Wireshark: Analyse traffic

Mihalis Tsoukalos explains the necessary things that you need to know to start using Wireshark, and analyses three kinds of network traffic.

Our expert

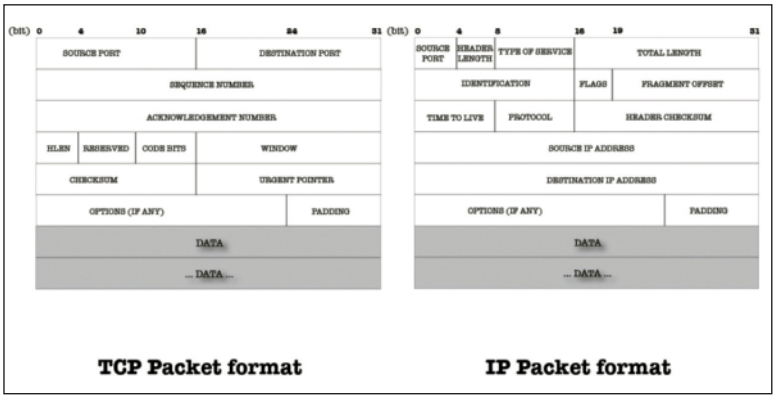
Mihalis Tsoukalos enjoys protocol analysis and packet inspection using Wireshark. He's a UNIX administrator, a programmer, a DBA and a mathematician.

Wireshark is a very popular and extremely capable network protocol analyser that was developed by Gerald Combs. *Wireshark* was born in June 2006 when Combs renamed the network tool *Ethereal*, which he also created, as he was changing jobs and couldn't use the old name anymore. Nowadays, most people use *Wireshark* and *Ethereal* has been consigned to history. Your Linux distribution will have a ready to install package for analyser too, so go ahead and install it.

You may ask what makes *Wireshark* different to other network analysers – apart from the fact that it's free – and why we're not simply advocating using *tcpdump* for packet capturing? The main advantage of *Wireshark* is that it's a graphical application. Capturing and inspecting network traffic using a graphical user interface is a very helpful thing because it cuts through the complexity of network data.

To help the beginner understand *Wireshark* they will need to understand network traffic. The aim of this article then is to supply a comprehensive introduction to TCP/IP to enable you to come to useful conclusions about the network traffic data you're analysing.

If you run *Wireshark* as a normal user, you won't be able to use any network interfaces for capturing, because of the default Unix file permission that network interfaces have. It's more convenient to run *Wireshark* as root (**sudo wireshark**) when capturing data and as a normal user when analysing network data. Alternatively, you can capture network data using the *tcpdump* command line utility as root and analyse it using *Wireshark* afterwards. Please keep in mind that on a truly busy network, capturing using *Wireshark* might slow down a machine or, even worse, might not enable you to capture everything because *Wireshark* needs more system resources than a command line program. In such cases using *tcpdump* for capturing network traffic is the wisest solution.



› The TCP packet and the IP packet format.

Capturing network data

The easiest way to start capturing network packets is by selecting your preferred interface after starting *Wireshark* and then pressing Start. *Wireshark* will show network data on your screen depending on the traffic of your network. Note that you can select more than one interface. If you know nothing about TCP, IP or the other TCP/IP protocols, you may find the output complicated and difficult to read or understand. In order to stop the capturing process you just select Capture > Stop from the menu. Alternatively, you can press the fourth icon from the left, the one with a red square (which is shorthand for 'Stop the running live capture') on the Main toolbar (Note: its exact location depends on your *Wireshark* version). This button can only be pressed while you are capturing network data.

When using the described method for capturing, you can't change any of the default *Wireshark* Capture Options. You can

see and change the Capture Options by selecting Capture > Options from the menu. There you can select the network Interface(s), see your IP address, apply capture filters, put your network card in promiscuous mode, and save your capture data in one or multiple files. You can even choose to stop packet capturing after a given number of network packets or a given amount of time or indeed a given size of data (in bytes).

Wireshark doesn't save the captured data by default but you can always save your data afterwards. It's considered good practice to first save and then examine the network packets unless there's a specific reason for not doing so.

Wireshark enables you to read and analyse already captured network data from a large amount of file formats including *tcpdump*, *libpcap*, Sun's *snoop*, HP's *nettl*, K12 text file etc. This means that you can read almost any format of captured network data with *Wireshark*. Similarly, *Wireshark* enables you to save your captured network data in a variety of formats. You can even use *Wireshark* to convert a file from a given format to another.

You can also export an existing file as a plain text file from the File menu. This option is mainly for manually processing network data or using it as input to another program.

There is an option that allows you to print your packets. I have never used this option in real life but it may be useful to print packets and their full contents for educational purposes.

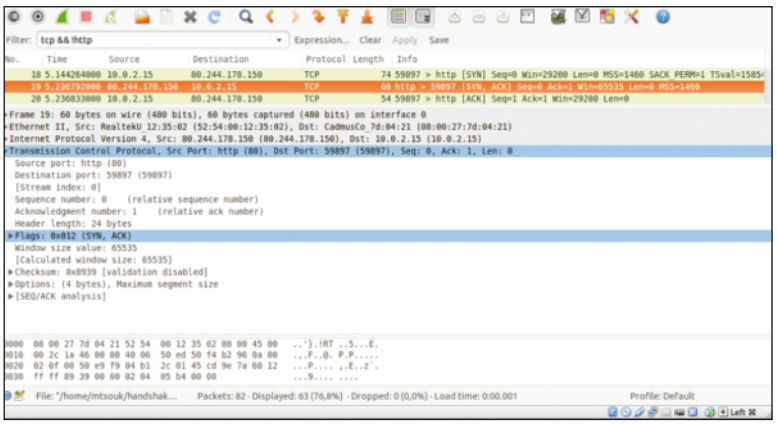
Display filters

While capture filters are applied during network data capture and make *Wireshark* discard network traffic that doesn't match the filter, display filters are applied after capture and 'hide' network traffic without deleting it. You can always disable a Display filter and get your hidden data back.

Generally, display filters are considered more useful and versatile than capture filters because it's unlikely you'll know in advance what you'll capture or want to examine. Nevertheless, applying filters at capture time can save you time and disk space and that's the main reason you might want to use them.

Wireshark will highlight when a display filter is syntactically correct with a light green background. When the syntax is erroneous, the background becomes pink.

Display filters support comparison and logical operators. The **http.response.code == 404 && ip.addr == 192.168.1.1** display filter shows the traffic that either comes from the 192.168.1.1 IP address or goes to the 192.168.1.1 IP address that also has the 404 (Not Found) HTTP response code in it. The **!bootp && !ip && !arp** filter excludes BOOTP, IP and ARP traffic from the output. The **eth.addr == 01:23:45:67:89:ab && tcp.port == 25** filter displays the traffic from or to network device with the 01:23:45:67:89:ab MAC address that uses TCP port number 25 in its incoming or outgoing connections.



The three packets (SYN, SYN+ACK and ACK) of a TCP 3-way handshake.

Keep in mind that display filters don't magically solve problems. They are extremely useful tools when used correctly but you still have to interpret the results, find the problem and think about the possible solutions yourself.

When defining rules please remember that the **(ip.addr != 192.168.1.5)** expression doesn't mean that none of the ip.addr fields can contain the 192.168.1.5 IP address. It actually means that one of the ip.addr fields should *not* contain the 192.168.1.5 IP address. Therefore, the other ip.addr field value can be equal to 192.168.1.5. You can think of it as 'there exists one ip.addr field that is not 192.168.1.5'. The correct way of expressing it is by typing **!(ip.addr == 192.168.1.5)**. This is a common misconception.

Also remember that MAC addresses are truly useful when you want to track a given machine on your LAN because the IP of a machine can change if it uses DHCP but its MAC address is more difficult to change.

It is advisable that you visit the display filters reference site for TCP related traffic at **http://bit.ly/WireSharkTCP**. For the list of all the available field names related to UDP traffic, it's advisable to look at **http://bit.ly/WireSharkUDP**.

Quick tip

The fact that the FTP protocol usually uses port number 21 doesn't mean it's not allowed to use a different port number. In other words, don't blindly rely on the port number to characterise TCP/IP traffic.

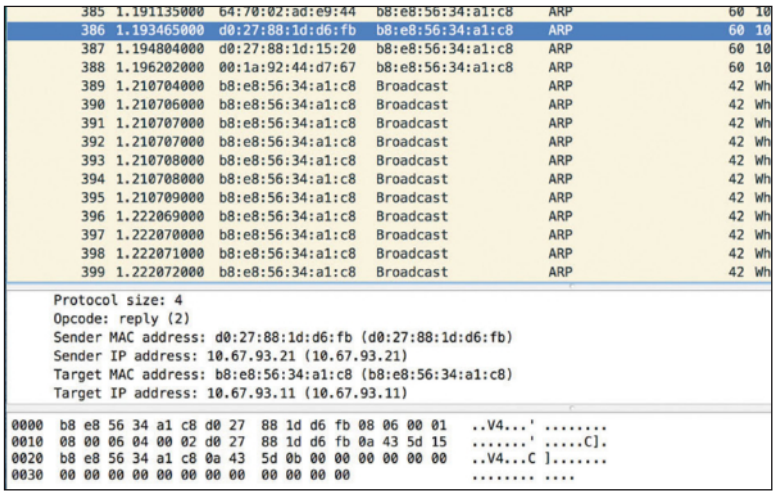
About TCP/IP, TCP and IP

TCP/IP is the most widely used protocol for interconnecting computers and it is so closely related to the internet that it's extremely difficult to discuss TCP/IP without talking about the Internet and vice versa. Every device that uses it has: »

The TCP protocol

TCP stands for Transmission Control Protocol. The main characteristic of TCP is that it's reliable and makes sure that a packet was delivered. If there's no proof of packet delivery, it resends the packet. TCP software transmits data between machines using segments (also called a TCP packet). TCP assigns a sequence number to each byte transmitted, and expects a positive acknowledgment (or ACK) from the receiving TCP stack. If the ACK is not received within a timeout interval, the data is retransmitted as the original packet is considered undelivered. The receiving TCP stack uses the sequence numbers to rearrange the segments when they arrive out of order, and to eliminate duplicate segments.

The TCP header includes both the Source Port and Destination Port fields. These two fields, plus the source and destination IP addresses are combined to uniquely identify each TCP connection. Ports help TCP/IP stacks in network connected devices (PCs and routers etc) to distribute traffic among multiple programs executing on a single device. If a service wants to be seen as reliable it's usually based on TCP, otherwise it's based on IP. But as you can imagine, reliability comes at a cost and therefore isn't always desirable.



› Part of an Nmap ping scan on a LAN as captured by Wireshark.

» **An IP address** This address must be unique at least to its local network.

» **A network mask** Used for dividing big IP networks into smaller networks that's related to the current network,

» **One or more DNS servers** Used for translating an IP address to a human-memorable format and vice versa

» **A Default Gateway** This is optional if you want to communicate with devices beyond your local network. A Default Gateway is the network device that TCP/IP sends a network packet to when it doesn't 'know' where else to actually send it.

Every TCP service listens to a port that is unique to each machine. A machine that supports the HTTP protocol, the protocol that serves WWW, is also called an HTTP server. Similarly there exist FTP servers, DNS servers, etc. It's the two pairs of the IP addresses and port numbers on both ends of a TCP/IP interaction that uniquely identify a connection between two machines that use TCP/IP.

A TCP packet (*see the format of a TCP and an IP packet segment, pictured on p70*) can be used to establish connections; transfer data; send acknowledgements, advertise the buffer that holds incoming data, which is called Window Size, and close connections. As you can see in the packet screenshot (*see p70*), each TCP segment has a header part and a data part.

The TCP 3-way handshake

TCP provides a connection-oriented, reliable byte stream service. It's a full duplex protocol, which means that each TCP connection supports a pair of byte streams; one flowing in each direction. The term 'connection-oriented' means the two applications using TCP must first establish a TCP connection with each other before exchanging any data.

The TCP header includes a 6-bit flags field that's used to relay control information between TCP peers. The possible flags include SYN, FIN, RESET, PUSH, URG, and ACK. SYN and ACK flags are used for the initial TCP 3-way handshake as you will see in a while. The RESET flag signifies that the receiver wants to abort the connection.

The TCP three-way handshake goes like this: the client sends a TCP SYN packet to the server, and its TCP header includes a sequence number field that has an arbitrary value in the SYN packet. The server sends back a TCP (SYN, ACK) packet which includes the sequence number of the opposite direction and an acknowledgement of the previous sequence number. Finally, in order to truly establish the TCP connection, the client sends a TCP ACK packet to acknowledge the sequence number of the server. After the TCP three-way handshake, the connection is established and is ready to send and receive data.

The traffic for this case was produced by running the following command:

```
$ wget http://www.linuxformat.com/
```

After some necessary DNS, ARP and ICMP network traffic, the TCP three-way handshake begins (*pictured top, p71*). The client IP address is 10.0.2.15 and the destination IP address is 80.244.178.150. A pretty simple display filter (**tcp && !http**) makes *Wireshark* display 63 out of 82 packets. The three packet numbers used in the handshake are sequential because the host wasn't performing any other network activity at the time of capturing, but this is rarely the case.

Quick tip

When you put your network card in promiscuous mode, you allow the network device to catch and read every network packet that arrives to it even if the receiver is another device on the network. Network packets still go to their original destination.

Ping scans

This part will examine the network traffic that's produced by *Nmap* when it performs a ping scan.

LAN ping scans are executed using the ARP protocol. Hosts outside a LAN are scanned using the ICMP protocol, so if you execute a *Nmap* ping scan outside of a LAN, the traffic will be different from one presented. In the example below, the *Nmap* command scans 255 IP addresses, from 10.67.93.1 to 10.67.93.255. The results show that at execution time only 10 hosts were up or, to be precise, only ten hosts answered the *Nmap* scan:

```
$ sudo nmap -sP 10.67.93.1-255
Starting Nmap 6.47 ( http://nmap.org ) at 2014-09-05 11:51 EEST
Nmap scan report for xxxxx.yyyyy.zzzzz.gr (10.67.93.1)
Host is up (0.0030s latency).
MAC Address: 64:70:02:AD:E9:44 (Tp-link Technologies CO.)
Nmap scan report for srv-gym-ag-anarg.att.sch.gr (10.67.93.10)
Host is up (0.0051s latency).
MAC Address: 00:0C:F1:E8:1D:6E (Intel)
Nmap scan report for 10.67.93.20
Host is up (0.0066s latency).
MAC Address: D0:27:88:1D:15:20 (Hon Hai Precision Ind. Co.Ltd)
Nmap scan report for 10.67.93.21
Host is up (0.0053s latency).
MAC Address: D0:27:88:1D:D6:FB (Hon Hai Precision Ind. Co.Ltd)
Nmap scan report for 10.67.93.22
Host is up (0.0080s latency).
MAC Address: 00:1A:92:44:D7:67 (Asustek Computer)
Nmap scan report for 10.67.93.29
Host is up (0.057s latency).
MAC Address: 00:78:E2:47:49:E5 (Unknown)
Nmap scan report for 10.67.93.78
Host is up (0.0023s latency).
MAC Address: 00:80:48:24:6A:CC (Compex Incorporated)
Nmap scan report for 10.67.93.147
Host is up (0.028s latency).
MAC Address: 00:14:38:64:5D:35 (Hewlett-Packard)
```

```
Nmap scan report for 10.67.93.172
Host is up (0.016s latency).
MAC Address: 00:50:27:00:E4:F0 (Genicom)
Nmap scan report for wwww.yyyyy.zzzzz.gr (10.67.93.11)
Host is up.
Nmap done: 255 IP addresses (10 hosts up) scanned in 1.25 seconds
```

The purpose of the ping test is simply to find out if an IP is up or not – see the grab on the opposite page. What's important for *Nmap* in a ping scan is not the actual data of the received packets but, put relatively simply, the existence of a reply packet. As all traffic is in a LAN, each network device uses its MAC address in the reply so you only see MAC addresses in both Source and Destination fields. The presence of a reply makes Nmap understand that a host is up and running. As a MAC address includes information about the manufacturer of the network device, *Nmap* also reports that information for you.

Nmap also calculates the round trip time delay (or latency). This gives a pretty accurate estimate of the time needed for the initial packet (sent by *Nmap*) to go to a target device, plus the time that the response packet took to return to *Nmap*. A big latency time is not a good thing and should certainly be examined.

Analysing DNS traffic

DNS queries are very common in TCP/IP networks. A DNS query creates little traffic and therefore it is an appropriate example for learning purposes. The following command will be used for generating the necessary DNS network traffic that will be examined:

```
$ host -t ns linuxformat.com
linuxformat.com name server ns0.future.net.uk.
linuxformat.com name server ns1.future.net.uk.
```

Two packets are needed in total: one for sending and one for answering the DNS query (*see grab, right*).

The first packet is number 3 and the second is number 4. A Display filter (DNS) is used to minimise the displayed data and reveal the useful information. The UDP (User Datagram Protocol) protocol was used and the desired information was sent back without any errors as shown by the Flags information. You can also tell by noting the time difference between the DNS query (1.246055000) and its answer (1.255059000) that the DNS services work fine because of the reasonable response time. The DNS server asked has the 10.67.93.1 IP address – as you can see from the destination IP address of the first packet. The same DNS server answered the DNS query as you can see from the source IP address of the second packet. The 'Answer RRs: 2' line informs us that there will be two answers for the DNS query. In time, you will be able to take all this in with one glance.

UDP uses the underlying IP protocol to transport a message from one machine to another, and provides the same unreliable, connectionless packet delivery as IP. It doesn't use acknowledgements to make sure messages arrive, it doesn't order incoming messages, and it doesn't provide feedback to control the rate at which information flows between the machines. Thus, UDP messages can be lost, duplicated, or arrive out of order. Furthermore, packets can arrive faster than the recipient can process them.

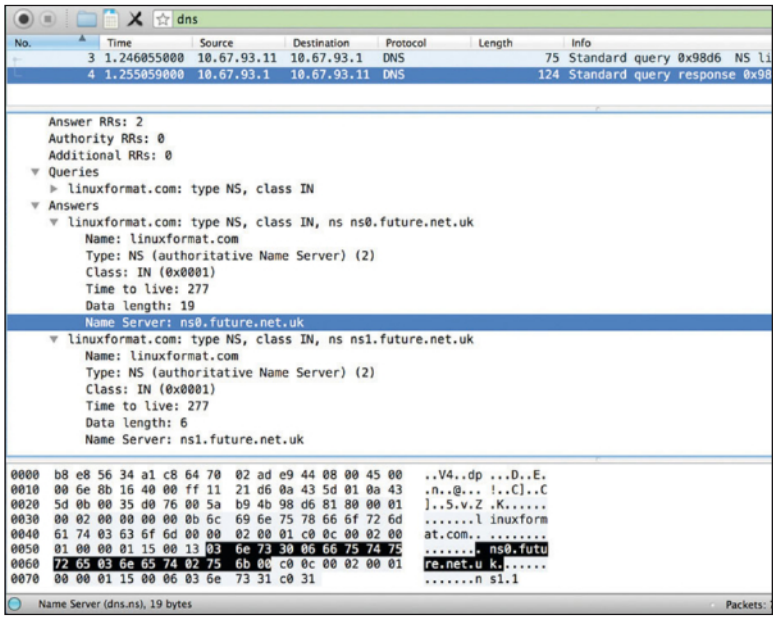
The destination port of the first packet is 53 which is the usual port number of the DNS service. The UDP part of the second packet shows the port numbers used for the reply:

```
User Datagram Protocol, Src Port: 53 (53), Dst Port: 53366 (53366)
    Source Port: 53 (53)
    Destination Port: 53366 (53366)
    Length: 90
    Checksum: 0xb94b [validation disabled]
    [Stream index: 0]
```

As it happens with most tools, the more you use *Wireshark*, the more efficient you will become with it, so keep on practicing and learning! LXF

Quick tip

There is also a console version of *Wireshark* called *tshark*. The two main advantages of *tshark* are that it can be used in scripts and that it can be used through an SSH connection. Its main disadvantage is that it does not have a GUI. *Tshark* can also entirely replace *tcpdump*.



Here is how *Wireshark* shows the traffic of a DNS query after applying a Display filter. Notice the green colour around DNS that shows the validity of it.

The IP protocol

IP stands for Internet Protocol. The main characteristic of IP is that it's not a reliable protocol by design. Unreliable means that packets may not reach its destination for various reasons, including transmission errors, network hardware failures and network congestion. Networks may also deliver packets out of order, deliver them after a substantial delay or deliver duplicates. Nevertheless, a programmer can program reliable applications that use IP by implementing their own error-checking code but this is a non-trivial task.

When the information doesn't need many network packets, using a protocol that's based on IP is more efficient than using TCP, even if you have to re-transmit a network packet, because there's no three-way handshake traffic overhead.

IP encapsulates the data that travels in a TCP/IP network, because it's responsible for delivering packets from the source host to the destination host according to the IP addresses. IP has to find an addressing method to effectively send the packet to its destination. Dedicated devices that you'd recognise as routers mainly perform IP routing but every TCP/IP device has to do basic routing.

Each IP address is a sequence of four 8-bit numbers, separated by dots. Each number has a value between 0 (=2^0-1) and 255 (=2^8-1). Example IP addresses are 10.25.128.254, 213.201.43.23 and 192.168.1.200.

IPv6 was developed by IETF and its purpose is to solve the problem of running out of IPv4 addresses. IP uses 32-bit addresses whereas IPv6 uses 128-bit addresses, offering more than 7.9×1,028 times as many as IPv4.