Practice 2: The HTTP Protocol

1. Pre-work

For the correct development of the practice, it is advisable to go to the laboratory, having studied the HTTP protocol in its versions 1.0 and 1.1. In this way, you can correctly answer the questions that arise and better understand the traffic captures made. Therefore, you must beforehand:

- Study chapter 2.2 of Kurose's book.

For optional reference information, you can also check out the following resources:

- HTTP:
  http://www.rfc-editor.org/rfc/rfc2616.txt
- Guía usuario Wireshark:
  http://www.wireshark.org/docs/
- Firefox Network Monitor
  https://developer.mozilla.org/es/docs/Tools/Monitor_de_Red

2. Objectives of the internship

At the end of the lab, you should be able to perform the following tasks:

Related to the NC program:

- Use the “nc” program as a basic TCP client, for example, to send an HTTP 1.0 and HTTP 1.1 request.

Related to the Wireshark program:

- Use Wireshark to capture traffic from a given application, particularly for HTTP traffic, using a capture filter.
- Use the "Follow TCP Stream" command to display the application's client/server dialog.

As for HTTP, you should be able to …

- Differentiate between mandatory and optional headers in versions 1.0 and 1.1 in customer requests.
- Understand how virtual web servers located on the same physical server can be used.

- Describe how the headers and message body (if any) are separated in client requests and server responses.
- Explain when conditional requests are used, what they consist of, how the server identifies them, and what responses it can send.
- Describe the meaning of the headers: Date, Last-Modified, and Content-Length.

3. Basic composition of an HTTP request

Using netcat to make an HTTP request

The Netcat tool, already described and used in previous practices (see Appendix A), allows us to send and receive information using TCP and UDP sockets through simple shell commands. Therefore, **you will need to use it from a terminal**. Appendix A shows detailed information about the parameters that it accepts.

Since the HTTP protocol is a character-oriented protocol, its syntax is perfectly readable, so we can use the **netcat** command to compose requests and get responses.

Exercise 1: Using Netcat

Send an HTTP request of type GET to the web server whose IP address is 158.42.180.62 (serveis-rdc.redes.upv.es) on its port 80 via netcat, to obtain the document whose URI is "http://serveis-rdc.redes.upv.es/". Try requesting it with HTTP/1.0. For example, type:

```
nc -C -v 158.42.180.62 80
```

With this command you are connecting, via the TCP protocol (default), to port 80 (HTTP port) of the server 158.42.180.62 (*serveisrdc.redes.upv.es*). The `-C` option tells *netcat* to send the character pair *<CR><LF> when the* <Enter> *key is pressed* (in macOS the "c" must be lowercase). The `-v option` indicates that a message is displayed when the connection has been initiated.

Once the connection is established, you must request the resource:

```
GET / HTTP/1.0 ↵
↵
```

Note: You must press the enter key **twice** after the request line. As a result of this action, you will receive a text similar to the following:

```
HTTP/1.1 200 OK
Date: Tue, 10 Sep 2024 10:40:53 GMT
Server: Apache/2.4.52 (Ubuntu)
Last-Modified: Fri, 18 Nov 2022 18:52:02 GMT
ETag: "f2-5edc3338ea3f3"
Accept-Ranges: bytes
Content-Length: 242
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<html>
      <head>
            <meta charset="UTF-8">
            <title>serveis-rdc.redes.upv.es</title>
      </head>
      <body>
<h1>Welcome to the serveis-rdc.redes.upv.es</h1 website>

<H2>serveis-rdc.redes.upv.es</H2>
<h2>158.42.180.62</h2>

      </body>
</html>
```

The content received has been displayed by the standard output, which in this case was the screen. If you want to capture this text, you can redirect the output to a file using the ">" operator.

In case the response has not been returned to the system prompt at the end of receiving the response, you must press Ctrl+C on the keyboard to finish the `nc command`.

Issues:

1. What is the code of the answer? What does that code mean? Note: If the answer code is not 200, check with your teacher. 200 means "success"
2. When the sending of the web page from the server to your computer is complete, has the TCP connection (received "Connection: close" header) been closed? It is possible that the nc program is not finished and you have had to press Ctrl+C, even though the TCP connection had been closed. This is simply a problem with the implementation of the NC program. Download the web page again, but this time use the command "telnet 158.42.180.62 80" instead of using the command "nc -C -v 158.42.180.62 80" that you used previously. Is the TCP connection closed after the website is sent? What conclusions do you draw? telnet closes the connection automatically, but nc doesn't. This is due to problems in the implementation of nc.

Exercise 2: Using netcat (continued)

Make a new connection with *netcat* and requests the page again, this time using version 1.1 of the HTTP protocol. **Note:** Your web server may respond with a "400 Bad Request" code.

If they answer you with that code, it means that your application was incomplete. It lacks a header. What should your application look like for it to be correct? As you will probably have to request the website several times, to avoid the tedious process of typing the "GET / HTTP/1.1" each time, you can make use of the gedit text editor (type "gedit &" in the terminal) and prepare your request in that editor. So you only have to copy and paste each time. In addition, it will be useful in the rest of the exercises. **Note:** To copy and paste very conveniently and quickly, you can select the text with the left mouse button and paste it into the destination window with the middle mouse button. Also, as long as the text remains selected, you don't need to select it again. Thus, by pressing the middle button of the mouse you can paste it multiple times.

Issues:

1. Which headers are mandatory in the customer's request? You also need the "Host" header (ex: Host: zoltar.redes.upv.es)
2. How do you indicate the end of the headers in the client request and in the server response? With a double newline (\r\n\r\n)
3. When was the HTML file last modified on the server?
4. What is the difference between Date and Last-Modified headers? Date refers to the time of the response, last-modified to the time the file was last modified
5. What is the byte size of the page sent by the server indicated in the "Content-Length" header?

Exercise 3: Virtual Server Access

Reconnect to the same server using the command `nc -C 158.42.180.62 80`

Make the same GET request line as in the previous exercise, but adding a seconds line with the header:

```
Host: zoltar.redes.upv.es
```

Now repeat the process, but using the header: `Host: serveis.redes.upv.es`

Issues:

1. Are the answers obtained the same as in the previous exercise? No, now the answer code is 200
2. Can you explain what's going on? For more clues, try connecting to the web server using the browser with the URLs:

serveis-rdc.redes.upv.es
zoltar.redes.upv.es
serveis.redes.upv.es

Also try to find out the IP addresses of the three servers with the commands:

host -t A serveis-rdc.redes.upv.es
host -t A zoltar.redes.upv.es
host -t A serveis.redes.upv.es

Are all three servers on the same machine (same IP address) or on different machines? They are on the same machine

What conclusion can you draw in this particular case by using the Host header with the three values used in this exercise? The 3 of them can be accessed through nc by using the host header

Exercise 4: Composing a Web Page

Start a new connection with *netcat* and request the main page again (nc -C 158.42.180.62 80 and use the Host: zoltar.redes.upv.es header). Inspect the HTML text provided by the server. You'll see that it references two JPG images.

Request, with a new connection to the *netcat* program, the first of the images (remember to use version 1.1 of the HTTP protocol). You'll need to put the "/" symbol before the image name.

Repeat the process to get the second JPG image.

Issues:

1. What is the value of the Content-Length and Content-Type headers for each of the images? What was that value in the case of the HTML page? 1518 and 1842 The HTML page had a content length of 297
2. After downloading the website and the two JPG images, could you tell us how many objects this website is made of? 3 (the .html, and the 2 images)
3. How many TCP connections have you made with the *netcat* program to download all the objects that make up this website? 3 (one for each object, although you can make just one with Keep-Alive)

Exercise 5: Using Persistent Connections

Version 1.1 of the HTTP protocol allows the TCP connection not to be closed after the requested object is returned to the server. This was not possible in version 1.0 of that protocol. Let's test IRL. Start using the netcat program to establish a TCP connection to our test server (nc-C 158.42.180.62 80) and request the web page using version 1.0 of the HTTP protocol as you

did in the first exercise. If you try to request that website again, you will see that it is not possible, because as we have seen in the first exercise, the TCP connection has been closed.

Now request the zoltar.redes.upv.es webpage using version 1.1. of the HTTP protocol. Once you receive the response, request it again (you will have to be quick because if you take too long, the server decides to close the TCP connection assuming that it is inactive; for this it will be convenient to copy and paste the request with the middle mouse button). Do you get new copies of the website? You can even try to download, after the website, one of the JPG images, as we have seen in exercise 4. The conclusion is that since in version 1.1 of the HTTP protocol the connections may be persistent, so on the same TCP connection, you can request and receive several objects. This was impossible with version 1.0 of the protocol, where each object had to have a new TCP connection.

Exercise 6: Keep-Alive Header

Because persistent connections can be used in version 1.1 of the protocol, the client can request the web server to keep the connection alive for a set amount of time. Another issue is that the server decides to listen to the client.

In this exercise, we're going to find out what happens to our sample server when the client requests to keep the connection open for a certain amount of time. To do this, it re-establishes a TCP connection to the server using the netcat program and requests the zoltar.redes.upv.es web page, but this time it adds the headers

```
Connection: Keep-Alive
Keep-Alive: timeout=50
```

With these headers, the client tells the server that it wants to keep the connection open for 50 seconds.

Look at the server's response. Is there any header in the answer that was not previously there? Does the server listen to the client? What does the server tell the client about how long it will keep the TCP connection open? In the answer, we find the "Keep-Alive" header, that says: "Keep-Alive: timeout=5, max=100"

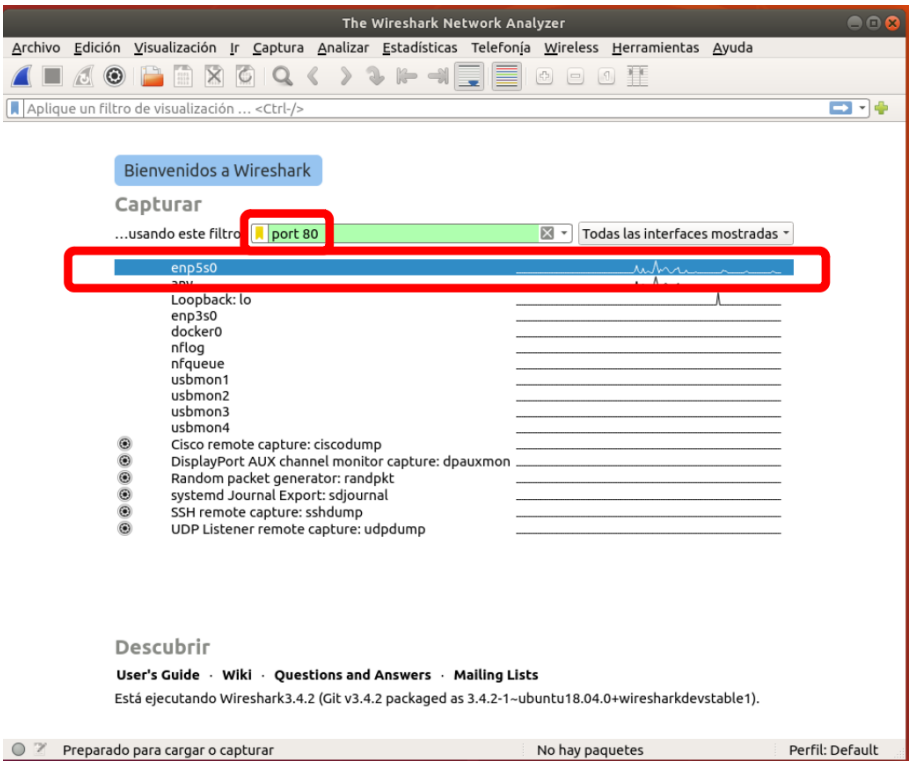
4. Using the Protocol Analyzer

In this part of the practice we are going to work with the Wireshark protocol analyzer, which will allow us to analyze the HTTP traffic exchanged between our browser and the web servers to which we connect.

Remember that before generating traffic (requesting web pages), you must start the capture using the Wireshark program. To do this, keep in mind a couple of basic steps:

- It is convenient that you check that the interface you are going to capture is the correct one. Check that the selected interface (blue bar) corresponds to your network card (enp5s0 in the image below). There are line graphs indicating traffic, so it must be one of those that record activity.

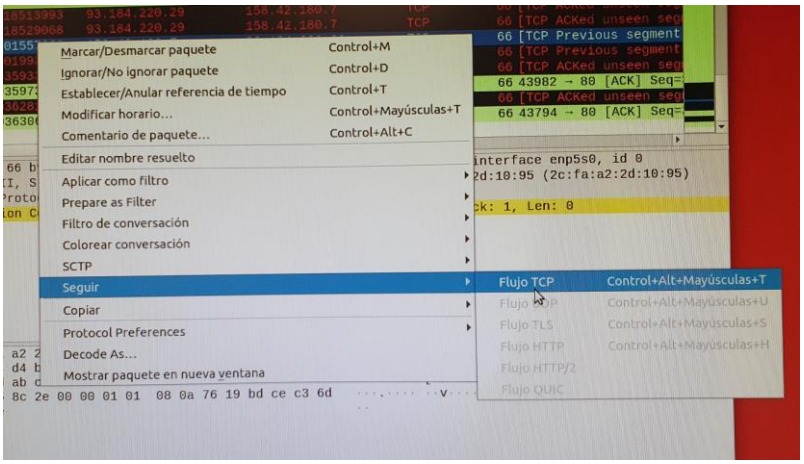
- As we will capture HTTP traffic in the section "... *Using this filter*" enter "port 80".



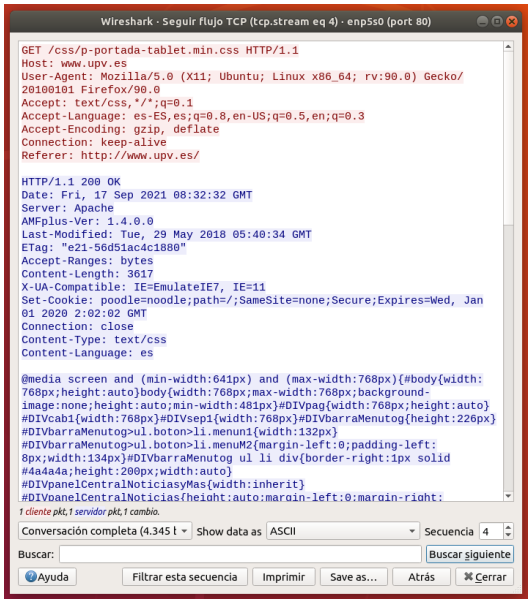
The "Follow>TCP Stream" command

When working with TCP-based application protocols, it can be very useful to be able to view the data exchanged on a TCP connection in the same way that the application layer sees it, especially when the packets associated with different TCP connections are interleaved in the capture. For this purpose, the Wireshark program provides the "TCP Sequence" option. When you select a packet from a given connection in which you are interested and select the "Follow>TCP Stream" option

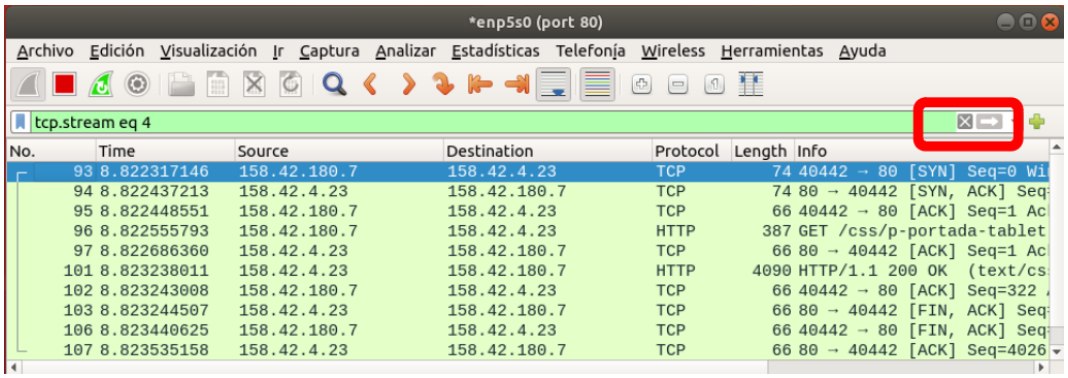
from the "Analyze" menu, Wireshark automatically applies a display filter and opens an additional window showing in order all the data exchanged at the application level by the client and server. You can also activate it via the menu that opens when you click on the right mouse button, after you have selected the corresponding package:



In addition, to facilitate its interpretation, it uses a different color for the data at each of the ends of the communication:



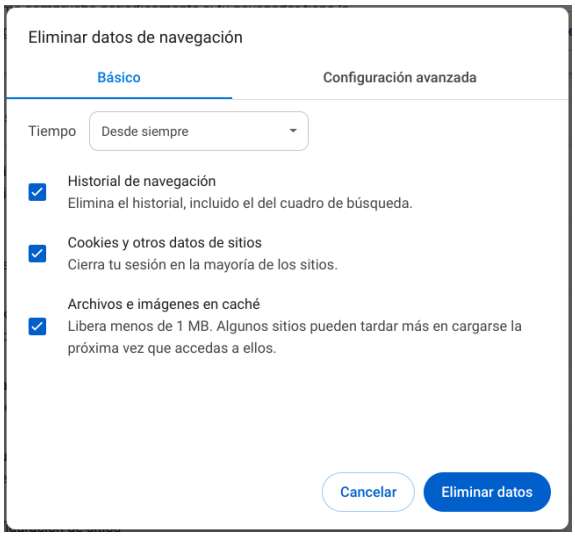
If the original screenshot showed several different TCP connections, you can now only see one. You can use other TCP connections by varying the number that appears in the "Sequence" box that appears at the bottom right of the window in the figure above. If you want to see them all again, you must remove the display filter that was automatically added when you selected the "Follow/TCP Sequence" option. Press "the X button" to remove it. The following figure will help you.



Clearing the browser cache

In the following exercises, it is advisable to clear the cache of the web browser before doing the exercise. In this practice we are going to focus on the use of the Chrome browser because the other widely available browser (Firefox) generates too much traffic autonomously before accessing the requested web server and that can be annoying when we are capturing HTTP traffic.

To clear the browser cache in Chrome, you have to click on the three vertical dots in the browser window, in the upper right corner and, in the new menu that appears, select the entry "Delete browsing data". A window like the one on the right will appear, in which we will select to delete the history from always (so we are sure not to fail).



Exercise 7: Capturing Traffic Over Port 80

Configure the Wireshark program to capture port 80 traffic using a capture filter ("port 80" syntax). Start the capture and, **using the `Chrome browser,`** access this resource:

`http://zoltar.redes.upv.es`

Analyze, using the "*Follow>TCP Stream" option*, the first HTTP request/response exchange of the captured traffic. You must have received a response from the server `"HTTP/1.1 200 OK",` if not, check the URL you had entered in the browser (you must include the protocol field of the URL). **If you have to recapture the page, you must first empty the browser cache.**

Issues:

1. Does your browser use HTTP 1.0 or 1.1? Which version of HTTP is the server using?
   Both the requests and the answers are made in http 1.1
2. What languages is your browser indicating to the server you prefer?
   en-US
3. What is the IP address of your computer? What about the server? (you can look at IP addresses in the general capture window, they are labeled as "*Source*" and "*Destination*"). 10.236.40.25 | 158.42.180.62
4. What TCP ports is the browser and web server using? 80 (I guess)
5. Analyze the "*Connection*" headers of the client and server, are they using persistent or non-persistent connections? They are using persistent connections (with the Keep-Alive header)
6. How many objects has the browser downloaded with that TCP connection? Are all the objects on the website there? Have other TCP connections been created during the download of the website? What ports does the browser use on those other TCP connections? What about the server?
7. Do the values of the Content-Length and Content-Type headers match the values of previous years?
8. Does the browser request any additional objects?

Exercise 8: Capturing www.upv.es Traffic Over Port 80

Configure the Wireshark program to capture port 80 traffic using a capture filter ("port 80" syntax). Start the capture and, **using the `Chrome browser`,** access this resource:

`http://www.upv.es`

Analyze, using the "*Follow>TCP Stream" option*, the second HTTP request/response exchange of the captured traffic. You must have received a response from the server "`HTTP/1.1 200 OK`"; if not, check the URL you entered in the browser (you must include the URL's protocol field). **If you have to recapture the page, you must first empty the browser cache.**

Issues:

1. Does your browser use HTTP 1.0 or 1.1? Which version of HTTP is the server using?
2. What languages is your browser indicating to the server you prefer?
3. What is the IP address of your computer? What about the server? What TCP ports are used in your browser and web server?
4. Analyze the "*Connection*" headers of the client and server, are they using persistent or non-persistent connections?
5. What is the value of the Content-Type header?
6. In the server response, is there a Content-Length header? What does the Transfer-Encoding header mean?
7. Use the connection counter in the lower right corner to cycle through the different TCP connections that have been generated during the download of the web page. How many TCP connections have been generated? Are these other TCP connections used persistent or non-persistent connections? What are the values of the Content-Type headers that appear in the different connections?

Exercise 9: Using Pipelining in HTTP 1.1

Version 1.1. of the HTTP protocol allows you to speed up the download of web pages by using the pipelining technique, which is nothing more than requesting several objects at once, without waiting for their arrival and, once requested, receiving them one after the other. In this way, productivity is greatly increased because we avoid many of the RTTs associated with the HTTP dialog. Let's check this out with our sample web server in zoltar.redes.upv.es

The first step for this experiment is to use the gedit text editor and write the following in it:

```
GET/HTTP/1.1
Host: zoltar.redes.upv.es

GET /oto1.jpg HTTP/1.1
Host: zoltar.redes.upv.es

GET /oto2.jpg HTTP/1.1
Host: zoltar.redes.upv.es
```

Notice that after each Host header there is a blank line.

Once the requests are prepared, configure the Wireshark program to capture port 80 traffic and use the netcat program to establish a TCP connection to our sample web server (nc-C 158.42.180.62 80). Once the TCP connection has been established, copy and paste the three GET requests from the gedit into the window in which you ran the netcat program. Finally, stop capturing traffic on Wireshark.

Issues:

1. Check on the terminal what you have received from the server. Have you received the three requested items?
2. In the traffic captured on Wireshark, do the three GET requests go together in the same frame or do they go in different frames?
3. In the traffic captured on Wireshark, are the three GET requests sent before receiving the first response from the server or, on the contrary, are they followed in a request-response, request-response, request-response order?
4. How many 200 OK responses do you receive? In what order are the objects received? In the same order as they were requested or in a different order?

HTTP headers related to the browser cache

Now let's see how the browser cache affects the requests made by the client. If the browser already has the resource in its cache, and suspects that it may have been modified since it obtained it, the client will make a conditional request, using the "If-modified-since" header or other conditional headers. If the resource has not been modified on the server, the response it sends will also not be the usual one (200 OK). Let's look at the differences, both in the client's request and in the server's response.

Exercise 10: Using the Cache with zoltar.redes.upv.es

Clear your browser's cache and access the URL:

`http://zoltar.redes.upv.es`

Check that all responses have "`200`" as the status code. Prepare the Wireshark analyzer to perform a capture on TCP port 80. Without clearing the cache, repeat the request for that web page.

Issues:

1. What does the browser prompt look like? Does it include any conditional headers (starting with "if")? How many conditional headers are there? What are they? What do they mean?

2. What is the response from the server? Could another response have been expected from the server?

3. How many requests does the browser send to the server? Have all the objects on the website been requested?

Exercise 11: Using the cache with www.upv.es

Clear your browser's cache and access the URL:

`http://www.upv.es`

Prepare the Wireshark analyzer to perform a capture on TCP port 80. Without clearing the cache, repeat the request for that web page. Use the "Follow>TCP Stream" option and the connection counter in the bottom right corner to go through the different TCP connections that have been generated during the download of the website. Does the web server return any responses with code other than 200?

Appendix A

The netcat tool: uses and parameters

usage: nc [-46DdhklnrStUuvzC] [-i interval] [-P proxy_username] [-p source_port]
[-s source_ip_address] [-T ToS] [-w timeout] [-X proxy_protocol]
[-x proxy_address[:port]] [hostname] [port[s]]

Command Summary:

-4	Use IPv4
-6	Use IPv6
-D	Enable the debug socket option
-d	Detach from stdin
-h	This help text
-i secs	Delay interval for lines sent, ports scanned
-k	Keep inbound sockets open for multiple connects
-l	Listen mode, for inbound connects
-n	Suppress name/port resolutions
-P proxyuser	Username for proxy authentication
-p port	Specify the local port for remote connects
-q dry	quit after EOF on stdin and delay of secs (-1 to not quit)
-r	Randomize remote ports
-S	Enable the TCP MD5 signature option
-s addr	Local source address
-T ToS	Set IP Type of Service
-C	Send CRLF as line-ending
-t	Answer TELNET negotiation
-In the	Use UNIX domain socket
-in the	UDP mode
-v	Verbose
-w secs	Timeout for connects and final net reads
-X therefore	Proxy protocol: "4", "5" (SOCKS) or "connect"
-x addr[:port]	Specify proxy address and port
-with	Zero-I/O mode [used for scanning]

Port numbers can be individual or ranges: lo-hi [inclusive]