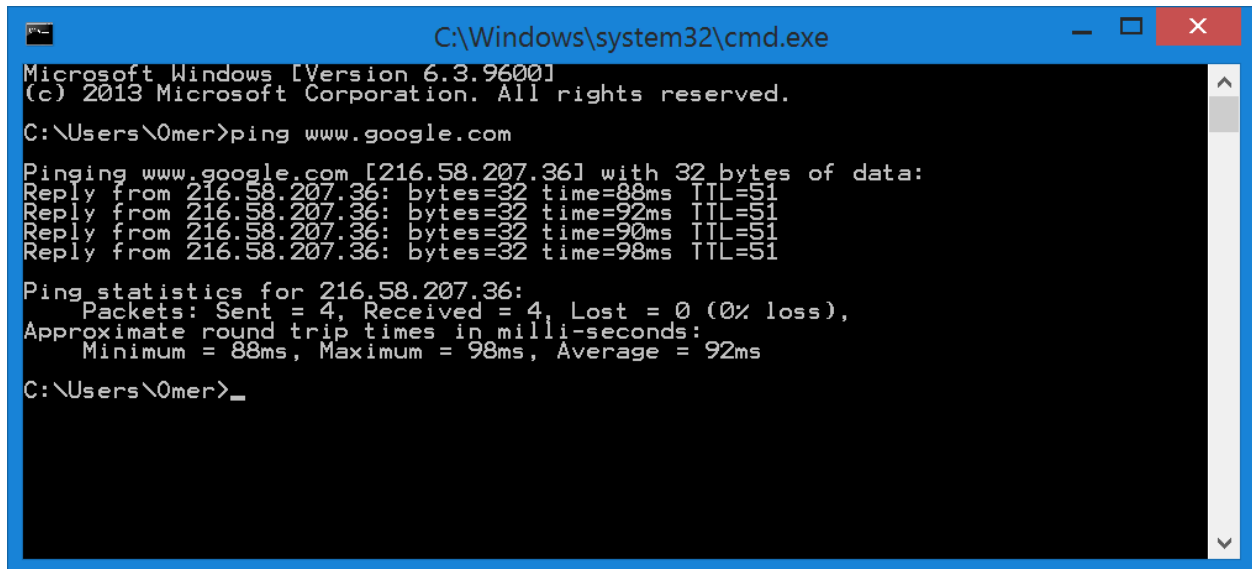


Wireshark Lab - Exercise 1 - Ping

Ping is a useful utility to check for remote servers' connectivity.

To use it, run the command line. [This page](#) explains how to do that in Windows, [this page](#) explains how to do that in OSX.

Now, we can try to ping <address> using the command line. By default, ping sends 4 requests and waits for a **pong** answer.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Omer>ping www.google.com

Pinging www.google.com [216.58.207.36] with 32 bytes of data:
Reply from 216.58.207.36: bytes=32 time=88ms TTL=51
Reply from 216.58.207.36: bytes=32 time=92ms TTL=51
Reply from 216.58.207.36: bytes=32 time=90ms TTL=51
Reply from 216.58.207.36: bytes=32 time=98ms TTL=51

Ping statistics for 216.58.207.36:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 88ms, Maximum = 98ms, Average = 92ms

C:\Users\Omer>
```

In the command line above, I've written the command **ping** [www.google.com](#).

We can see that Google has responded with four replies. The time it took for the message to return varied between 88 and 98 milliseconds.

Ping is useful to determine whether a remote service is available, and how fast it is to reach that service. If it takes a very long time to reach a reliable server such as google.com, we might have a connectivity problem.

Run the following command from your command line (or terminal):

ping -n 1 [www.google.com](#) (Note the -n option is equivalent to the -c one in *NIX systems!)

Use wireshark to answer the following questions:

- 1) What protocol does the **ping** utility use? Internet Control Message Protocol (ICMP)
- 2) Using only wireshark, compute the RTT (Round Trip Time) – how long it took since your ping request was sent and until the ping reply was received?

Right-click to set the initial request as the reference and check the time of arrival of the second packet. It's roughly the same as the one shown in the CLI (6,674 ms)

The `-l` option is equivalent to `-s` in *NIX systems and controls the data field size. By default it'll be 56B so that when we add the 8B introduced by ICMP's overhead the total packet length will total 64B. If we set it to 342 we'll see the ICMP datagram will have a size of 350B! Note google.com's ping server implementation is non-compliant as the reply it generates does fiddle with the data field as soon as the total ICMP packet size is above 76B! This implies that if we use `-s 68` the reply will respect the standard BUT if we issue `-s 69` it won't! It just "cuts off" the size at a given point. This doesn't happen when pinging our own machine or other services such as theguardian.com!

Next, run the following command:

```
ping -n 1 -l 342 www.google.com
```

3) What is the main difference between the packet sent by this command, and the packet sent by the previous command? Where in wireshark can you see this difference, inspecting the packets?

4) What is the content (data) provided in the ping request packet? What is the content provided in the ping response packet?

It's just a bunch of padding data. If you look at the binary dump it's just a series of ever incrementing values. We should note the reply's data should be exactly the same but as we pointed out google.com's implementation is NOT respecting that unless we use the default ECHO REQUEST size... The timestamp is NOT part of this data field!

Wireshark Lab - Exercise 2 - PCAP

Download the PCAP file from [here](#).

Answer the following questions:

1) How many packets were sent to 192.168.1.3?

By applying the `'ip.dst == 192.168.1.3'` we can see it was sent 149 packets as that's the amount of displayed packets as seen in the bottom right hand corner.

2) What frames are ARP frames? Provide the frame numbers.

Applying the `"arp"` filter we find this protocol used in frames 87, 134 and 135.

3) How many packets have we captured overall?

As seen in the bottom right corner we have a total of 292 packets.

4) How many of these packets are TCP (Transmission Control Protocol) packets?

If we only consider "pure" TCP traffic we'll find we have 100 segments in the trace. We can verify that by applying the `'tcp && !tls'` filter to avoid encrypted segments. We can check the counts in the bottom right corner as always. If we are to consider TLS traffic as TCP one (which is usually the case) we are then looking at 234 segments, something we can test with the `'tcp'` filter that, as we said, includes the TLS segments by default.