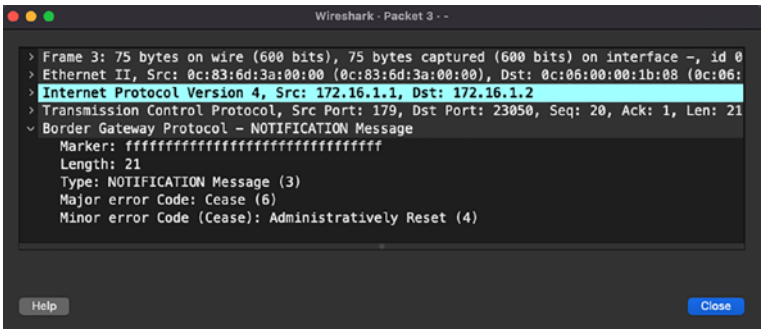


Figure 5-18. *BGP NOTIFICATION message*

Before we dive into the BGP UPDATE message, let's first understand how the BGP update packaging happens. Once the initial TCP session is established, both endpoints maintain the information about the TCP MSS. As mentioned in Chapter 3, MSS = MTU – IP Header (20 bytes) – TCP Header (20 bytes). When the BGP process wants to send updates to its BGP peers, it packages all the updates to a maximum of MSS bytes and sends it across to the remote BGP peer with the Don't Fragment (DF) bit set. If all the updates cannot be sent in one single update, BGP sends multiple updates to update the remote BGP peers. It is possible that if any of the segments has lower interface MTU or IP MTU settings, but the MSS negotiation happened with a higher value, the BGP updates might not be able to make it to the remote BGP peer. When the BGP UPDATE packet is sent, the BGP process does not send a BGP KEEPALIVE message. It treats the BGP UPDATE packet as the BGP KEEPALIVE message and the acknowledgment of the BGP UPDATE packet as an acknowledgment to the KEEPALIVE message. Therefore, if the BGP UPDATE packet is unable to make it to the remote end, the BGP session will flap due to Hold Timer expiry. Figure 5-19 displays the Wireshark capture of the BGP UPDATE packet. Notice that in the IP header, the DF bit is set, and in the BGP header, we can see the BGP UPDATE message. The BGP UPDATE message contains the attributes attached to the BGP prefixes and the BGP prefixes are listed as Network Layer Reachability Information (NLRI).

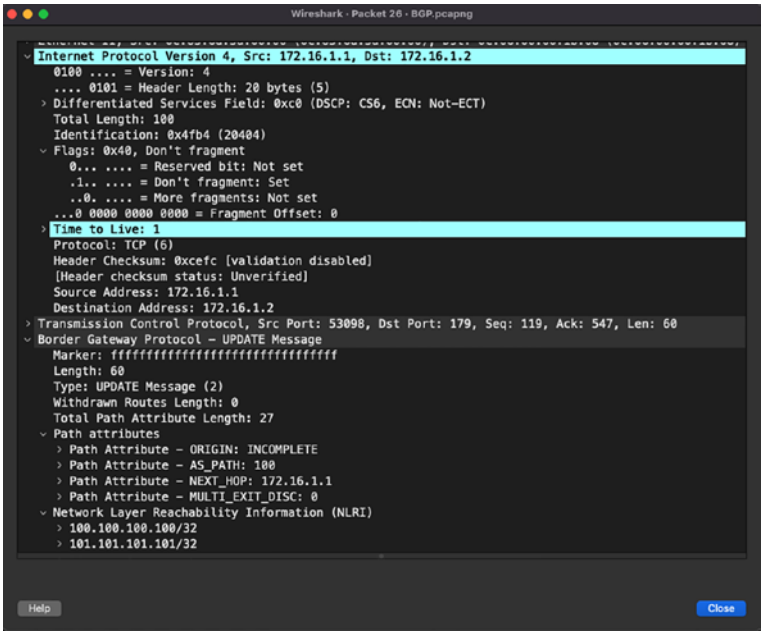


Figure 5-19. *BGP UPDATE message*

Most BGP issues can be investigated from the CLI. You might only need to leverage the help of Wireshark when there is an issue with the following:

- TCP session
- Packet loss
- Network OS not generating packets in a timely manner
- Device not sending the BGP packets out in a timely manner
- BGP updates getting corrupted

PIM

Today, almost every network uses multicast in one way or the other. Multicast allows for one-to-many traffic, but only to those who have subscribed or are interested in that traffic. Multicast applications have wide implementation and use cases in financial, health care, digital streaming, and many other types of organizations. Before we dive into multicast and routing protocols to carry PIM-related traffic, we need to understand some key terms:

- *Source address:* Unicast address of a multicast source or sender.
- *Group address:* Destination IP address of a multicast group. Note that multicast addresses range from 224.0.0.0 to 239.255.255.255.
- *Multicast distribution tree (MDT):* Multicast flows from source to receivers over an MDT. The MDT is either shared or dedicated based on the multicast implementation
- *Rendezvous point (RP):* A multicast-enabled router that acts as the root of the shared MDT.
- *Protocol Independent Multicast (PIM):* Routing protocol used to create MDTs.
- *First-hop router (FHR):* First L3 hop that is directly adjacent to the multicast source.
- *Last-hop router (LHR):* First L3 hop that is directly adjacent to the receivers.

In this chapter, we focus on the PIM protocol and its messages and see how it is used to build MDT.

The PIM protocol is used to build shared trees as well as shortest path trees from source to receivers to facilitate the distribution of multicast traffic. The PIM protocol runs over the L3 network and builds an overlay network for multicast using the information from the underlying IGP. Thus, when troubleshooting multicast issues, it is important to validate the unicast routing information learned via the IGP. With the help of IGP, PIM is able to locate where the source, receiver, and the RP resides. PIM operates in two modes:

- *Dense mode:* – Based on a push model, PIM Dense mode operates under the assumption that receivers are densely dispersed through the network. In this mode, multicast traffic is flooded domain-wide to build a shortest path tree, and the branches are pruned back where no receivers are found.
- *Sparse mode:* Based on a pull model, PIM Sparse mode assumes that the receivers are sparsely dispersed. In this mode, PIM neighbors are formed and traffic is forwarded only over the PIM-enabled path. Using PIM messages, the join request from receivers is forwarded to the RP and thus the mechanism is known as explicit join. Because of this method, it is also the most preferred and widely used method for multicast distribution.

The PIM protocol has the following fields in its header:

- *PIM Version (4 bits):* Version number is set to 2.
- *Type (4 bits):* Used to specify the PIM message type.
- *Reserved (8 bits):* Reserved for future use. The value is set to 0 in this field during transmission and is ignored by the PIM neighbor.

- *Checksum (16 bits):* Used to calculate the checksum of the entire PIM message except for the payload section.

There are multiple PIM message types, but not all messages are used in all deployments. Some of the most commonly seen PIM messages in basic multicast deployment are listed in Table 5-3.

Table 5-3. *PIM message types*

Type	Message Type	Destination Address	Description
0	Hello	224.0.0.13	Neighbor discovery.
1	Register	Address of RP (unicast)	Register message is sent by FHR to RP to register the source.
2	Register-stop	Address of FHR (unicast)	This message is sent by RP to the FHR in response to the PIM Register message.
3	Join/Prune	224.0.0.13	Join or prune from an MDT.

PIM Hello Message

The PIM Hello message, identified with Type 0, is sent on all PIM-enabled interfaces to discover and form PIM neighbor adjacencies. PIM neighborship is unidirectional in nature, so it is important to validate the PIM neighborship from both ends of the link. The PIM Hello messages are sent periodically and with the destination address of 224.0.0.13. A PIM Hello message allows for multiple options in Type, Length, and Value (TLV) format. The options supported in PIM Hello messages are listed in Table 5-4.

Table 5-4. *PIM Hello Message Options*

Option Type	Option Value
1	Holdtime: The amount of time in which the neighbor is in a reachable state
2	Has the following parts: • LAN Prune Delay: Delay before transmitting Prune message in a shared LAN segment • Interval: Time interval for overriding a Prune message • T: Join message suppression capability
19	DR priority used during DR election
20	Generation ID: Random number indicating neighbor status
24	Address List: used for informing neighbors about secondary IP address on interface

Figure 5-20 displays the PIM Hello message between two PIM neighbors.

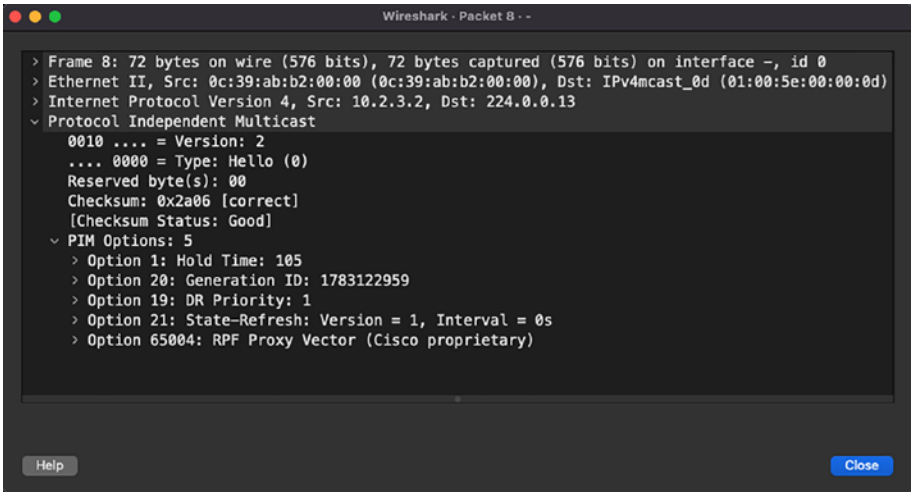


Figure 5-20. *PIM Hello message*

PIM Register Message

When the source sends multicast traffic, the FHR's PIM DR takes the first packet, encapsulates it with the PIM header, and sends it as a unicast packet to the PIM RP. The PIM Register message is used to inform the PIM RP that the source is actively sending traffic for the given multicast group. The PIM Register message contains the following fields in its header:

- *Type:* Value is set to 1 for Register message.
- *Border (B-bit):* The PIM multicast border router functionality is defined in RFC 4601, which designates a local source when this bit is set to 0 and designates the source in a directly connected cloud when this bit is set to 1.
- *Null-Register:* This bit is set to 1 when a Null-Register message is sent. In the Null-Register message, the FHR only encapsulates the header from the source and not the complete encapsulated data packet of the multicast stream coming from the source.
- *Multicast Data packet:* The original multicast packet sent by the source is encapsulated inside the PIM Register message. If the message is a Null-Register message, only a dummy IP header containing the source and group address is encapsulated in the PIM Register message. Note that the TTL of the original packet decrements before encapsulation into the PIM Register message.

Figure 5-21 displays the Wireshark capture of the PIM Register message sent by the FHR to the RP (192.168.3.3).

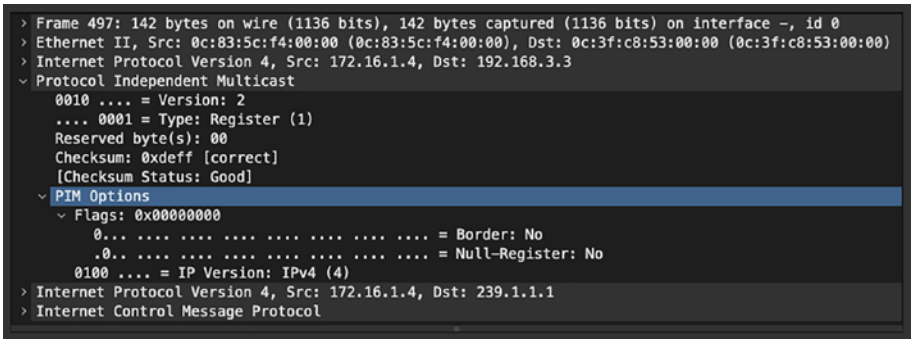


Figure 5-21. *PIM Register message*

PIM Register-Stop Message

On receiving the PIM Register message, the RP adds the source to the multicast distribution tree. Once the RP receives the first packet natively through the shortest path, it will send a PIM Register-stop message to the DR that has built the Shortest Path Tree (SPT) toward the source. The PIM Register-stop message has the following fields:

- *Type:* Value is set to 2 for PIM Register-stop message.
- *Group Address:* Group address of the encapsulated multicast packet in the PIM Register message.
- *Source Address:* Source address of the encapsulated multicast packet in the PIM Register message.

Figure 5-22 displays the Wireshark capture of the PIM Register-stop message from RP to the DR that sent the PIM Register message.

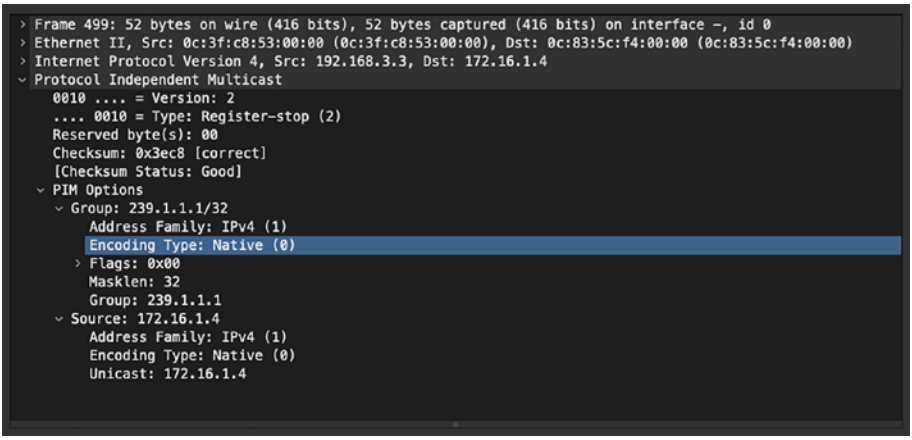


Figure 5-22. *PIM Register-stop message*

PIM Join/Prune Message

The PIM Join/Prune message is sent by PIM routers toward the PIM RP or toward the source with the destination set to PIM multicast address 224.0.0.13. These messages are used to build RP trees (RPTs) toward the PIM RP or to build SPT toward the source. The PIM Join/Prune message contains a list of sources (called *source lists*) and groups (called *group sets*) to be joined or pruned. The following fields are present in the PIM Join/Prune message:

- *Type:* Value is set to 3 for Join/Prune message.
- *Upstream Address:* Address of the upstream neighbor to which the message is targeted. It also has subfields that represent the address family of the upstream neighbor as well as the encoding.

- *Number of Groups:* Represents the number of multicast group sets in the message.
- *Holdtime:* The amount of time to keep the Join/Prune state alive.
- *Num Joins:* Number of joined sources in the message.
- *Joined Source Address {IP Address x.x.x.x/32}*
    - Sparse bit (S): Set to 1 for PIM Sparse mode.
    - Wildcard bit (W): When set to 1, this represents wildcard a in the (*, G) entry. When set to 0, this indicates that the encoded source address for (S, G) entry.
    - RP bit (R): When set to 0, join is sent toward source. When set to 1, join is sent toward RP.
- *Num Prunes:* Number of pruned sources in the message.
- *Pruned Source Address {IP Address x.x.x.x/32}:* Represents the list of sources being pruned for the group. All three flags in Joined Source Address are applicable for Pruned Source Address, too.

The PIM Join message is sent by the LHR's DR toward the RP whenever a receiver shows an interest in receiving a multicast stream. Figure 5-23 displays the Wireshark capture of the PIM Join message from the FHR toward the RP.

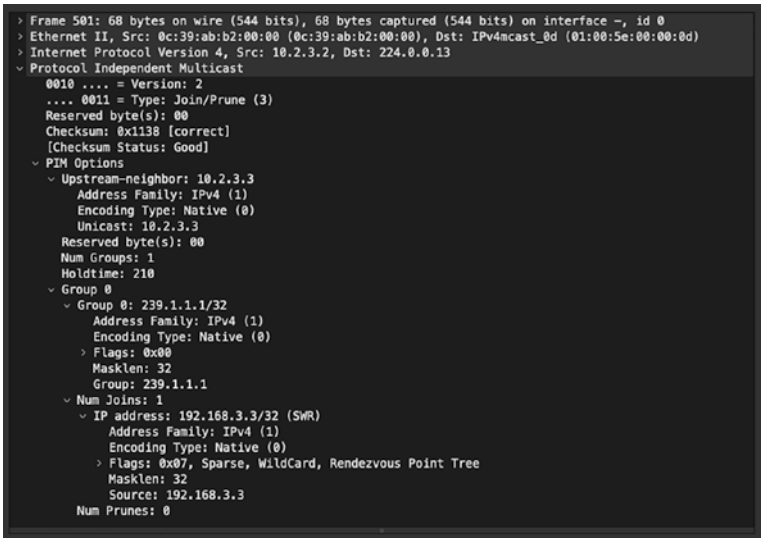


Figure 5-23. *PIM Join message*

A PIM Prune message is sent by a PIM router when it wants to remove itself from the multicast tree for a particular multicast group. Figure 5-24 displays the Wireshark capture of a PIM Prune message when there is no receiver interested in the multicast stream.

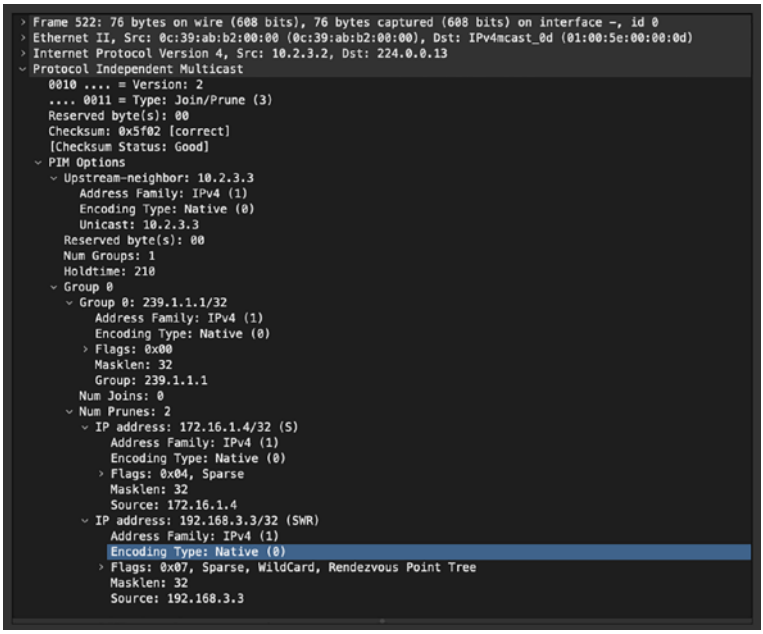


Figure 5-24. *PIM Prune message*

Analyzing Overlay Traffic

So far, we have learned about analyzing routing protocol traffic that can run on physical links or virtual links such as SVIs. Such networks are known as underlay networks. The routing protocols, however, can also run over an overlay network. An overlay network is a network that is built on top of another network and leverages underlying network configuration and protocols to establish communication as if they were locally connected. The devices or endpoints in an overlay network could be residing multiple hops away in the same or a different geographical location. In overlay traffic, the actual host traffic is encapsulated with the headers of the underlay network. We next look at different overlay protocols and how we can analyze the overlay traffic using Wireshark.

GRE

Generic Routing Encapsulation (GRE), defined in RFC 2784, is an overlay protocol that allows users to create virtual point-to-point links and encapsulate the data packets in a tunnel interface. Because it creates a point-to-point link, each side can encapsulate any outgoing packets toward the remote end and de-encapsulate any incoming packets from the far end of the tunnel. With GRE, users might be running a different routing protocol in the underlay to establish the reachability between the two endpoints of the tunnel while running a different routing protocol in overlay to establish end-to-end connectivity of hosts and devices sitting behind the tunnel endpoints. Figure 5-25 displays the Wireshark capture of the GRE encapsulated packet. Notice that GRE is a 4-byte header, but there is also an overhead of 20-byte outer IP header after the encapsulation. Thus, we need to make sure that the IP MTU value is adjusted accordingly when encapsulating traffic with GRE.

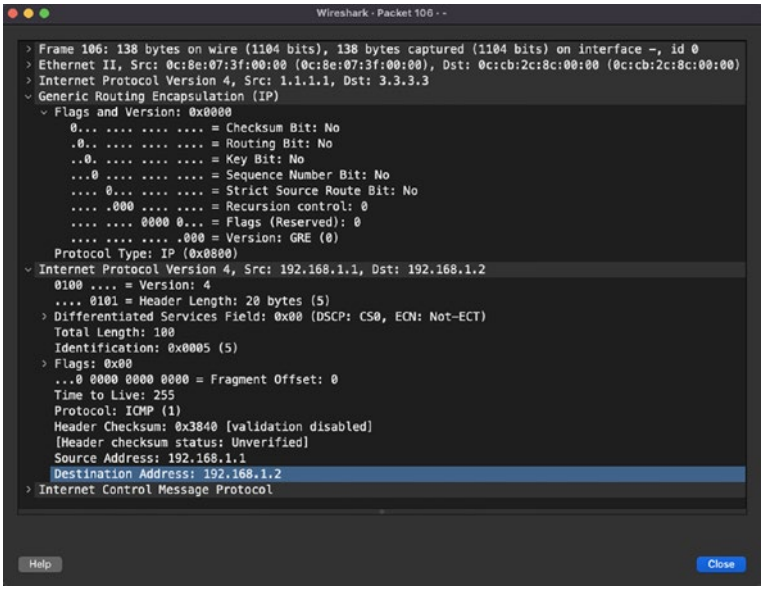


Figure 5-25. *GRE encapsulation*

When data traffic is GRE encapsulated, the TTL value in the outer IP header decrements but does not in the inner IP header. Figure 5-26 displays the Wireshark capture of GRE encapsulated traffic captured after the first Layer 3 hop. Notice that the outer IP header has a TTL value of 254, whereas the inner IP header (with source IP set to 192.168.1.1 and destination IP set to 192.168.2.2) has a TTL value of 255.

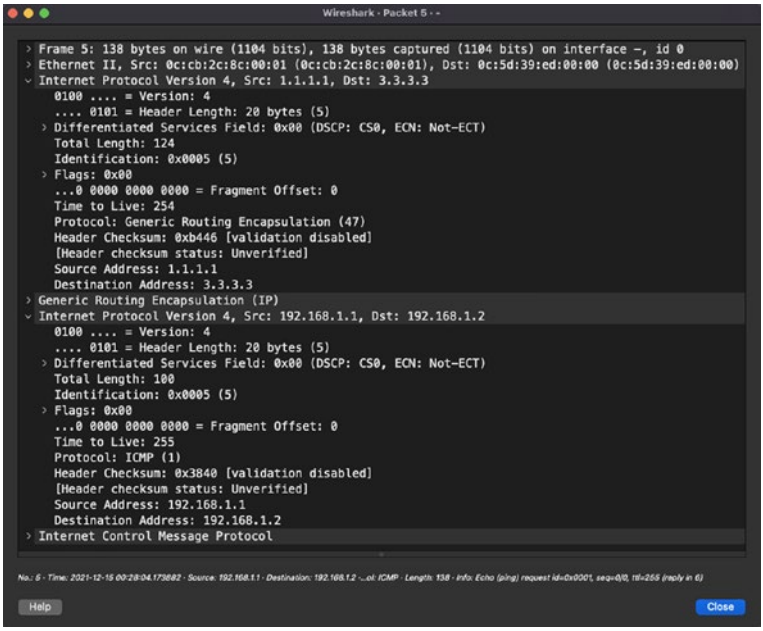


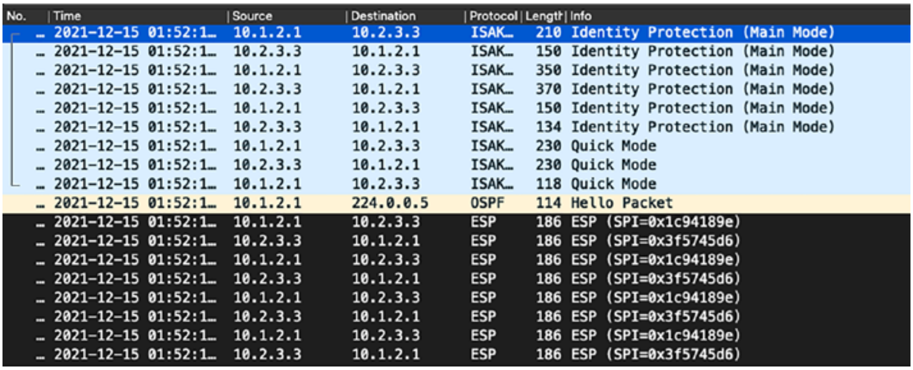
Figure 5-26. *GRE encapsulated traffic after first Layer 3 hop*

IPSec

IP Security (IPSec), defined in RFC 1825 through RFC 1827, is a suite of protocols to establish secure communication between two endpoints across the IP network that provides authentication, data integrity, and confidentiality. The RFC also defines the protocols needed for secure key exchange and key management. The following protocols are part of the IPSec protocol suite:

- *Authentication Headers (AH):* AH provides data integrity, authentication, and antireplay capabilities, which protects against unauthorized transmission of packets.
- *Internet Key Exchange (IKE):* – IKE is a network security protocol that defines how to dynamically exchange encryption keys and use Security Associations (SAs) to establish shared security attributes between the two IPSec tunnel endpoints. The Internet Security Association Key Management Protocol (ISAKMP) provides a framework for authentication and key exchange and defines how to setup SAs. There are two versions of IKE:
  - IKEv1
  - IKEv2
- *Encapsulating Security Payload (ESP):* ESP provides authentication for the payload or data. It ensures data integrity, encryption, and authentication and prevents any replay attacks on the payload.

Let's now look at the negotiation for IKEv1 in Wireshark. Figure 5-27 displays the Wireshark capture of all the initial communication between the two routers participating in IPSec IKEv1 negotiations and then transmitting data after a secure communication has been established. From the Wireshark capture we can see there are six Main mode messages as part of Phase 1 that negotiate security parameters to protect the next three Quick mode messages as part of Phase 2.



No.	Time	Source	Destination	Protocol	Length	Info
…	2021-12-15 01:52:1…	10.1.2.1	10.2.3.3	ISAK…	210	Identity Protection (Main Mode)
…	2021-12-15 01:52:1…	10.2.3.3	10.1.2.1	ISAK…	150	Identity Protection (Main Mode)
…	2021-12-15 01:52:1…	10.1.2.1	10.2.3.3	ISAK…	350	Identity Protection (Main Mode)
…	2021-12-15 01:52:1…	10.2.3.3	10.1.2.1	ISAK…	370	Identity Protection (Main Mode)
…	2021-12-15 01:52:1…	10.1.2.1	10.2.3.3	ISAK…	150	Identity Protection (Main Mode)
…	2021-12-15 01:52:1…	10.2.3.3	10.1.2.1	ISAK…	134	Identity Protection (Main Mode)
…	2021-12-15 01:52:1…	10.1.2.1	10.2.3.3	ISAK…	230	Quick Mode
…	2021-12-15 01:52:1…	10.2.3.3	10.1.2.1	ISAK…	230	Quick Mode
…	2021-12-15 01:52:1…	10.1.2.1	10.2.3.3	ISAK…	118	Quick Mode
…	2021-12-15 01:52:1…	10.1.2.1	224.0.0.5	OSPF	114	Hello Packet
…	2021-12-15 01:52:1…	10.1.2.1	10.2.3.3	ESP	186	ESP (SPI=0x1c94189e)
…	2021-12-15 01:52:1…	10.2.3.3	10.1.2.1	ESP	186	ESP (SPI=0x3f5745d6)
…	2021-12-15 01:52:1…	10.1.2.1	10.2.3.3	ESP	186	ESP (SPI=0x1c94189e)
…	2021-12-15 01:52:1…	10.2.3.3	10.1.2.1	ESP	186	ESP (SPI=0x3f5745d6)
…	2021-12-15 01:52:1…	10.1.2.1	10.2.3.3	ESP	186	ESP (SPI=0x1c94189e)
…	2021-12-15 01:52:1…	10.2.3.3	10.1.2.1	ESP	186	ESP (SPI=0x3f5745d6)
…	2021-12-15 01:52:1…	10.1.2.1	10.2.3.3	ESP	186	ESP (SPI=0x1c94189e)
…	2021-12-15 01:52:1…	10.2.3.3	10.1.2.1	ESP	186	ESP (SPI=0x3f5745d6)

Figure 5-27. *Wireshark capture of IPSec IKEv1 negotiations*

In Phase 1, as shown in Figure 5-28, the first step is policy negotiation. In the first packet, the sender adds its unique Security Parameter Index (SPI) to identify itself. Along with the SPI, the sender also sends a set of proposals with various security parameters, called *transforms*. These transforms are used by the receiver to match with its local policies.

Figure 5-28. *Wireshark capture of first Phase 1 packet*

On receiving the packet, the receiver responds with the Responder SPI and picks one of the transforms that it received based on the configuration. Figure 5-29 displays the Wireshark capture of the reply sent by the responder for the first packet.

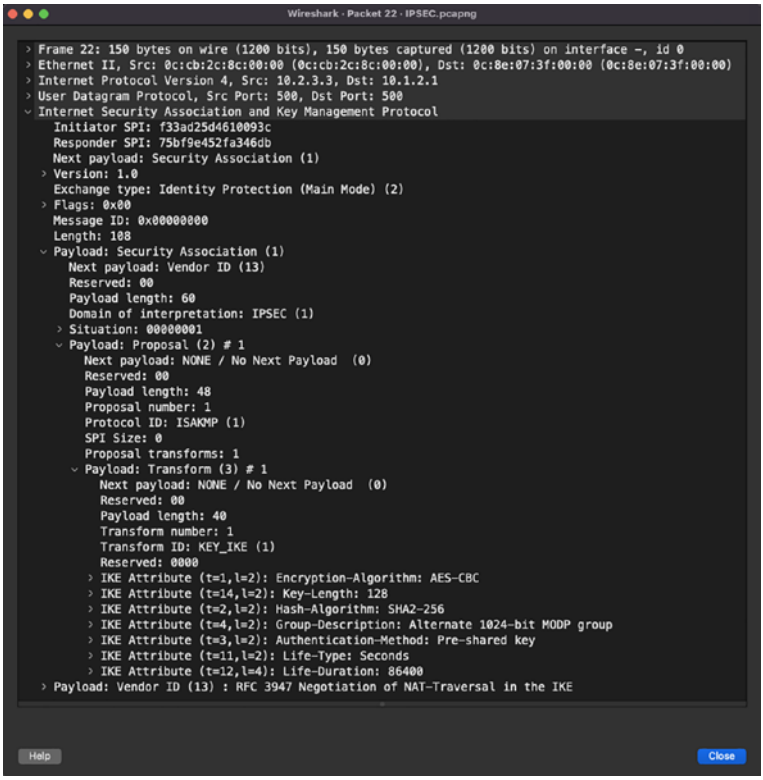


Figure 5-29. *Wireshark capture of second Phase 1 packet*

In the next two packets, both the peers exchange Diffie-Hellman (DH) public keys, which allows them to agree on a shared secret key. Figure 5-30 displays the Wireshark capture of the DH keys. Notice that there is a Nonce data highlighted in the packet capture. The Nonce value helps protect against replay attacks by adding randomness to the key generation process.

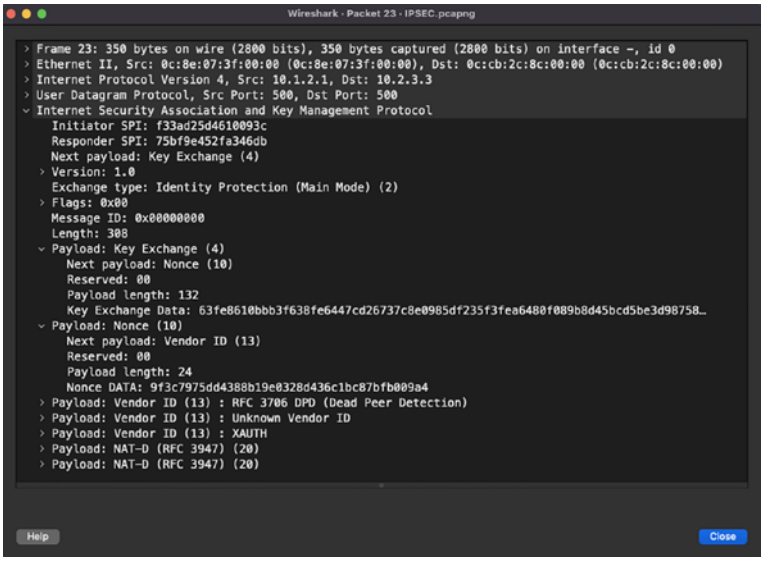


Figure 5-30. *Wireshark capture of DH keys*

The last two packets of the Main mode are used for authentication purposes. In this exchange, both peers confirm each other's identity. If both sides agreed on a preshared mechanism of authentication, then both sides check whether they have the same preshared key or not. Figure 5-31 displays the Wireshark capture of the identification-related payload. Notice in this Wireshark capture that the Flags field highlights that there is no authentication between the two peers.

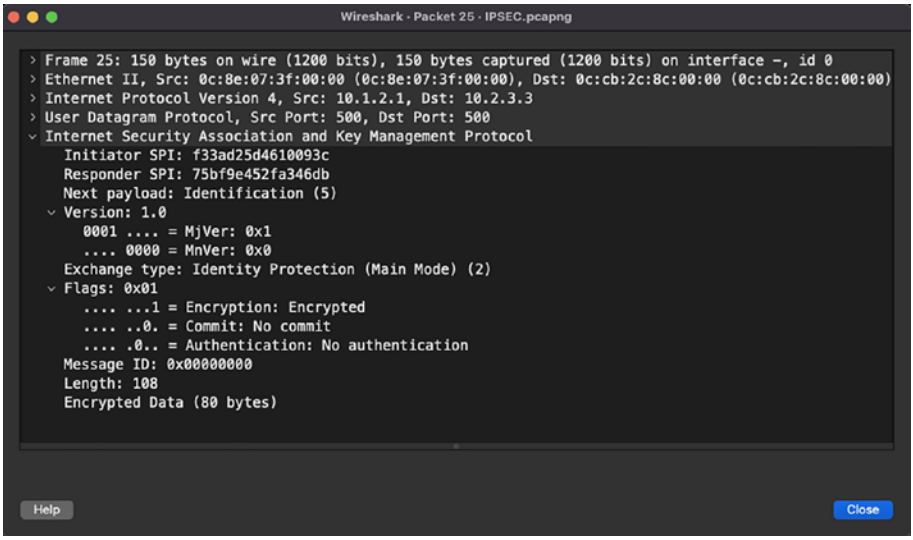


Figure 5-31. *Wireshark capture of Phase 1 authentication process*

After this step, we move to Phase 2 (Quick mode). In this phase, we primarily focus on establishing security parameters that will be used by IPSec SA. Figure 5-32 displays the packet exchanged in Quick mode. Remember that there are three packets that are exchanged in Quick mode but only one is showed for brevity.

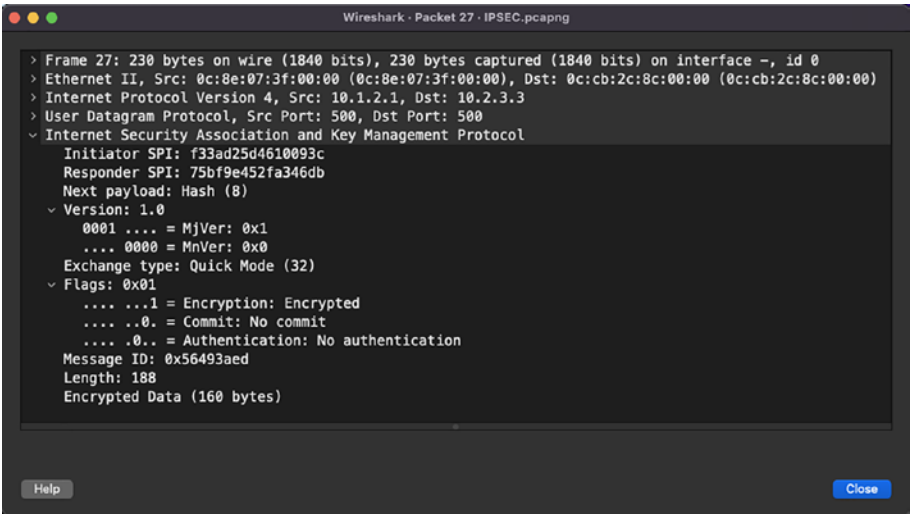


Figure 5-32. *Wireshark capture of Phase 2 Quick mode*

Once Phase 2 is completed, the IPSec tunnels are formed, and all the packets exchanged over the tunnel interface are encrypted. For instance, if you send ICMP traffic, looking at the Wireshark capture you might not be able to identify that it is an ICMP packet or some other type of packet.

VXLAN

VXLAN is an overlay protocol that provides Layer 2 extensions in a datacenter environment. It allows users to extend Layer 2 domains in multitenant environments leveraging the underlying IP infrastructure. VXLAN can also be called a MAC-in-UDP encapsulation. With VXLAN encapsulation, the original Layer 2 header is encapsulated with a UDP header and a VXLAN header. VXLAN packets are sent on the destination UDP port 4789. The VXLAN header provides a 24-bit segment ID that allows users to have up to 16 million VXLAN segments in the same datacenter environment. Figure 5-33 displays how the classical Ethernet frame looks when encapsulated with VXLAN.

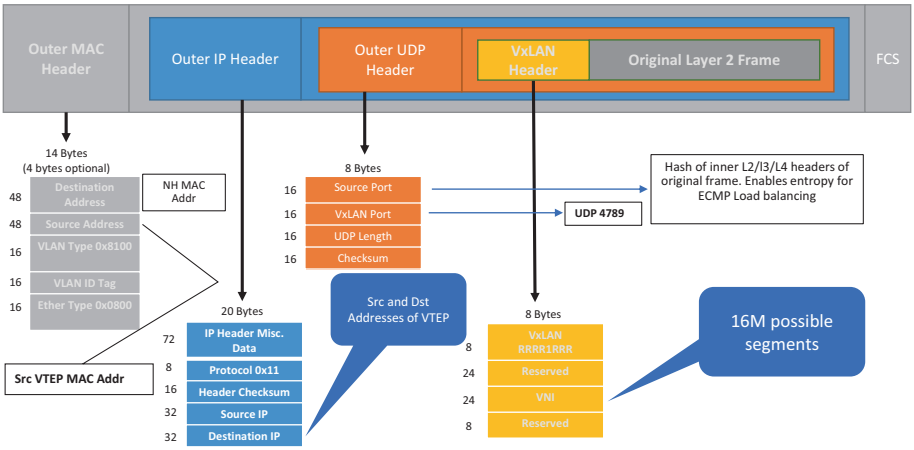


Figure 5-33. *VXLAN encapsulated Ethernet frame*

The VXLAN encapsulation and de-encapsulation is done by Virtual Tunnel End Points (VTEPs) that connect classic Ethernet segments to the VXLAN fabric. The VXLAN core fabric is usually based on a spine-leaf architecture. Traffic forwarding in VXLAN fabric is dependent on the type of traffic. Broadcast, Unknown Unicast, and Multicast (BUM) traffic requires either multicast replication or unicast replication of packets to a remote VTEP as these packets are sent to multiple VTEPs at the same time. Unicast traffic, on the other hand, does not require any kind of replication. Unicast traffic is encapsulated with VXLAN and a UDP header and sent to the destination VTEP where the host resides. There are, thus, two types of replication methods supported with VXLAN.

The first method is multicast replication. In multicast replication, a multicast group is mapped to the VXLAN Network Identifier (VNI), which in turn is mapped to a VLAN ID where the host resides. When BUM traffic is sent—for instance, an ARP request is sent for a destination host residing in the same VLAN or same VXLAN segment—the ARP request is multicast replicated to all the VTEPs that have the matching VXLAN Network Identifier (VNID) configured. The multicast destination address in the

VXLAN encapsulation is the same multicast address that was mapped to the VNI. Figure 5-34 displays the VXLAN-encapsulated BUM traffic. Notice that in this Wireshark capture, the destination address in the IP header is set to 239.1.150.1, which is the multicast address mapped to VNI 10000.

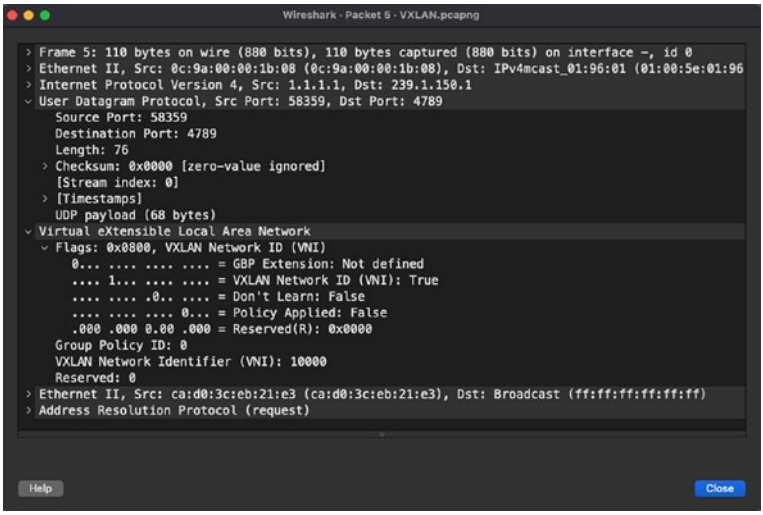


Figure 5-34. *VXLAN encapsulation BUM traffic with multicast replication*

Because the ARP response is a unicast packet, the ARP reply will be encapsulated with the VXLAN header, but will be sent as a unicast packet to the source VTEP where the source host resides. Once both the end hosts have learned about each other's MAC address, all the communication will be unicast-based communication. Figure 5-35 displays the Wireshark capture of unicast packets between the two hosts residing in same the VNI segment. Notice that the outer header has the IP address of the VTEPs and the inner IP header has the source and destination IP address of the source and destination hosts.

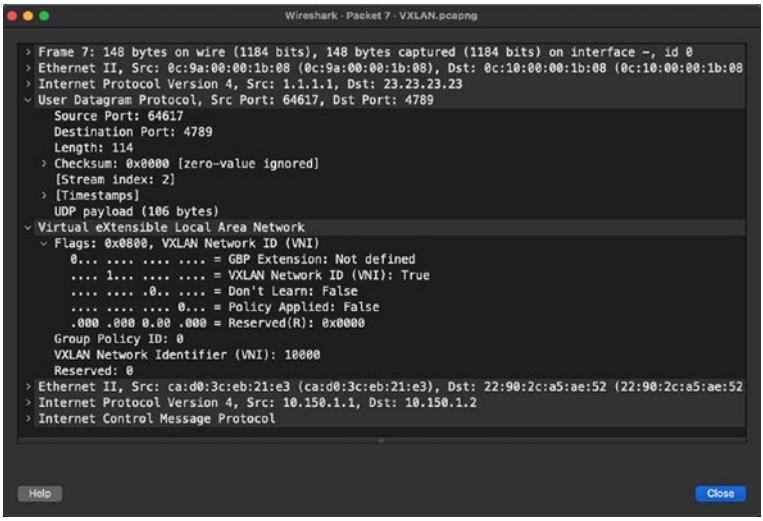


Figure 5-35. *VXLAN encapsulated unicast packet*

The second replication method is ingress replication, or unicast replication. This method is used in scenarios where either the organization is not interested in enabling multicast in its fabric or the devices are incapable of running multicast features. The BUM traffic, in this case, is replicated to statically configured remote VTEPs as unicast packets.

So far, we have explored the communication of hosts within the same VNI. Inter-VNI communication in VXLAN fabric is performed through symmetrical Integrated Routing and Bridging (IRB) and with the help of a Layer 3 VNI. For some context of what a Layer 3 or Layer 2 VNI is, let's first understand the concept of a tenant. A tenant is a logical instance that provides Layer 2 or Layer 3 services in a datacenter. Each tenant consists of multiple Layer 2 VNIs and a Layer 3 VNI. Layer 2 VNIs are the segments where the hosts are connected and the Layer 3 VNI is used for inter-VNI routing.

If we try to understand the symmetrical IRB from a packet forwarding perspective, let's consider an example where the host H1 with IP address 10.150.1.1, residing in VLAN 1501, which is mapped to VXLAN segment ID

10000, tries to reach to a host H3 with IP address 10.150.2.3 residing in VLAN 1502, which is mapped to VXLAN segment ID 10001. Because these hosts are in different VXLAN segments, we will have to leverage the Layer 3 VNI, let's say 50000. When the packet from the source host reaches the source VTEP, the VTEP performs a lookup for the destination and understands that the destination resides in a different VXLAN segment and on a remote VTEP. It therefore switches the traffic coming in on segment 10000 and sets the VNID value to 50000 when encapsulating the packet with a VXLAN header and sends it out. When the remote VTEP receives the VXLAN encapsulated packet, it notices the VNID is set to L3 VNI and it performs a routing lookup for the destination IP in the tenant VRF and realizes that it resides in the segment 10001. Because the segment after de-encapsulation is just a VLAN segment, the packet is forwarded to the host residing in VLAN 1502. Figure 5-36 displays the Wireshark capture of the VXLAN encapsulated packet with the VNID value set to 50000, which is the Layer 3 VNI.

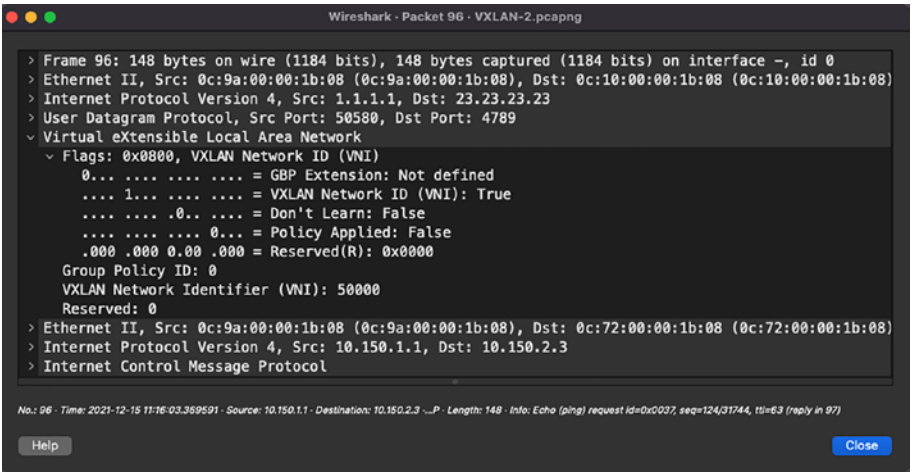


Figure 5-36. *Typical LAN*

There are various implementations of VXLAN such as VXLAN-EVPN and VXLAN Multi-site, but the concept remains the same and the method of encapsulation and de-encapsulation remains the same. Thus, when

investigating any VXLAN issue, you might run into issues related to BUM replication or unicast forwarding. In the case of BUM replication with multicast, you might want to troubleshoot the issue from a multicast perspective more than from a VXLAN perspective.

Summary

This chapter is primarily focused on topics that are specific to network engineers to assist them in day-to-day troubleshooting of various routing protocols and overlay network traffic. We began the chapter learning about how to analyze routing protocol traffic such as OSPF, EIGRP, BGP, and PIM. We then moved on to learn about overlay traffic such as GRE and IPSec VPNs. As part of analyzing overlay traffic, we also covered one of the most widely used and critical encapsulations, VXLAN. This chapter assumes that readers understand how these protocols work. They can then build on top of that to reach a deeper understanding of those protocols by learning about the content of their headers and how they can troubleshoot some scenarios that are commonly seen in production environments.

Index

V. Jain, *Wireshark Fundamentals*, https://doi.org/10.1007/978-1-4842-8002-7

F

G

H

I, J, K

L

M

N

O

P, Q

R

S

T

U

V

W, X, Y, Z