CS 2690 Packet Sniffer Lab
Using Wireshark v3.0.7 and Npcap v0.9983

Lab Exercise 1: IP Packet Fragmentation

Approximate time required: 1.5 hours

Lab Grading Policy

The CS 2690 lab exercises may be completed in the Network Lab in CS 516 or, in most cases, on your own system. *To receive credit for lab exercises, lab reports must be submitted in word processed format, by the date and time specified in Canvas.* No credit will be given unless these requirements are met. Late lab exercises and lab exercises left in my mailbox or emailed to me will not be graded.

Lab Reports are graded on a 10 point scale, as follows:

Lab report meets all requirements	10 points
Minor errors and/or omissions	8 – 9
Significant errors and/or omissions	4 – 7
Majority of lab report is incorrect or missing	0 – 3
Lab not submitted, or virtually the same as another student's submission	0

Each student must perform their own lab work. If multiple lab reports are submitted that contain essentially the same content, no credit will be granted. When weighted, each lab assignment will be worth approximately 2.5% of your semester grade.

Introduction

In Computer Networks I, we did several Wireshark labs that demonstrated how to capture packets and filter for certain specific protocols, including IPv4. This lab exercise will continue those activities by showing you how to capture and interpret IPv4 packet *fragments*. Read the relevant sections in the text (pp. 203 - 213) before starting this lab. You'll want to have the Lecture 2 PowerPoint slide titled "Figure 3.16 IPv4 Packet Header" available for reference. Note that a fairly complete Wireshark User's Guide is available from within Wireshark (**Help | Contents**). You can use a WiFi network for this lab, but Ethernet is preferable if you have the option.

Installing Wireshark on Your Own Computer

If you will be running Wireshark on your own computer (highly recommended), use the built in Wireshark update function, or browse to `https://www.wireshark.org/download.html`. Locate "Old Stable Release (3.0.7)" and download the appropriate installer for your operating system. This lab exercise was tested with that version. You can use the default settings during installation. Windows installations will include the new Npcap driver, which replaces WinPcap. Running Wireshark on macOS or Linux will require the libpcap driver, which comes pre-installed with macOS and most Linux distributions. We will use this version of Wireshark for the remainder of the semester. *v3.2.0 is unstable on Windows 10, so when Wireshark asks you if you want to update to v3.2.0, "just say no".*

Running Wireshark in the Network Lab (CS 516)

You may also complete this lab exercise using the Windows workstations in the Network Lab (CS 516). The 8 workstations in the 2nd and 3rd rows in CS 516 are configured with Wireshark, Riverbed Modeler and Microsoft Office. The student accounts on the lab machines have administrator privileges. Login with your regular UVID and UVU password. The lab computers are incapable of storing your work after reboot, so bring a flash drive if you anticipate needing to save off a partially completed assignment.

During the first week of class all registered students were added to the electronic lock on the CS 516 door. To enter CS 516, wave your UVU I.D. card over the sensor with the red LED, wait for the LED to turn green, and turn the door handle. Old-style UVU I.D. cards won't work with electronic locks, so you may need to visit Campus Connection to obtain the newer card. If you added the class after the access list was created and can't access CS 516, send an email with your name, UVID, course, and section numbers to `sayeed.sajal@uvu.edu` to be added to the access list. This can take 2 – 3 days.

If you plan to use the computers in CS 516 to complete your lab exercises, test your UVU I.D. card now to make sure that it opens the door. Also note that CS 516 can occasionally be crowded, so be sure to complete the lab well before the deadline to prevent last minute access problems. Inability to access CS 516 is not a valid excuse for late submissions.

IPv4 Review

The first part of this lab is a brief review of what you did in the final CS 2600 lab. If you don't remember much about the IPv4 header, complete the optional **Generating and Capturing Web Traffic** and **Display Filters** sections below, as a quick review. Otherwise, skip to the IP **Fragmentation Experiment** section, where the new material begins.

Generating and Capturing Web Traffic

Start Wireshark. Prepare Wireshark for packet capture (**Capture | Options | Input** tab), select the appropriate interface, and click **Start**. (If you aren't sure which interface to select, find the network type being used (Ethernet/Wi-Fi) and then look for indications of traffic in the "Traffic" column.) Point your browser to a relatively simple website with a URL that starts with "`http://`" (not "`https://`"). `http://www.oidview.com/` or `http://desource.uvu.edu` `will work`. Download a page, and then stop the Wireshark capture (**Capture | Stop**). After you stop the capture you should see packets carrying a variety of upper-layer protocols displayed in the upper (*Packet List*) pane.

If you wish to save the captured packets to a file (optional), choose **File | Save As…**, browse to the desired folder, enter a file name in the *File name:* field and click **Save**.

Click **Statistics | Capture File Properties** at the top to review some general information about the capture. Note that the average data transmission rate is displayed at the bottom in Bps and bps.

Close the "Wireshark: Capture File Properties" window, then click **Statistics | Protocol Hierarchy**. This window shows the percentages of packets and bytes captured for each protocol at various layers (Internet, Transport, etc.) Because not all IPv4 packets contain TCP segments, the percentage of TCP packets will be lower than the percentage of IPv4 packets. Viewing this window after running a long-term capture in promiscuous mode would provide some insight into the different types of traffic being transmitted over your local network. (Note that 802.11 traffic will be classified as "Ethernet", even though you are using a WiFi adapter.)

Statistics | Conversations shows you a breakout of point-to-point traffic flow between pairs of MAC addresses, IP addresses, TCP connections or UDP flows. The list displays one row for each unique conversation, and displays total number of packets/bytes captured as well as number of packets/bytes in each direction. **Statistics | Endpoints** provides similar information for individual endpoints.

Statistics | IO Graph provides a graph of packets or bytes per specified time interval.

The middle section of the **Statistics** menu provides many protocol-specific tools for analyzing traffic. **Statistics | Flow Graph** provides a useful vertical time-sequence diagram showing traffic flow between various nodes (best viewed in full-screen mode). Click on a particular transaction (represented by an arrow) to highlight the corresponding packet (in gray, by default) in the *Packet List* pane of the main Wireshark display.

Statistics | Packet Lengths provides a count of the number of packets captured (or just the ones allowed by the display filter), grouped by size.

Finally, the various **Statistics | TCP Stream Graph** options allow you to analyze the performance characteristics of TCP connections, based on response time (latency), throughput, etc. A TCP message must be selected in the *Packet List* pane to enable this option.

Using a Display Filter

Next, we need to find the first packet returned to your computer from the web server that you browsed to earlier. You could just scroll through the *Packet List* pane and look for it, but when a lot of traffic is captured, this can be tedious. Instead, we'll create a display filter that shows only the web (HTTP) traffic originating from that web server. (**H**yper**T**ext **T**ransport **P**rotocol is the Application layer protocol used by web browsers and servers.) We want to view HTTP packets only, and include only packets with the server's IP address in the IP header's "Source Address" field.

Use `nslookup` (Windows command line) or `dig` (Linux/macOS terminal) to determine the IP address of the web server, by typing something like the following (depending on your OS and which web server you queried earlier):

`nslookup oidview.com` (on Windows)

`dig oidview.com +noall +answer` (on Linux and macOS) `nslookup` will typically return at least two IP addresses. The first will be the address of your local domain name server, often located at your ISP (or on campus if you are at UVU). This domain name server responded to your browser's request for the IP address that matches the domain name you were browsing to (e.g., `oidview.com`). The "Non-authoritative answer" is the IP address (or addresses) of the target web server (e.g, `208.84.118.115`). You may need to skip over that server's IPv6 address (or addresses) to find it's IPv4 address(es). IPv6 addresses are displayed in hexadecimal and separated by colons, as in `2400:cb00:2048:1::6814:155`.)

The version of `dig` shown above will return the ANSWER SECTION of the DNS record only. The IP address that you are after should be listed at the right. If `dig` doesn't work as described, check the man page for your distribution.

If `nslookup` or `dig` doesn't work, try `ping oidview.com` and note the address in the reply.

Write down the target server's IPv4 address.

Now, where were we? We were finding the first packet returned to your computer from the web server that you browsed to earlier.

Let's create the display filter. Back in Wireshark, click **Analyze | Display Filters…**, click the **+** button at the lower left, and enter the name of your new filter in the *New display filter* field at the bottom (something like "Responses from oidview server").

Next we'll determine the combination of filter elements that we need to construct the display filter string (expression). Since we want the first packet returned by the web server, we are looking for packets where the source address is the web server (not your browser) and which contain HTTP messages.

Here's a sample filter string: `ip.src == 208.84.118.115 && http`

Enter your filter string into the *Filter* column at the right, and note that the background color turns from pink to green when the field contains a valid filter string. Click **OK** to save the filter.

Enable your new filter by clicking the **Filter Bookmark** button near the upper left corner of the screen and choosing the name of your filter from the display filter list. You should see the corresponding filter string displayed with a green background in the display filter field near the top.

Confirm that your display filter is working by viewing the *Packet List* pane, which should be displaying one or more HTTP packets, showing the server's IP address in the *Source* column. The packet that we're interested in is the first HTTP packet returned by the server, which will usually (but not always) be the topmost packet displayed in the *Packet List* pane when your filter is active. It will say "HTTP" in the *Protocol* column. Highlight this packet in the *Packet List* pane to display the details of its contents in the *Packet Details* (middle) pane.

In the *Packet Details* (middle) pane, click the **>** icon to expand the Internet Protocol (IP) header. Enlarge the *Packet Details* (middle) pane as needed to view the contents of the IP header. Verify that you can match the IP header fields shown in the Wireshark *Packet List* pane with the lecture 1 slide titled **IPv4 Packet Header** (figure 3.16 in the textbook).

IP Fragmentation Experiment

If you skipped the previous two sections, start Wireshark.

In this section, we'll perform a simple experiment with packet fragmentation. We're going to force IPv4 to fragment a packet using the `ping` command (Echo Request), which is one function of the ICMP protocol.

File | Close the previous capture file, if necessary. Find the IP address or domain name of some other computer to use as the target of the ping. If possible, choose another computer on your own network, for example, your home router (which may be `192.168.1.1`).

> To enhance security, Microsoft Windows computers and some home router/access point devices may be configured *not* to respond to a fragmented ping request, even if they do respond to a normal ping. Most Linux distributions *will* respond to a fragmented ping.

To determine the IP address of one of your own computers, type `ipconfig` from the command line *on that computer* (use `ifconfig` in Linux and macOS). In the Network Lab, you can ping the lab router at `161.28.108.254`.

Do a quick test to verify that your target will respond to fragmented ICMP ping requests. Your target may be an IP address or a domain name. Open a command line window on the source (non-target) computer, and type the following (the `–l` option is a lower case "L" for "length", not the digit "1"):

Windows: `ping –l 3000 <target>`

Linux & macOS: `ping –s 3000 <target>`

In some Linux versions and configurations, you may need to be superuser to execute this version of ping. If so, try this, and enter the root password when prompted:

```
sudo ping –s 3000 <target>
```

Verify that at least some of the responses are "Reply from …" rather than "Request timed out" or another message indicating no response from the target.

If your ping test fails (usually with a "Request timed out." message), try pinging a different target. Servers that currently respond to fragmented pings include `uvu.edu`, `ietf.org`, `iana.org`, and `icann.org`.

If you receive a reply with a longer address, like this…

```
Reply from fe80::dcec:2db6:1c32:522a: time100ms
```

…it means that your ping used an IPv6 address (this can happen with Xfinity). We want to use IPv4 here, so retry the ping using the following syntax, to force the use of IPv4:

`ping -4 –l 3000 google.com` (Windows only)

(If you are using Windows, you will need to remember to add the `-4` when you use ping in future lab exercises. Or you can temporary disable IPv6 in **Control Panel | Network and Sharing Center**. Click **Change adapter settings**, right click on the adapter you are using, click **Properties**, and temporarily uncheck **Internet Protocol Version 6**.)

If you are running a software ("personal") firewall on the source computer and filtering outbound traffic (not usually the default mode), that firewall may detect the fragmented pings (incorrectly) as an outbound "Jolt" attack. Jolt is a denial of service attack that sends a large number of identical illegally fragmented ICMP ping or UDP packets, effectively preventing the targeted machine from running other processes. If your firewall is configured to block the outbound fragmented pings, you may need to disable it temporarily.

After you have located a responsive target, test the limits of fragmentation. On Ethernet, the MTU (the maximum frame payload) is 1500 bytes. With a ping message, the payload of the Ethernet frame is an ICMP ping message. So this payload includes a 20-byte IP header, followed by an 8-byte ICMP Echo Request header, followed by some ICMP "dummy" data (see Figure 1 below). In this case, there is no TCP or UDP header. Instead, the 8-byte ICMP header occupies the position where the TCP or UDP header would normally begin. The `-l` option in the Windows `ping` command specifies the number of "dummy" data bytes in the ICMP message. Thus, to exactly fill the 1500 byte MTU in an Ethernet frame, we'll want to specify `-l 1472` (20 + 8 + 1472 = 1500). Note that the length of the Ethernet header is *not included* in the MTU byte count, because MTU describes the length of the Ethernet frame *payload*, beginning with the IP header.

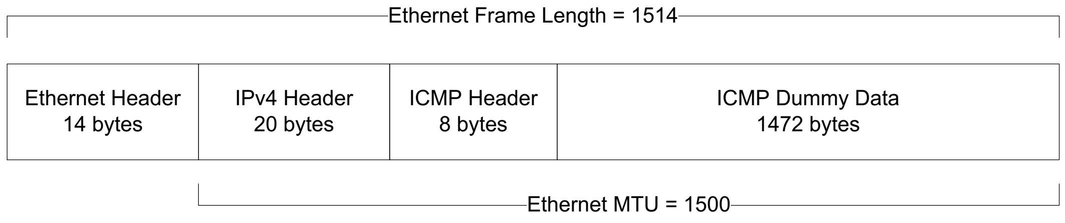


Figure 1. ICMP ping Message Format

If you are running Wireshark on an 802.11 WiFi network, the actual MTU is typically around 2312 bytes, depending on the type of WiFi security being used. However, Wireshark emulates Ethernet in this situation, so the 1500 byte MTU size will still apply.

Close your web browser. In Wireshark, cancel your display filter and start a Wireshark **Capture.**

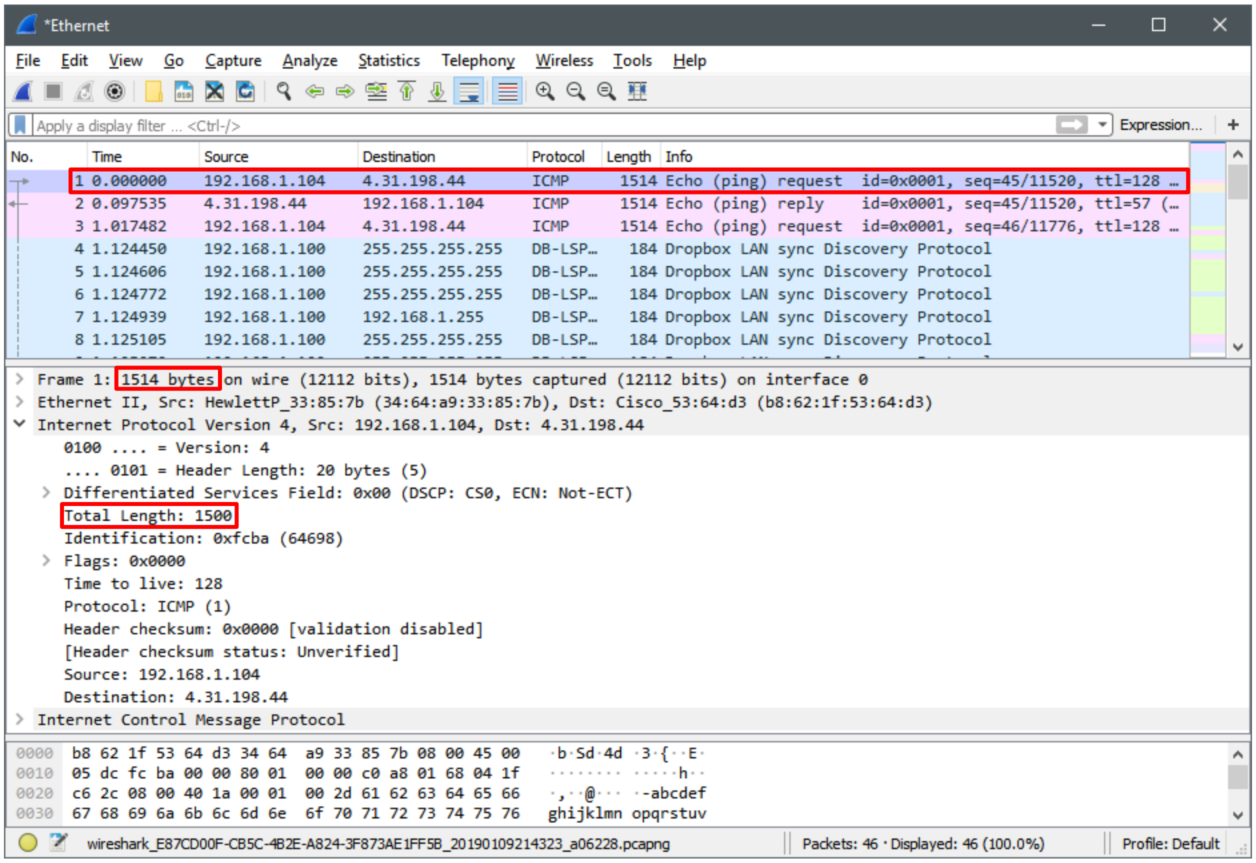
In a command line window, type one of the following…

Windows: `ping -l 1472 <target>`

Linux & macOS: `ping -s 1472 <target>`

…where `<target>` is the IP address or domain name of the other computer. After you've received some responses from the target, stop the capture.

With no filters enabled, locate one of the ICMP Echo (ping) **request** packets and select it. Your *Packet List* pane should look something Figure 2 below. Note that the ICMP "dummy" data in the Packet Bytes pane at the bottom is lower case ASCII letters in alphabetical order. The *overall* length of the ping messages is shown as 1514 bytes, because it includes the Ethernet header. The IP packet length is 1500 bytes, exactly matching the Maximum Transmission Unit (MTU) size of the Ethernet frame.


*Ethernet
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.  Time      Source          Destination      Protocol Length Info
1 0.000000   192.168.1.104   4.31.198.44      ICMP     1514 Echo (ping) request  id=0x0001, seq=45/11520, ttl=128 …
2 0.097535   4.31.198.44     192.168.1.104    ICMP     1514 Echo (ping) reply    id=0x0001, seq=45/11520, ttl=57 (…
3 1.017482   192.168.1.104   4.31.198.44      ICMP     1514 Echo (ping) request  id=0x0001, seq=46/11776, ttl=128 …
4 1.124450   192.168.1.100   255.255.255.255  DB-LSP…   184 Dropbox LAN sync Discovery Protocol
5 1.124606   192.168.1.100   255.255.255.255  DB-LSP…   184 Dropbox LAN sync Discovery Protocol
6 1.124772   192.168.1.100   255.255.255.255  DB-LSP…   184 Dropbox LAN sync Discovery Protocol
7 1.124939   192.168.1.100   192.168.1.255    DB-LSP…   184 Dropbox LAN sync Discovery Protocol
8 1.125105   192.168.1.100   255.255.255.255  DB-LSP…   184 Dropbox LAN sync Discovery Protocol

> Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
> Ethernet II, Src: HewlettP_33:85:7b (34:64:a9:33:85:7b), Dst: Cisco_53:64:d3 (b8:62:1f:53:64:d3)
v Internet Protocol Version 4, Src: 192.168.1.104, Dst: 4.31.198.44
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0xfcba (64698)
  > Flags: 0x0000
    Time to live: 128
    Protocol: ICMP (1)
    Header checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.1.104
    Destination: 4.31.198.44
> Internet Control Message Protocol

0000  b8 62 1f 53 64 d3 34 64  a9 33 85 7b 08 00 45 00   ·b·Sd·4d ·3·{··E·
0010  05 dc fc ba 00 00 80 01  00 00 c0 a8 01 68 04 1f   ········ ·····h··
0020  c6 2c 08 00 40 1a 00 01  00 2d 61 62 63 64 65 66   ·,··@··· ·-abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv

wireshark_E87CD00F-CB5C-4B2E-A824-3F873AE1FF5B_20190109214323_a06228.pcapng | Packets: 46 · Displayed: 46 (100.0%) | Profile: Default
```

Figure 2. Screen Shot of Non-fragmented ping Messages

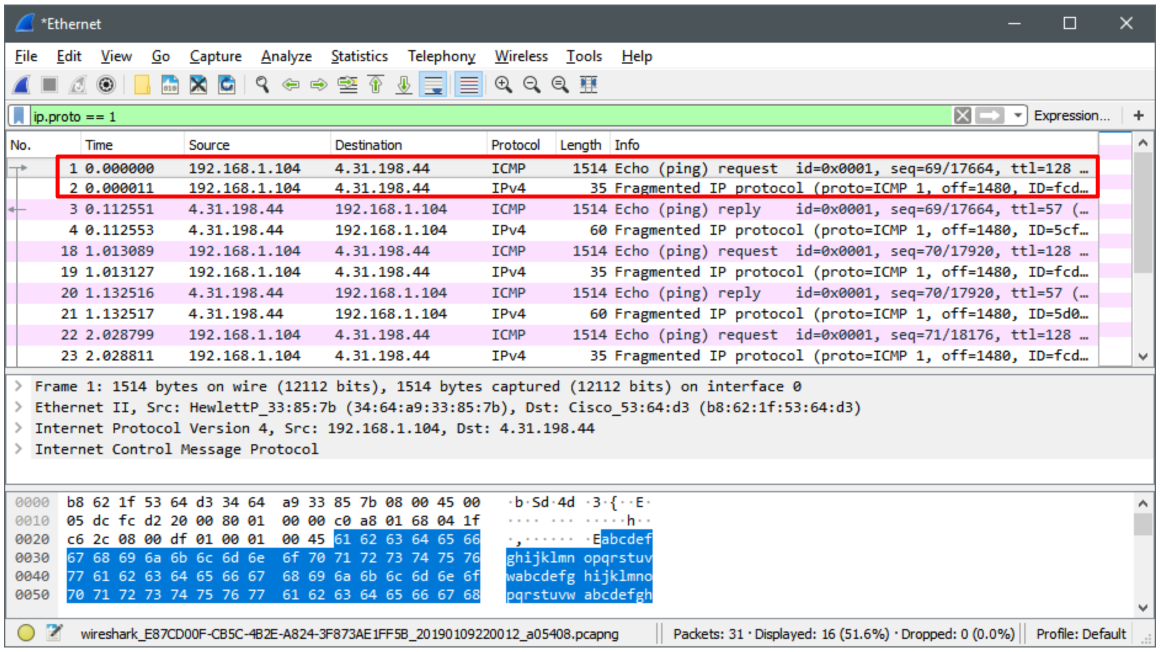
There is no need to save this capture file.

By default, Wireshark will automatically reassemble fragments and display the original unfragmented packet. To view the *individual fragments* in Wireshark (as they would be transmitted over the Internet), choose **Edit | Preferences**. Expand **Protocols** at the left, scroll down and select *IPv4*, uncheck *Reassemble fragmented IPv4 datagrams*, and click **OK**.

Now try another Wireshark **Capture**, using `ping -l 1473` or `ping -s 1473`. This will push the MTU over the fragmentation threshold by one byte. After you've received your responses from the target, stop the capture. If the first ping response in your command line/terminal window is a "Request timed out.", discard this capture file and re-run the capture.

Your *Packet List* pane will probably be cluttered with a lot of non-ICMP packets, so use this display filter to show only the ICMP packets and their fragments: `ip.proto == 1`.

With one of the Echo (ping) **request** packets selected, your *Packet List* pane for the second capture should look something like Figure 3 below.



```
ip.proto == 1
```

```
No.  Time      Source         Destination    Protocol Length Info
   1 0.000000  192.168.1.104  4.31.198.44    ICMP     1514 Echo (ping) request  id=0x0001, seq=69/17664, ttl=128 …
   2 0.000011  192.168.1.104  4.31.198.44    IPv4       35 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fcd…
   3 0.112551  4.31.198.44    192.168.1.104  ICMP     1514 Echo (ping) reply    id=0x0001, seq=69/17664, ttl=57 (…
   4 0.112553  4.31.198.44    192.168.1.104  IPv4       60 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=5cf…
  18 1.013089  192.168.1.104  4.31.198.44    ICMP     1514 Echo (ping) request  id=0x0001, seq=70/17920, ttl=128 …
  19 1.013127  192.168.1.104  4.31.198.44    IPv4       35 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fcd…
  20 1.132516  4.31.198.44    192.168.1.104  ICMP     1514 Echo (ping) reply    id=0x0001, seq=70/17920, ttl=57 (…
  21 1.132517  4.31.198.44    192.168.1.104  IPv4       60 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=5d0…
  22 2.028799  192.168.1.104  4.31.198.44    ICMP     1514 Echo (ping) request  id=0x0001, seq=71/18176, ttl=128 …
  23 2.028811  192.168.1.104  4.31.198.44    IPv4       35 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fcd…

> Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
> Ethernet II, Src: HewlettP_33:85:7b (34:64:a9:33:85:7b), Dst: Cisco_53:64:d3 (b8:62:1f:53:64:d3)
> Internet Protocol Version 4, Src: 192.168.1.104, Dst: 4.31.198.44
> Internet Control Message Protocol

0000  b8 62 1f 53 64 d3 34 64  a9 33 85 7b 08 00 45 00   ·b·Sd·4d ·3·{··E·
0010  05 dc fc d2 20 00 80 01  00 00 c0 a8 01 68 04 1f   ···· ··· ·····h··
0020  c6 2c 08 00 df 01 00 01  00 45 61 62 63 64 65 66   ·,······ ·Eabcdef
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 6a 6b 6c 6d 6e 6f   wabcdefg hijklmno
0050  70 71 72 73 74 75 76 77  61 62 63 64 65 66 67 68   pqrstuvw abcdefgh

wireshark_E87CD00F-CB5C-4B2E-A824-3F873AE1FF5B_20190109220012_a05408.pcapng || Packets: 31 · Displayed: 16 (51.6%) · Dropped: 0 (0.0%) || Profile: Default
```

Figure 3. Screen Shot of Fragmented ping Messages

When ICMP fragmentation occurs, Wireshark displays two different types of output in the *Packet List* pane's Protocol column, "ICMP" packets and "IPv4" packets. Wireshark is displaying the *first* of the two fragments as "ICMP" traffic (frame no. 1 in Figure 3 above).

The following "IPv4" packet (frame no. 2) contains the second fragment, which includes an IP header (as always) and the one remaining byte of ICMP "dummy" data. The second fragment does *not* contain a duplicate ICMP header, so Wireshark labels it (somewhat ambiguously) as an IPv4 packet.

Normally you could use `icmp` as your display filter, but that won't show you the second fragment because that one is missing the ICMP header that the `icmp display` filter is looking for. Instead, by using the `ip.proto == 1` display filter, you have requested the display of any IP packet that has it's **Protocol** field set to 1, indicating ICMP. Both fragments will use this Protocol code, since both are carrying part of an ICMP message.

Make a screen shot of the second capture (containing the ICMP fragments), with one of the ICMP Echo (ping) requests highlighted, similar to the example above. Include this screen shot in your Lab Report.

Questions

1. In the *Packet List* pane, select the first ICMP Echo (ping) request packet. Expand the Internet Protocol header in the *Packet Details* pane. Then expand the *Flags* field. Which two IP header elements and corresponding values indicate that this is the *first* of multiple fragments?

2. Select the *second* fragment of the same IP packet. What *Fragment Offset* is shown? (Drill down into the **Flags** field to find this.) What is the actual Fragment Offset in bytes? Be careful here, and refer to the IP Packet Header slide.)

3. How can you verify that these two fragments were originally part of the same packet?

4. Add up the Total Length values of the two fragments from the first ICMP request. Does this equal the 1473 byte length specified in the ping request? What is the explanation for this? (Hint: an unfragmented ping request packet has a 20 byte IP header and an inner 8 byte ICMP header.)

The problem of piecing together fragments that we've dealt with in this section is similar to the problem faced by a firewall that is attempting to detect malicious traffic. The firewall may be programmed to look for various patterns in the IP or Transport headers, or it may look for malicious code in the packet payload (similar to a virus scanner). Attackers may try to fool the firewall by intentionally fragmenting their attacks, so that the known malicious data patterns are distributed across several packet fragments. This means that the firewall may need to reassemble packet fragments in order to detect an attack, violating the general IPv4 fragmentation policy which dictates that fragments are only reassembled when they reach the final destination computer. It also means that a security analyst attempting to understand the nature of an attack may need to view suspicious packets in both fragmented and reassembled form.

Summary of Deliverables

- Screen shot from *IP Fragmentation Experiment* with one of the ICMP Echo (ping) requests highlighted
- Answers to *Questions 1 - 4*

Submit the lab report in Canvas in word processed format. If you are working in the lab, log off and shut down your workstation. If you are working at home and have modified your firewall settings, *don't forget to restore them.* If you disabled IPv6, enable it in Control Panel.