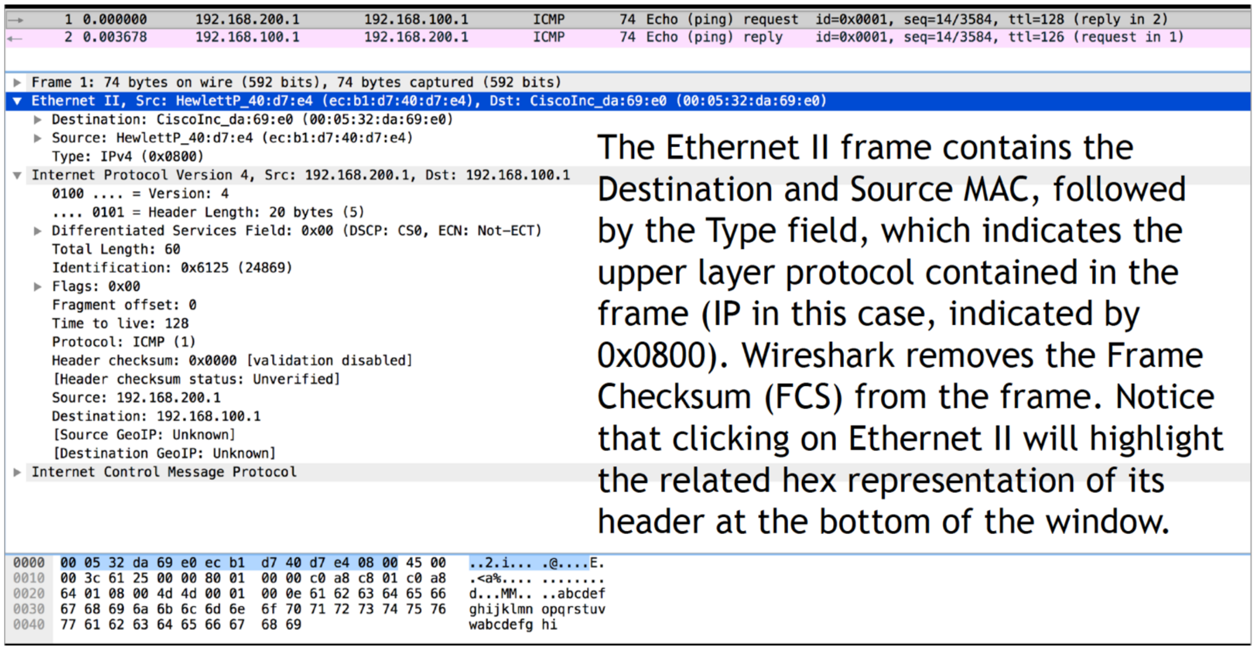
**Packet Capture Analysis Intro**

Packet capture analysis allows network engineers, system administrators, and security analysts to quantify network performance and analyze network traffic using packet captures on various components in the network. Packet capture analysis may result in outcomes such as introducing additional network components for load balancing, new routes/paths through the network, or spinning up further analysis for confirming networking breaches.

**Internet Control Message Protocol (ICMP)**

ICMP is used by the Internet Protocol to send error messages and operational/diagnostic information to devices in a network. The most common use of ICMP is the “ping” program, which generates an ICMP Echo Request (an ICMP Type 8 message) and waits for an ICMP Echo Reply (an ICMP Type 0 message) to determine connectivity between two network devices. Any time you see “ping” in an online game for example, what’s actually being computed is the **round trip time** between your device and the server hosting the game. That is, how long it took for your ICMP Echo Request message to reach the server plus how long it took for the server’s ICMP Echo Reply message to reach your device.



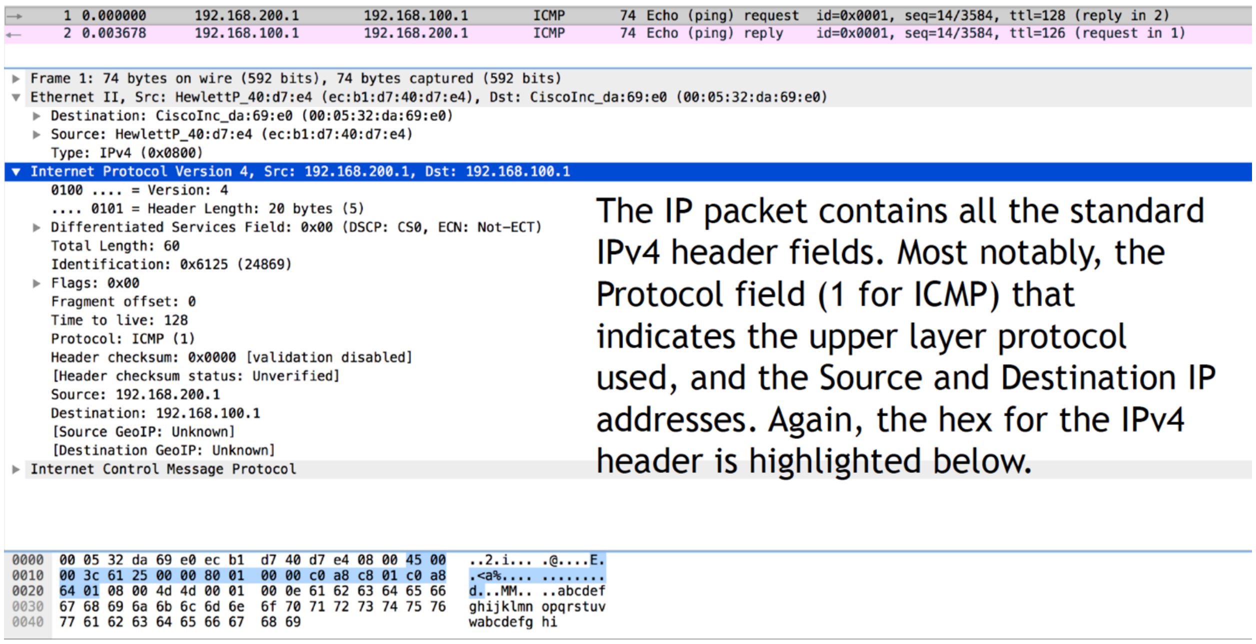
```
→     1 0.000000      192.168.200.1       192.168.100.1       ICMP      74 Echo (ping) request  id=0x0001, seq=14/3584, ttl=128 (reply in 2)
←     2 0.003678      192.168.100.1       192.168.200.1       ICMP      74 Echo (ping) reply    id=0x0001, seq=14/3584, ttl=126 (request in 1)

▶ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▼ Ethernet II, Src: HewlettP_40:d7:e4 (ec:b1:d7:40:d7:e4), Dst: CiscoInc_da:69:e0 (00:05:32:da:69:e0)
   ▶ Destination: CiscoInc_da:69:e0 (00:05:32:da:69:e0)
   ▶ Source: HewlettP_40:d7:e4 (ec:b1:d7:40:d7:e4)
     Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 192.168.200.1, Dst: 192.168.100.1
     0100 .... = Version: 4
     .... 0101 = Header Length: 20 bytes (5)
   ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
     Total Length: 60
     Identification: 0x6125 (24869)
   ▶ Flags: 0x00
     Fragment offset: 0
     Time to live: 128
     Protocol: ICMP (1)
     Header checksum: 0x0000 [validation disabled]
     [Header checksum status: Unverified]
     Source: 192.168.200.1
     Destination: 192.168.100.1
     [Source GeoIP: Unknown]
     [Destination GeoIP: Unknown]
▶ Internet Control Message Protocol

0000  00 05 32 da 69 e0 ec b1  d7 40 d7 e4 08 00 45 00   ..2.i... .@....E.
0010  00 3c 61 25 00 00 80 01  00 00 c0 a8 c8 01 c0 a8   .<a%.... ........
0020  64 01 08 00 4d 4d 00 01  00 0e 61 62 63 64 65 66   d...MM.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

The Ethernet II frame contains the Destination and Source MAC, followed by the Type field, which indicates the upper layer protocol contained in the frame (IP in this case, indicated by 0x0800). Wireshark removes the Frame Checksum (FCS) from the frame. Notice that clicking on Ethernet II will highlight the related hex representation of its header at the bottom of the window.

Taking a look at the above packet capture from Wireshark (try opening example.pcap in Wireshark yourself to examine the packet capture. Wireshark is a free download that runs on any OS), you’ll see that ICMP operates **on top of** the Internet Protocol (IP) protocol. The IP protocol operates at Layer 3 of the 5-layer TCP/IP protocol stack. As such, this means that ICMP operates at Layer 3.5 (who knew there was a Layer 3.5? Now you do!). Furthermore, the IP protocol is encapsulated inside an Ethernet II frame at Layer 2.

Let’s take a look at some of the necessary details of Ethernet II, IP, and ICMP, so that we can discuss how to analyze packets! I’m assuming that this is review for you, but if this is new, please reach out to your instructor if you have questions.

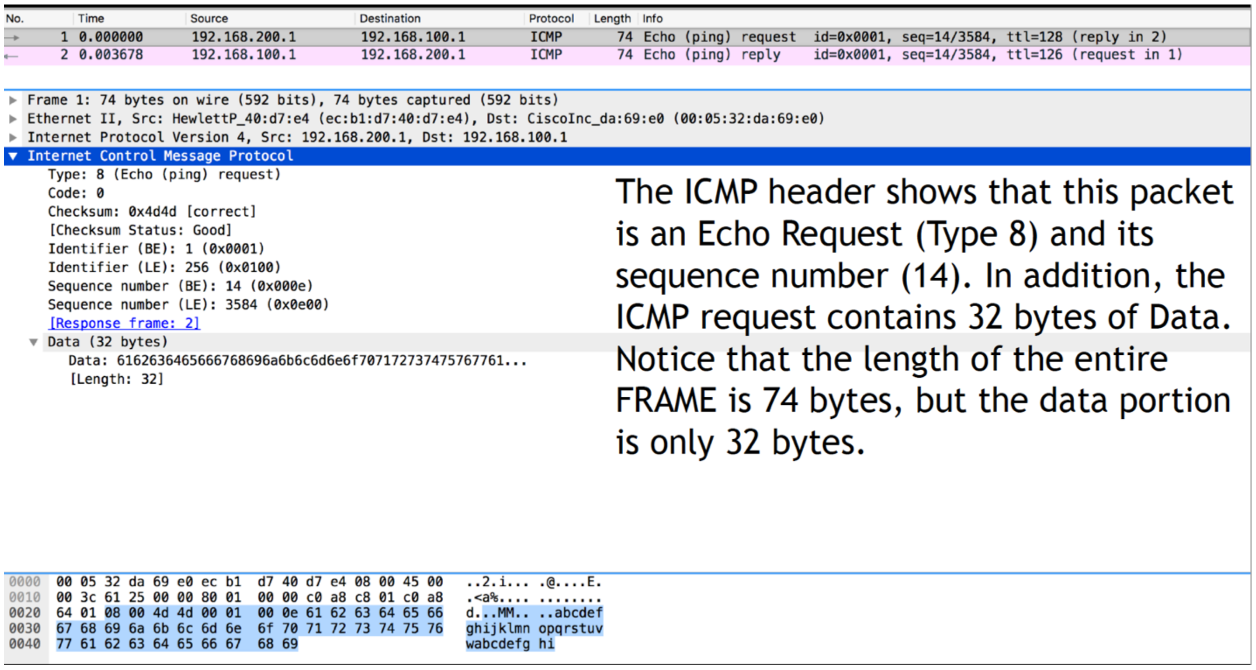
The Ethernet II protocol uses frames to allow devices on the same network to communicate with one another. Like all three protocols we'll discuss here, Ethernet II has a header with a particular consistent format. **In a Wireshark packet capture**, Ethernet II contains a destination MAC address (6 bytes), source MAC address (6 bytes), and a type field (2 bytes), which define where the frame is destined, the device that generated the frame, and the upper layer protocol being used. In total, **the Ethernet II frame is 14 bytes** long.



```
→    1 0.000000     192.168.200.1      192.168.100.1      ICMP     74 Echo (ping) request  id=0x0001, seq=14/3584, ttl=128 (reply in 2)
←    2 0.003678     192.168.100.1      192.168.200.1      ICMP     74 Echo (ping) reply    id=0x0001, seq=14/3584, ttl=126 (request in 1)

▶ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▼ Ethernet II, Src: HewlettP_40:d7:e4 (ec:b1:d7:40:d7:e4), Dst: CiscoInc_da:69:e0 (00:05:32:da:69:e0)
  ▶ Destination: CiscoInc_da:69:e0 (00:05:32:da:69:e0)
  ▶ Source: HewlettP_40:d7:e4 (ec:b1:d7:40:d7:e4)
    Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 192.168.200.1, Dst: 192.168.100.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0x6125 (24869)
  ▶ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: ICMP (1)
    Header checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.200.1
    Destination: 192.168.100.1
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▶ Internet Control Message Protocol

0000  00 05 32 da 69 e0 ec b1  d7 40 d7 e4 08 00 45 00   ..2.i... .@....E.
0010  00 3c 61 25 00 00 80 01  00 00 c0 a8 c8 01 c0 a8   .<a%.... ........
0020  64 01 08 00 4d 4d 00 01  00 0e 61 62 63 64 65 66   d...MM.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

The IP packet contains all the standard IPv4 header fields. Most notably, the Protocol field (1 for ICMP) that indicates the upper layer protocol used, and the Source and Destination IP addresses. Again, the hex for the IPv4 header is highlighted below.

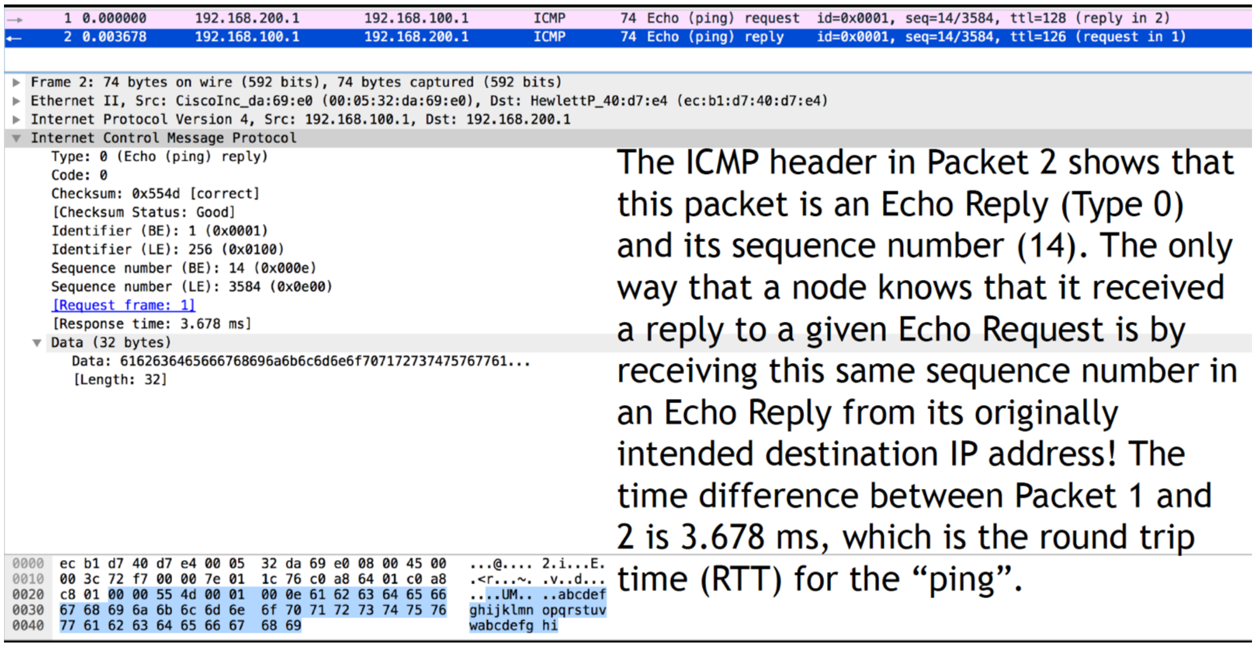
The IP protocol uses packets to transfer between networks and has a header length of **20 bytes**. The fields we care about most for this course are Total Length, Source IP, and Destination IP. The Total Length field denotes the number of bytes in the IP header plus the number of bytes associated with the upper layer protocol, ICMP, in this case.



```
No.   Time        Source           Destination      Protocol  Length  Info
→   1 0.000000    192.168.200.1    192.168.100.1    ICMP        74  Echo (ping) request  id=0x0001, seq=14/3584, ttl=128 (reply in 2)
←   2 0.003678    192.168.100.1    192.168.200.1    ICMP        74  Echo (ping) reply    id=0x0001, seq=14/3584, ttl=126 (request in 1)

▶ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▶ Ethernet II, Src: HewlettP_40:d7:e4 (ec:b1:d7:40:d7:e4), Dst: CiscoInc_da:69:e0 (00:05:32:da:69:e0)
▶ Internet Protocol Version 4, Src: 192.168.200.1, Dst: 192.168.100.1
▼ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4d4d [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 14 (0x000e)
    Sequence number (LE): 3584 (0x0e00)
    [Response frame: 2]
  ▼ Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f7071727374757677661...
      [Length: 32]

0000  00 05 32 da 69 e0 ec b1  d7 40 d7 e4 08 00 45 00   ..2.i... .@....E.
0010  00 3c 61 25 00 00 80 01  00 00 c0 a8 c8 01 c0 a8   .<a%.... ........
0020  64 01 08 00 4d 4d 00 01  00 0e 61 62 63 64 65 66   d...MM.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

The ICMP header shows that this packet is an Echo Request (Type 8) and its sequence number (14). In addition, the ICMP request contains 32 bytes of Data. Notice that the length of the entire FRAME is 74 bytes, but the data portion is only 32 bytes.

Looking further into this Wireshark packet, we see the ICMP header and its associated data, called a payload. The fields we care about in the ICMP header are Type and Sequence Number. In total, the ICMP Header is **8 bytes**. The Type field in this case shows that this is an Echo Request packet since it's **Type 8**, while the Sequence Number indicates which Echo Request this is; 14 in this case (see Sequence Number (BE)). The way the ping program calculates the round trip time between an Echo Request and Echo Reply is it knows the Destination IP of the device receiving the request, the sequence number of the request, and the time that the request occurred at (see the Time column at the top of the packet capture). In this case, a PC with an IP address 192.168.200.1 is sending an Echo Request to a PC with an IP address of 192.168.100.1 with a Sequence Number of 14 at time 0.000000 seconds.



```
→   1 0.000000    192.168.200.1    192.168.100.1    ICMP        74  Echo (ping) request  id=0x0001, seq=14/3584, ttl=128 (reply in 2)
←   2 0.003678    192.168.100.1    192.168.200.1    ICMP        74  Echo (ping) reply    id=0x0001, seq=14/3584, ttl=126 (request in 1)

▶ Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▶ Ethernet II, Src: CiscoInc_da:69:e0 (00:05:32:da:69:e0), Dst: HewlettP_40:d7:e4 (ec:b1:d7:40:d7:e4)
▶ Internet Protocol Version 4, Src: 192.168.100.1, Dst: 192.168.200.1
▼ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x554d [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 14 (0x000e)
    Sequence number (LE): 3584 (0x0e00)
    [Request frame: 1]
    [Response time: 3.678 ms]
  ▼ Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f7071727374757677661...
      [Length: 32]

0000  ec b1 d7 40 d7 e4 00 05  32 da 69 e0 08 00 45 00   ...@.... 2.i...E.
0010  00 3c 72 f7 00 00 7e 01  1c 76 c0 a8 64 01 c0 a8   .<r...~. .v..d...
0020  c8 01 00 00 55 4d 00 01  00 0e 61 62 63 64 65 66   ....UM.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

The ICMP header in Packet 2 shows that this packet is an Echo Reply (Type 0) and its sequence number (14). The only way that a node knows that it received a reply to a given Echo Request is by receiving this same sequence number in an Echo Reply from its originally intended destination IP address! The time difference between Packet 1 and 2 is 3.678 ms, which is the round trip time (RTT) for the "ping".

This next packet capture shows the Echo Reply (ICMP Type 0) from 192.168.100.1 (see its Source IP) back to the originator of the request, 192.168.200.1 (Destination IP), with a Sequence Number of 14, which was received at time 0.003678. Taking the time difference between the time of the Echo Reply and the time of the Echo Request, we conclude that this “ping” had a round trip time (RTT) of 3.678 ms. To get milliseconds (ms), we multiply the time difference by 1000.

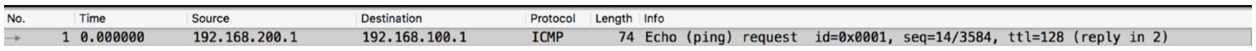
**Computing Packet Capture Analysis Metrics**

Packet Capture Analysis begins with computing some common metrics that help quantify network performance and activity. We may want to compute things like round trip times between devices, number of hops (distance) between device, and network bandwidth utilization (throughput).

In this lesson, we already talked about how to compute RTT. We’re also going to look at a few simple, yet powerful, metrics related to a given protocol, called data size metrics. For a given protocol, ICMP in our case, we can calculate things like number of packets sent/received, total bytes sent/received, and total data sent/received. These computations are performed for a certain *type* of packet and *direction* of packet, for example Echo Requests sent, Echo Requests received, Echo Replies sent, Echo Replies received.

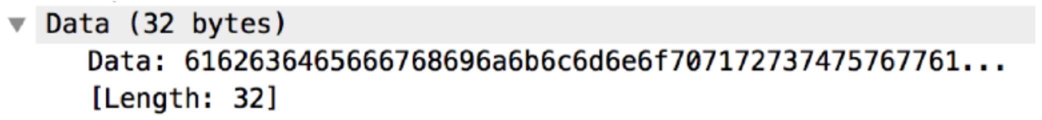
The number of packets sent is pretty self-explanatory. For a given packet capture file, if we wanted to determine how many Echo Requests that a device sent during that packet capture, we’d count up how many packets there were where this device’s IP address was the Source IP Address for an ICMP Echo Request (Type 8).

If we wanted to know how many **bytes** were sent, we’d have to look at the total size of the frame containing each of those Echo Requests that the device sent.



```
No.     Time        Source          Destination     Protocol  Length  Info
  1     0.000000    192.168.200.1   192.168.100.1   ICMP          74  Echo (ping) request  id=0x0001, seq=14/3584, ttl=128 (reply in 2)
```

The total number of bytes of the frame is indicated by the Length column above; this particular Wireshark packet has a frame length of 74 bytes. This information is **not directly** available within the packet capture. The way you compute this quantity is using the IP protocol’s Total Length field. Recall that when we looked at the packet’s Total Length field, it was 60 bytes. These 60 bytes include the entire IP header as well as the ICMP header and ICMP payload. To compute the number of bytes for this frame, we add the Total Length field of 60 bytes plus the size of the Ethernet header, which is 14 bytes, for a total of 74 bytes.



```
▼ Data (32 bytes)
    Data: 6162636465666768696a6b6c6d6e6f7071727374757677 61...
    [Length: 32]
```

Similarly, to compute the total amount of Echo Request **data** sent, we’re referring to the amount of data in the ICMP payload. In the packet capture, we can see that there is 32 bytes of data. Just like the total frame size, this information is not directly available from the packet capture. Instead, we use IP protocol’s Total Length field once again. To compute how much data is in ICMP Echo Request (or Reply) packet, we total the Total Length field and subtract from it the length of the IP Header (20 bytes) and the length of the ICMP Header (8 bytes). 60 – 20 – 8 gives us the 32 bytes of data.