# **Capturing Traffic with Wireshark**

Prepared for UCC Networking Course, June - August 2012.

# Purpose of this lab:

The purpose of this lab is to gain practical experience with the Wireshark traffic analyzer which allows the observation of network traffic.

### Software Tools:

- Wireshark traffic analyzer.
- Wireshark is installed on the computers of the classroom.

### What to turn in:

- A report with your answers to the questions in this lab, including the plots, copies of your MATLAB code, and the anonymous feedback form.
- The symbol

indicates questions for the lab report.

- The lab exercises and the lab reported are to be completed individually.
- The estimated time to complete the lab is 3 hours.

This lab uses

- Wireshark Lab: Getting Started v6.0
- Wireshark Lab: Ethernet and ARP v6.01
- Wireshark Lab: ICMP v6.0

Supplement to *Computer Networking: A Top-Down Approach*, 6<sup>th</sup> ed., J.F. Kurose and K.W. Ross, © 2005-21012, J.F Kurose and K.W. Ross, All Rights Reserved

# Part 1. Getting Started with Wireshark

One's understanding of network protocols can often be greatly deepened by "seeing protocols in action" and by "playing around with protocols" – observing the sequence of messages exchanged between two protocol entities, delving down into the details of protocol operation, and causing protocols to perform certain actions and then observing these actions and their consequences. This can be done in simulated scenarios or in a "real" network environment such as the Internet. In the Wireshark labs you'll be doing in this course, you'll be running various network applications in different scenarios using your own computer (or you can borrow a friends; let me know if you don't have access to a computer where you can install/run Wireshark). You'll observe the network protocols in your computer "in action," interacting and exchanging messages with protocol entities executing elsewhere in the Internet. Thus, you and your computer will be an integral part of these "live" labs. You'll observe, and you'll learn, by doing.

In this first Wireshark lab, you'll get acquainted with Wireshark, and make some simple packet captures and observations.

The basic tool for observing the messages exchanged between executing protocol entities is called a **packet sniffer**. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a *copy* of packets that are sent/received from/by application and protocols executing on your machine.

Figure 1 shows the structure of a packet sniffer. At the right of Figure 1 are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniffer, shown within the dashed rectangle in Figure 1 is an addition to the usual software in your computer, and consists of two parts. The **packet capture library** receives a copy of every link-layer frame that is sent from or received by your computer. Recall from the discussion from section 1.5 in the text (Figure 1.24<sup>1</sup>) that messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. In Figure 1, the assumed physical media is an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent/received from/by all protocols and applications executing in your computer.

<sup>&</sup>lt;sup>1</sup> References to figures and sections are for the  $6^{th}$  edition of our text, *Computer Networks, A Top-down Approach,*  $6^{th}$  ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2012.



Figure 1: Packet sniffer structure

The second component of a packet sniffer is the **packet analyzer**, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer must "understand" the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol in Figure 1. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment. Finally, it understands the HTTP protocol and so, for example, knows that the first bytes of an HTTP message will contain the string "GET," "POST," or "HEAD," as shown in Figure 2.8 in the text.

We will be using the Wireshark packet sniffer [http://www.wireshark.org/] for these labs, allowing us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, Wireshark is a packet analyzer that uses a packet capture library in your computer). Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers. It's an ideal packet analyzer for our labs – it is stable, has a large user base and well-documented support that includes a user-guide (http://www.wireshark.org/docs/wsug\_html\_chunked/), man pages (http://www.wireshark.org/docs/man-pages/), and a detailed FAQ (http://www.wireshark.org/faq.html), rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface. It operates in computers using Ethernet, serial (PPP and SLIP), 802.11 wireless LANs, and many other link-layer technologies (if the OS on which it's running allows Wireshark to do so).

# **Getting Wireshark**

In order to run Wireshark, you will need to have access to a computer that supports both Wireshark and the *libpcap* or *WinPCap* packet capture library. The *libpcap* software will be installed for you, if it is not installed within your operating system, when you install Wireshark. See <u>http://www.wireshark.org/download.html</u> for a list of supported operating systems and download sites

Download and install the Wireshark software:

• Go to <u>http://www.wireshark.org/download.html</u> and download and install the Wireshark binary for your computer.

The Wireshark FAQ has a number of helpful hints and interesting tidbits of information, particularly if you have trouble installing or running Wireshark.

# Running Wireshark

When you run the Wireshark program, you'll get a startup screen, as shown below:



Figure 2: Initial Wireshark Screen

Take a look at the upper left hand side of the screen – you'll see an "Interface list". This is the list of network interfaces on your computer. Once you choose an interface, Wireshark will capture all packets on that interface. In the example above, there is an Ethernet interface (Gigabit network Connection) and a wireless interface ("Microsoft").

If you click on one of these interfaces to start packet capture (i.e., for Wireshark to begin capturing all packets being sent to/from that interface), a screen like the one below will

be displayed, showing information about the packets being captured. Once you start packet capture, you can stop it by using the Capture pull down menu and selecting Stop.



Figure 3: Wireshark Graphical User Interface, during packet capture and analysis

The Wireshark interface has five major components:

- The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data, and exit the Wireshark application. The Capture menu allows you to begin packet capture.
- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is *not* a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest-level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.
- The **packet-header details window** provides details about the packet selected (highlighted) in the packet-listing window. (To select a packet in the packet-listing window, place the cursor over the packet's one-line summary in the packet-listing window and click with the left mouse button.). These details

include information about the Ethernet frame (assuming the packet was sent/received over an Ethernet interface) and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the plus minus boxes to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest-level protocol that sent or received this packet are also provided.

- The **packet-contents window** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.
- Towards the top of the Wireshark graphical user interface, is the **packet display filter field**, into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows). In the example below, we'll use the packet-display filter field to have Wireshark hide (not display) packets except those that correspond to HTTP messages.

# Taking Wireshark for a Test Run

The best way to learn about any new piece of software is to try it out! We'll assume that your computer is connected to the Internet via a wired Ethernet interface. Indeed, I recommend that you do this first lab on a computer that has a wired Ethernet connection, rather than just a wireless connection. Do the following

- 1. Start up your favorite web browser, which will display your selected homepage.
- 2. Start up the Wireshark software. You will initially see a window similar to that shown in Figure 2. Wireshark has not yet begun capturing packets.
- 3. To begin packet capture, select the Capture pull down menu and select *Interfaces*. This will cause the "Wireshark: Capture Interfaces" window to be displayed, as shown in Figure 4.

Wireshark: Capture Interfaces				
Description	IP	Packets	Packets/s	Stop
📄 🗩 Intel(R) 82567LM Gigabit Network	Connection fe80::844a:1239:ab19:74e	a 28	0	Start Options Details
🗩 Microsoft	192.168.1.100	17	0	Start Options Details
Help				<u>C</u> lose

Figure 4: Wireshark Capture Interface Window

4. You'll see a list of the interfaces on your computer as well as a count of the packets that have been observed on that interface so far. Click on *Start* for the interface on which you want to begin packet capture (in the case, the Gigabit

network Connection). Packet capture will now begin - Wireshark is now capturing all packets being sent/received from/by your computer!

- 5. Once you begin packet capture, a window similar to that shown in Figure 3 will appear. This window shows the packets being captured. By selecting *Capture* pulldown menu and selecting *Stop*, you can stop packet capture. But don't stop packet capture yet. Let's capture some interesting packets first. To do so, we'll need to generate some network traffic. Let's do so using a web browser, which will use the HTTP protocol that we will study in detail in class to download content from a website.
- 6. While Wireshark is running, enter the URL: <u>http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html</u> and have that page displayed in your browser. In order to display this page, your browser will contact the HTTP server at gaia.cs.umass.edu and exchange HTTP messages with the server in order to download this page, as discussed in section 2.2 of the text. The Ethernet frames containing these HTTP messages (as well as all other frames passing through your Ethernet adapter) will be captured by Wireshark.
- 7. After your browser has displayed the INTRO-wireshark-file1.html page (it is a simple one line of congratulations), stop Wireshark packet capture by selecting stop in the Wireshark capture window. The main Wireshark window should now look similar to Figure 3. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP message exchanges with the gaia.cs.umass.edu web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well (see, e.g., the many different protocol types shown in the *Protocol* column in Figure 3). Even though the only action you took was to download a web page, there were evidently many other protocols running on your computer that are unseen by the user. We'll learn much more about these protocols as we progress through the text! For now, you should just be aware that there is often much more going on than "meet's the eye"!
- Type in "http" (without the quotes, and in lower case all protocol names are in lower case in Wireshark) into the display filter specification window at the top of the main Wireshark window. Then select *Apply* (to the right of where you entered "http"). This will cause only HTTP message to be displayed in the packet-listing window.
- 9. Find the HTTP GET message that was sent from your computer to the gaia.cs.umass.edu HTTP server. (Look for an HTTP GET message in the "listing of captured packets" portion of the Wireshark window (see Figure 3) that shows "GET" followed by the gaia.cs.umass.edu URL that you entered. When you select the HTTP GET message, the Ethernet frame, IP datagram, TCP segment,

and HTTP message header information will be displayed in the packet-header window<sup>2</sup>. By clicking on '+' and '-' right-pointing and down-pointing arrowheads to the left side of the packet details window, *minimize* the amount of Frame, Ethernet, Internet Protocol, and Transmission Control Protocol information displayed. *Maximize* the amount information displayed about the HTTP protocol. Your Wireshark display should now look roughly as shown in Figure 5. (Note, in particular, the minimized amount of protocol information for all protocols except HTTP, and the maximized amount of protocol information for HTTP in the packet-header window).

10. Exit Wireshark

Intel(R) 82567LM Gigabit Network Connecti	ion [Wireshark 1.6.2 (SVN Rev	38931 from /trunk-1.6)]	CONTRACTOR NO.	Constitution in case	
<u>File Edit View Go Capture Analyze</u>	Statistics Telephony Tools	Internals <u>H</u> elp			
	占   🍳 🗢 🔶 🚡 ;	<u>4</u>   <b>1 1 1 1 1 1 1</b>	. 🖭   🔐 🔛 🍋	) ¾   📜	
Filter: http		Expression Clear Apply	/		
No. Time	Source	Destination	Protocol	Length Info	A
813 43.946687	192.168.1.101	66.103.80.47	НТТР	181 GET /cgi-bin/	alive?0001088 HTTP/1.1
816 43.996668	66.103.80.47	192.168.1.101	нттр	60 HTTP/1.1 200	DK (text/plain)
826 44.457577	192.168.1.101	204.9.163.166	НТТР	333 POST /api/v1.	0/pnr?language=EN&plugin=F
828 44.50/1/1	204.9.163.166	192.168.1.101	HTTP	2/1 HTTP/1.1 200	DK / John / TNITRO winoshank fil
837 45 646802	128 110 245 12	102 168 1 101	нттр	489 GET /WITESHAT	K-TADS/INTRO-WITESHAFK-ITT
838 45 670226	192 168 1 101	128 119 245 12	нттр	429 GET /favicon	ico HTTP/1_1
839 45, 687572	128.119.245.12	192.168.1.101	HTTP	564 HTTP/1.1 404	Not Found (text/html)
840 45.724273	192.168.1.101	128.119.245.12	НТТР	459 GET /favicon.	ico HTTP/1.1
841 45.739188	128.119.245.12	192.168.1.101	нттр	564 HTTP/1.1 404	Not Found (text/html)
847 48.670194	192.168.1.101	128.119.245.12	нттр	459 GET /favicon.	ico HTTP/1.1
848 48.689680	128.119.245.12	192.168.1.101	HTTP	564 HTTP/1.1 404	Not Found (text/html) -
<		III			
■ Frame 835: 489 bytes on wire	(3912 bits), 489 byte	s captured (3912 bits	)		
🗄 Ethernet II, Src: HonHaiPr_Od	:ca:8f (00:22:68:0d:c	a:8f), Dst: Cisco-Li_	45:1f:1b (00:22	2:6b:45:1f:1b)	
Internet Protocol Version 4,	Src: 192.168.1.101 (1	92.168.1.101), Dst: 1	28.119.245.12 (	(128.119.245.12)	
Transmission Control Protocol	, Src Port: 57522 (57	522), Dst Port: http	(80), Seq: 1, A	Ack: 1, Len: 435	
Hypertext Transfer Protocol Winosbark labs (INTRO W	irochark filo1 html W	$TTD / 1 = 1 \ n \ n$			
Host: gaia cs umass edu/r/n	Treshark-Trier.noitr h				
User-Agent: Mozilla/5.0 (Wi	ndows: U: Windows NT	6.1: en-US: rv:1.9.2.	22) Gecko/20110	902 Firefox/3.6.22 (	NET CLR 3.5.30729)\r\n
Accept: text/html,applicati	on/xhtml+xml,applicat	ion/xml;q=0.9,*/*;q=0	.8\r\n		
Accept-Language: en-us,en;q	=0.5\r\n				
Accept-Encoding: gzip,defla	te\r\n				
Accept-Charset: ISO-8859-1,	utf-8;q=0.7,*;q=0.7\r	\n			
Keep-Alive: 115\r\n					
Connection: keep-alive\r\n					
\r\ri [Full request URT: http://a	aia co umaco odu /wino	chark labs/INTRO wino	chark filo1 btg		
<u>runnequest okr. http://g</u>	ara. cs. unass. edu/ wrre	511ar K-1403/111KO-W11 E	Shark-Triet.nei		
0000 00 22 6b 45 1f 1b 00 22 0	68 Od ca 8f 08 00 45 (	00 . "ke" hе.			
0010 01 db 29 13 40 00 80 06 (	00 00 c0 a8 01 65 80 7	77).@e.v			_
0020 f5 0c e0 b2 00 50 ca 16 0000 40 29 39 5f 00 00 47 45	89 b3 d9 41 b1 83 50 1 54 20 2f 77 69 72 65 1	L8PAP. 73 @)9 GE T /wires			
0040 68 61 72 6b 2d 6c 61 62	73 2f 49 4e 54 5 <u>2 4f 2</u>	2d hark-lab s/ <u>INTRO</u>			-
COSO 77 60 72 65 72 68 61 72	Sh 2d 66 60 6c 65 21 1	Marked: 0 Dropped: 0			Profile: Default
Frame (frame), 409 bytes	Packets: 050 Displayed: 132	Marked: 0 Dropped: 0			Profile: Default

Congratulations! You've now completed the first lab.

Figure 5: Wireshark window after step 9

 $<sup>^{2}</sup>$  Recall that the HTTP GET message that is sent to the gaia.cs.umass.edu web server is contained within a TCP segment, which is contained (encapsulated) in an IP datagram, which is encapsulated in an Ethernet frame. If this process of encapsulation isn't quite clear yet, review section 1.5 in the text



# What to hand in

The goal of this first lab was primarily to introduce you to Wireshark. The following questions will demonstrate that you've been able to get Wireshark up and running, and have explored some of its capabilities. Answer the following questions, based on your Wireshark experimentation:

- 1. List 3 different protocols that appear in the protocol column in the unfiltered packet-listing window in step 7 above.
- 2. How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packetlisting window is the amount of time, in seconds, since Wireshark tracing began. To display the Time field in time-of-day format, select the Wireshark *View* pull down menu, then select Time *Display Format*, then select *Time-of-day*.)
- 3. What is the Internet address of the gaia.cs.umass.edu (also known as www-net.cs.umass.edu)? What is the Internet address of your computer?
- 4. Print the two HTTP messages (GET and OK) referred to in question 2 above. To do so, select *Print* from the Wireshark *File* command menu, and select the *"Selected Packet Only"* and *"Print as displayed"* radial buttons, and then click OK.

# Part 2: Ethernet and ARP

In this lab, we'll investigate the Ethernet protocol and the ARP protocol. Before beginning this lab, you'll probably want to review sections 5.4.1 (link-layer addressing and ARP) and 5.4.2 (Ethernet) in the text<sup>3</sup>. RFC 826 (<u>ftp://ftp.rfc-editor.org/in-notes/std/std37.txt</u>) contains the gory details of the ARP protocol, which is used by an IP device to determine the IP address of a remote interface whose Ethernet address is known.

#### 1. Capturing and analyzing Ethernet frames

Let's begin by capturing a set of Ethernet frames to study. Do the following<sup>4</sup>:

- First, make sure your browser's cache is empty. To do this under Mozilla Firefox V3, select *Tools->Clear Recent History* and check the box for Cache. For Internet Explorer, select *Tools->Internet Options->Delete Files*. Start up the Wireshark packet sniffer
- Enter the following URL into your browser http://gaia.cs.umass.edu/wireshark-labs/HTTP-ethereal-lab-file3.html Your browser should display the rather lengthy US Bill of Rights.

<sup>4</sup> If you are unable to run Wireshark live on a computer, you can download the zip file

<sup>&</sup>lt;sup>3</sup> References to figures and sections are for the 6<sup>th</sup> edition of our text, *Computer Networks, A Top-down Approach, 6<sup>th</sup> ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2012.* 

<sup>&</sup>lt;u>http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip</u> and extract the file *ethernet--ethereal-trace-1*. The traces in this zip file were collected by Wireshark running on one of the author's computers, while performing the steps indicated in the Wireshark lab. Once you have downloaded the trace, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the ethernet-ethereal-trace-1 trace file. You can then use this trace file to answer the questions below.

• Stop Wireshark packet capture. First, find the packet numbers (the leftmost column in the upper Wireshark window) of the HTTP GET message that was sent from your computer to gaia.cs.umass.edu, as well as the beginning of the HTTP response message sent to your computer by gaia.cs.umass.edu. You should see a screen that looks something like this (where packet 4 in the screen shot below contains the HTTP GET message)

📶 (Unt	titled) - Wireshark							<u>- 🗆 ×</u>
<u>E</u> ile į	<u>E</u> dit <u>V</u> iew <u>G</u> o <u>C</u> a	pture <u>A</u> nalyze <u>S</u> tatistics (	<u>t</u> elp					
	≝ <b>≥</b>	💓 🗁 🗔 🗙	°) 🛆 🖓	ı ⇔	∞ ۍ ⊈		●, ●, 0,	<b>•••</b>
Eilter:			-	Expression	<u>⊂</u> lear <u>A</u> pply			
No. +	Time	Source	Destination	Protocol	Info			<u>–</u>
	1 0.000000	192.168.2.145	128.119.245.12	TCP	2038 > http	[SYN] Seq=0	Len=0 MSS=14	160 Vin-5
	3 0.050729	192.168.2.145	192.168.2.145	TCP	2038 > http	[SYN, ACK] S [ACK] Seq=1	Ack=1 Win=6	5535
	4 0.055906	192.168.2.145	128.119.245.12	HTTP	GET /wiresh	ark-labs/HTTP	-ethereal-1	ab-fi
<u> </u>	6 0.134167	128.119.245.12	192.168.2.145	TCP	TCP segmen	t of a reasse	mbled PDU]	=6432
	7 0.150302	128.119.245.12	192.168.2.145	TCP	[TCP segmen	t of a reasse	mbled PDU]	
-	8 0.150487 9 0.213639	192.168.2.145	128.119.245.12 192.168.2.145	TCP	ITCP segmen	[ACK] Seq=45 t of a reasse	3 ACK=1762 W mbled PDUl	v1n=6
	10 0.215724	128.119.245.12	192.168.2.145	TCP	TCP Previo	us segment lo	st] [TCP se	gment
	11 0.215947	192.168.2.145	128.119.245.12	TCP HTTP	2038 > http TCB Betran	[ACK] Seq=45	3 Ack=3214 \	√in=6
	13 0.232145	192.168.2.145	128.119.245.12	TCP	2038 > http	[ACK] Seq=45	3 Ack=4810 V	vin=6
	14 0.320470	192.168.2.145	128.119.245.12	HTTP	GET /favico	n.ico HTTP/1.	1	
	16 0.423932	192.168.2.145	168.66.12.224	TCP	2039 > http	4 NOC FOUND ( [SYN] Seq=0	Len=0 MSS=14	160
	17 0.579522	192.168.2.145	128.119.245.12	TCP	2038 > http	[ACK] Seq=79	3 Ack=6235 V	vin=6
	18 3.383584	192.168.2.145	168.66.12.224	TCP TCP	2039 > http 2039 > http	[SYN] Seq=0 [SYN] Seq=0	Len=0 MSS=14 Len=0 MSS=14	160 160
	20 10.389131	128.119.245.12	192.168.2.145	TCP	http > 2038	[FIN, ACK] S	eq=6235 Ack	=793 '
	21 10.389258	192.168.2.145	128.119.245.12	TCP	2038 > http	[ACK] Seq=79	3 Ack=6236 \	vin=6
1								
🕀 Fra	ame 4 (506 byt	es on wire, 506 byt	es captured)					
. Etł	hernet II, Sro	: Netgear_61:8e:6d	(00:09:5b:61:8e:60	i), Dst:	LinksysG_45:	90:a8 (00:0c:	:41:45:90:a8	0
. E Int	ternet Protoco	)], Src: 192.168.2.1	45 (192.168.2.145)	), Dst:	128.119.245.1	.2 (128.119.24	45.12)	
	ansmission Cor nertext Transf	er Protocol, Src	Port: 2038 (2038),	DST PO	ητ: πττρ (80,	I, Seq: I, ACK	<: I, Len: 4	52
	GET /wineshark	-labs/HTTP-etherea	-lab-file3.html Hi	TP/1.1\	r\n			
	Request Meth	nod: GET						
	Request URI:	/wireshark-labs/Hl	TP-ethereal-lab-f	le3.htm	1			
	Request Vers	ion: HTTP/1.1						
H H	Host: gaia.cs.	umass.edu\r\n mille/5_0_(windows)	U. Windows NT 5 1			- Cocko (20070)	515 Finatovi	
	oser-Agent: Mc Accent: text/x	ml annlication/yml	application/vhtml	.; en-us .yml.tey	; rv:⊥.8.⊥.4, τ/html:α=0.9	text/nlain.da	=0.8.image/n	2.0.0.2 na.*/*:
1	Accept-Languad	ae: en−us.en:a=0.5\r	'\n	,	.c/nciin, q=0. ),	cext/pram,q-	-0.0, image/p	···9, / ·
1	Accept-Encodir	ng: gzip,deflate\r\r	ì					
1	Accept-Charset	: ISO-8859-1,utf-8;	q=0.7,*;q=0.7\r\n					
ŀ	Keep-Alive: 30	)0\r\n						
	Connection: ke	ep-alive\r\n						
	\r \n							
4								<u> </u>
0040	68 61 72 6b	2d 6c 61 62 73 2f	48 54 54 50 2d 65	hark-	lah s/HTTP-e			4
0050	74 68 65 72	65 61 6c 2d 6c 61	62 2d 66 69 6c 65	therea	al- lab-file			
0060	33 2e 68 74 1 0a <b>48 6f 7</b> 3	6d 6c 20 48 54 54 74 3a 20 67 61 69	50 2f 31 2e 31 0d 61 2e 63 73 2e 75	3.htm .Host	і н ттр/1.1. п aia.cs.u			
0080	6d 61 73 73	2e 65 64 75 0d 0a	55 73 65 72 2d 41	mass.	edu .Uşer_A			
0090	67 65 6e 74	sa 20 4d 6t /a 69	bc bc 61 27 35 2e	gent	Mo zilla/5.			
				P: 21 D: 21	1 M: 0 Drops: 0			4

• Since this lab is about Ethernet and ARP, we're not interested in IP or higherlayer protocols. So let's change Wireshark's "listing of captured packets" window so that it shows information only about protocols below IP. To have Wireshark do this, select *Analyze->Enabled Protocols*. Then uncheck the IP box and select *OK*. You should now see an Wireshark window that looks like:

🔽 (Ur	ntitle	:d) - W	irest	nark																						_	
Eile	<u>E</u> dit	<u>V</u> iew	<u>G</u> o	<u> </u>	pture	• <u>A</u>	nalyze	e S	tatisti	cs <u>ł</u>	<u>t</u> elp																
	ë	( @	l I	<b>@</b> í		ļ	D	(	-	x	¢,	Ē	3	٩	\$	\$	¢	D	Ŧ	⊉			Ţ	•	Θ,	0	
<u>F</u> ilter:															• [	xpressi	on	⊆le	ar <u>A</u>	pply							
No. +		Time			So	urce					Des	tinatio	n			Protoc	:ol	Info									-
	1	0.00	0000	0	Ne	etge	ear_	61:	8e:0	5d • a 8	Lir	iksy Ingg	SG_4 r 61	5:90	):a8 6d	0×08	00	IP TP									
	3	0.05	0729	9	Ne	etge	ear_	61:	8e:0	5d	Lir	iksy	sG_4	5:90	:a8	0×08	ŏŏ	IP									
	4	0.05	5900 870(	6 0	Ne Li	et ge inks	ear_ svsG	61: 45	8e:0 :90	od :a8	L1r Net	iksy :dea	sG_4 r 61	5:90 :8e	1:48 6d	0x08 0x08	100 100	IP IP									
	6	0.13	416	2	Li	inks	ýsg	_45	:90	a8	Net	gea	r_61	:8e:	6d	0×08	00	IP									
	8	0.15	0302 0483	2 7	Ne	etge	syse ear_	_45 61:	:90 8e:0	: a8 5d	Lir	igea 1ksy	г_ю⊥ sG_4	5:90	00 1:a8	0x08	100	IP									
	9	0.21	3639	9 1	Li	inks	ysG	_45	:90 •90	: a8	Net	:gea	r_61	:8e:	6d 6d	0x08	00	IP									
	$11^{10}$	0.21	5947	7	Ne	etge	ear_	61:	8e:0	.ao 5d	Lir	iksy	sG_4	5:90	:a8	0x08	00	IP									
	12	0.23	1749 2149	9	Li	inks	iysG Dan	_45	:90) 80:0	: a8	Net	:geā	r_61	:8e:	6d	0x08	00	IP									
	14	0.32	0470	0	Ne	etge	ear_	61:	8e:0	5d	Lir	iksy	sG_4	5:90	:a8	0x08	00	IP									
	15	0.40	3428	8	Li	inks	ysG	_45	:90:	: a8	Net	gea	r_61	:8e:	6d	0x08	00	IP									
	17	0.42	9522	2	NE	etge	ear_	61:	8e:0	5d	Lir	iksy iksy	5G_4 SG_4	5:90	1:a8	0x08	00	IP									
	18	3.38	3584	4	Ne	etĝe	ear_	61:	8e:0	5d	Lir	iksý	sG_4	5:90	:a8	0x08	00	IP									
	20	9.39	219. 8913	/ 31	Li	inks	sar_ svsG	_45	:90	a8	Net	iksy :gea	sG_4 r_61	:8e:	6d	0x08	00	IP IP									
	21	10.3	892	58	Ne	etge	ear_	61:	8e:0	5d	Lir	īksy	sG_4	5:90	:a8	0×08	00	IΡ									
4																											Þ
∃ Fr	ame	24 (	506	byt	es	on	wir	e,	506	byt	es (	capt	ure	3) (k													^
🗆 Et	:her	net	II,	Sno	: N	Veto	gear	_61	:8e	:6d	(00	:09:	5b:0	51:80	e:6d)	, Ds1	:: L	_ink	sys	G_45	:90:	a8	(00:0	)c:41	:45:9	0:a8)	
Ξ	Des	tina ddm	tion cc.	n:L	ink.	(sys	5G_4	5:9	0:a)	в (С Соо.	0:0	::41	:45	:90:8	18)												
		uure	55: (	0	iк 5 ј	/56_	45:	90:	40	(00:	UC:4	hit	: TI	ndivi	, idua]	addr	• es <	5.0	unic	astì							
											LG	bit	: G	loba'	l]γ ι	inique	e ac	ddre	255	(fac	tory	/ det	Fault	:)			
=	Sou	ırce:	Ne <sup>1</sup>	tgea	ır_€	51:8	3e:6	d (	00:	09:5	b:63	L:8e	:6d)	)	-						-						
	A	ddre	ss:	Net	gea	ar_0	51:8	e:6	id (1	00:0	9:5	o:61	:8e	:6d)		1-1-				`							
	•	•••	••••	0	••	• • •	• •	• • •	•••	••• =	: IG · IG	bit	: II • c'	ndiv: Lobal	idua i Il viri	addi unique	ress	5 (l Hara	unic Sss	ast) (fac	tory	/ dat	Fault				
	TVR	 e: I	 Р ((	 0x08	3003	)	• •	•••	•••	••••	LG.	DIC		гора	i iy t	inque	= a(	June	= > >	(i ac		ue	aun	-)			
Da	ita	(492	byt	tes)																							-
4																											
0000	0	0 0c 1 ec	41 87	45 ·	90 40	a8 00	00 ( 80 (	29 16	5b 38	61 65	Be 6	d 08	3 00 2 01	45	00 77	AE	na.	۰Ľ	a.m.	.Е. "							~
0020	f	5 QC	07	f6	0ŏ	50	7a 7	74	<u>c4</u>	58	7d a	6 2	7 90	50	18		Pz	÷.	x}.	.P.							
0030	† 6	f ff 8 61	3a 72	9C 1 6b 3	00 2 d	00 · 6c	47 4 61 6	45 52	54 73	20 ) 2f 4	2† 7 48 5	769 454	972 450	65 2d	73 65	hark	G -la	ЕT bs	/w1 /нт1	ires P-e							
0050	7	4 68	65	72	65 6d	61	6c 2	2d	6C	61 54	62 2	d 60	5 69	6C	65 0d	ther	eal	- 1	ab-f	iļē							
0070	0	a 48	6f	73	74	3a	20 6	57	61	69	50 Z	e 63	3 73	2e	75	.Hos	t:	ga	ia.c	 							_
File: "(	:\DC	CUME	-1\PA	ULAW	/~1\I	.OCA	LS~1	Tem	p\ethe	erXXX	(a026	20" 85	84 By	tes OO	P	21 D: 2	1 M:	0 Dro	ops: O								11.

In order to answer the following questions, you'll need to look into the packet details and packet contents windows (the middle and lower display windows in Wireshark).

Select the Ethernet frame containing the HTTP GET message. (Recall that the HTTP GET message is carried inside of a TCP segment, which is carried inside of an IP datagram, which is carried inside of an Ethernet frame; reread section 1.5.2 in the text if you find this encapsulation a bit confusing). Expand the Ethernet II information in the packet details window. Note that the contents of the Ethernet frame (header as well as payload) are displayed in the packet contents window.

Answer the following questions, based on the contents of the Ethernet frame containing the HTTP GET message. Whenever possible, when answering a question you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. Annotate the printout<sup>5</sup> to explain your answer. To print a packet, use *File->Print*, choose *Selected packet only*, choose *Packet summary line*, and select the minimum amount of packet detail that you need to answer the question.

- 1. What is the 48-bit Ethernet address of your computer?
- 2. What is the 48-bit destination address in the Ethernet frame? Is this the Ethernet address of gaia.cs.umass.edu? (Hint: the answer is *no*). What device has this as its Ethernet address? [Note: this is an important question, and one that students sometimes get wrong. Re-read pages 468-469 in the text and make sure you understand the answer here.]
- 3. Give the hexadecimal value for the two-byte Frame type field. What upper layer protocol does this correspond to?
- 4. How many bytes from the very start of the Ethernet frame does the ASCII "G" in "GET" appear in the Ethernet frame?

Next, answer the following questions, based on the contents of the Ethernet frame containing the first byte of the HTTP response message.

- 5. What is the value of the Ethernet source address? Is this the address of your computer, or of gaia.cs.umass.edu (Hint: the answer is *no*). What device has this as its Ethernet address?
- 6. What is the destination address in the Ethernet frame? Is this the Ethernet address of your computer?
- 7. Give the hexadecimal value for the two-byte Frame type field. What upper layer protocol does this correspond to?
- 8. How many bytes from the very start of the Ethernet frame does the ASCII "O" in "OK" (i.e., the HTTP response code) appear in the Ethernet frame?

<sup>&</sup>lt;sup>5</sup> What do we mean by "annotate"? If you hand in a paper copy, please highlight where in the printout you've found the answer and add some text (preferably with a colored pen) noting what you found in what you 've highlight. If you hand in an electronic copy, it would be great if you could also highlight and annotate.

#### 2. The Address Resolution Protocol

In this section, we'll observe the ARP protocol in action. We strongly recommend that you re-read section 5.4.1 in the text before proceeding.

#### **ARP Caching**

Recall that the ARP protocol typically maintains a cache of IP-to-Ethernet address translation pairs on your computer The *arp* command (in both MSDOS and Linux/Unix) is used to view and manipulate the contents of this cache. Since the *arp* command and the ARP protocol have the same name, it's understandably easy to confuse them. But keep in mind that they are different - the *arp* command is used to view and manipulate the ARP protocol defines the format and meaning of the messages sent and received, and defines the actions taken on message transmission and receipt.

Let's take a look at the contents of the ARP cache on your computer:

- **MS-DOS.** The *arp* command is in c:\windows\system32, so type either "*arp*" or "*c:\windows\system32\arp*" in the MS-DOS command line (without quotation marks).
- Linux/Unix/MacOS. The executable for the *arp* command can be in various places. Popular locations are /sbin/arp (for linux) and /usr/etc/arp (for some Unix variants).

The Windows *arp* command with no arguments will display the contents of the ARP cache on your computer. Run the *arp* command.

9. Write down the contents of your computer's ARP cache. What is the meaning of each column value?

In order to observe your computer sending and receiving ARP messages, we'll need to clear the ARP cache, since otherwise your computer is likely to find a needed IP-Ethernet address translation pair in its cache and consequently not need to send out an ARP message.

- **MS-DOS.** The MS-DOS *arp* –*d* \* command will clear your ARP cache. The –*d* flag indicates a deletion operation, and the \* is the wildcard that says to delete all table entries.
- Linux/Unix/MacOS. The *arp* –*d* \* will clear your ARP cache. In order to run this command you'll need root privileges. If you don't have root privileges and can't run Wireshark on a Windows machine, you can skip the trace collection part of this lab and just use the trace discussed in the earlier footnote.

#### **Observing ARP in action**

Do the following<sup>6</sup>:

- Clear your ARP cache, as described above.
- Next, make sure your browser's cache is empty. To do this under Mozilla Firefox V3, select *Tools->Clear Recent History* and check the box for Cache. For Internet Explorer, select *Tools->Internet Options->Delete Files*.
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-lab-file3.html Your browser should again display the rather lengthy US Bill of Rights.
- Stop Wireshark packet capture. Again, we're not interested in IP or higher-layer protocols, so change Wireshark's "listing of captured packets" window so that it shows information only about protocols below IP. To have Wireshark do this, select *Analyze->Enabled Protocols*. Then uncheck the IP box and select *OK*. You should now see an Wireshark window that looks like:

<sup>&</sup>lt;sup>6</sup> The *ethernet-ethereal-trace-1* trace file in <u>http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip</u> was created using the steps below (in particular after the ARP cache had been flushed).

<b>M</b> et	nerne	t-ethe	real-tra	ce-1 - '	Wireshar	k															_ 🗆 🗵
Eile	<u>E</u> dit	⊻iew	<u>G</u> o <u>C</u> a	apture	<u>A</u> nalyze	<u>S</u> tatis	tics <u>F</u>	<u>t</u> elp													
	ê.	0	0	<b>)</b>	D	<b></b>	×	e,	<u> </u>	٩	4	⇔	Ø	Ŧ	⊉		<b>P</b>	€,	Θ,	0	**
Eilter											- E	xpression	n 🤉	<u>lear A</u>	pply						
No		Time		Sour	ce			Destin	nation			Protoco	l In	fo							
	1	$0.000 \\ 0.001$	000	Amb Lin	itMic_ ksvsG i	a9:3d da:af	:68 :73	Ambi	idcast tMic	a9:3	d:68	ARP	Wh 19	no has 02.168	$\frac{192}{3.1.1}$	.168.1 is at	.1?	Tell 1 6:25:0	92.1 la:af	68.1. :73	105
	3	0.001	028	Amb	itMic_	a9:3d	:68	Link	sysG_	da:a	f:73	0x080	)0 IF								
	5	2.962 8.971	488	amb Amb	itMic_	a9:30 a9:3d	:68	Link	.sysu_ :sysu_	_ua:a _da:a	if:73	0x080	)0 IF	, ,							
	6	13.54 17.44	2974 4423	Tel Amb	ebit_7. itMic .	3:8d: a9:3d	ce :68	Broa	idcast isvsG	: data	f:73	ARP 0x080	Wł ЭО ТЕ	no has >	5 192	.168.1	1.117?	Tell	192	.168.	1.104
	8	17.46	5902	Lin	ksysG_	da:af	:73	Ambi	tMic_	a9:3	d:68	0×080	)0 IF								
	10	17.46 17.46	6468	amo Amb	itMic_	a9:3d a9:3d	:68	Link	:sysg_ :sysg_	_da:a _da:a	if:73	0X080	)O IF )O IF	5							
	11 :	17.49	4766	Lin	ksysG_i	da:af da:af	:73	Ambi	tMic_	a9:3	d:68	0x080	)0 IF	) )							
	13	17.50	0025	Lin	ksysG_i	da:af	:73	Ambi	tMic_	a9:3	d:68	0x080	)0 IF	5							
	14 :	17.50 17.52	0069	Amb Lin	itMic_/ ksysG_⊨	a9:3d da:af	:68	Ambi	sysG_ tMic_	_da:a _a9:3	d:68	0×080 0×080	)O IF )O IF	) )							
	16	17.52	7422	Lin	ksysG_	da:af	:73	Ambi	tMic_	a9:3	d:68	0x080	)0 IF								
	17.	17.02	/43/	AIID	TUMIC_	a9:50	:00		.syse_	ua:a	1:75	0,000	JU 16								
1																					
± FI	ame	1 (4	2 byt	es on	wire,	42 k	ytes	Capt	uned]	)											
E1	her:	net I ss Da	I, Sr solut:	c: Am ion D	bitMic rotoco	_a9:3 1 (no	d:68	(00) + )	:d0:59	9:a9:	:3d:68	3), Ds <sup>.</sup>	t:В	roadc	ast (	(ff:ff	:ff:ff	:ff:f	f)		
	Har	dware	type:	: Eth	ernet	(0x00	01)	0													
	Pro	tocol	type	: IP	(0×080	0)															
	Har	dware tocol	size size	:6 •л																	
	Opc	ode:	reque	. 4 st (0	×0001)																
	sen	der M	IAC ad	dress	: Ambi	tMic_	a9:3	d:68	(00:0	d0:59	9:a9:3	d:68)									
	Sen	der I det N	P addi MC add	ress: drass	192.1	68.1. 0.00	105	(192. 0.00	.168.1	L.105	5) )•00•0	0.000									
	Tar	get I get I	P add	ress:	192.1	68.1.	1(1	92.10	58.1.1	L)											
0000	ff	ff ·	FF FF	FF FI	- 00 dr	) 59	a9 -	3d 68	08.0	6 00	01			Y. =h							
0010	08	3 00	26 04		L 00 d0	59	a9 3	3d 68	c0 a	8 01	69			Υ.=h.	i						
0020	00	00 0	00 00	00 00		, 01	UL						• • •	••							
File: "	C:\Doc	uments	and Sett	ings\Pau	ula Wing\M	ly Docur	nents\\	Viresha	rk\traces	s - ethe	ereal(	P: 17 D:	: 17 M:	0							1.

In the example above, the first two frames in the trace contain ARP messages (as does the  $6^{th}$  message). The screen shot above corresponds to the trace referenced in footnote 1.



Answer the following questions:

- 10. What are the hexadecimal values for the source and destination addresses in the Ethernet frame containing the ARP request message?
- 11. Give the hexadecimal value for the two-byte Ethernet Frame type field. What upper layer protocol does this correspond to?
- 12. Download the ARP specification from <u>ftp://ftp.rfc-editor.org/in-notes/std/std37.txt</u>. A readable, detailed discussion of ARP is also at <u>http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/arp.html</u>.
  - a) How many bytes from the very beginning of the Ethernet frame does the ARP *opcode* field begin?
  - b) What is the value of the *opcode* field within the ARP-payload part of the Ethernet frame in which an ARP request is made?
  - c) Does the ARP message contain the IP address of the sender?

- d) Where in the ARP request does the "question" appear the Ethernet address of the machine whose corresponding IP address is being queried?
- 13. Now find the ARP reply that was sent in response to the ARP request.
  - a) How many bytes from the very beginning of the Ethernet frame does the ARP *opcode* field begin?
  - b) What is the value of the *opcode* field within the ARP-payload part of the Ethernet frame in which an ARP response is made?
  - c) Where in the ARP message does the "answer" to the earlier ARP request appear the IP address of the machine having the Ethernet address whose corresponding IP address is being queried?
- 14. What are the hexadecimal values for the source and destination addresses in the Ethernet frame containing the ARP reply message?
- 15. Open the *ethernet-ethereal-trace-1* trace file in http://gaja.cs.umass.edu/wireshark-labs/wireshark-traces.ziv

<u>http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip</u>. The first and second ARP packets in this trace correspond to an ARP request sent by the computer running Wireshark, and the ARP reply sent to the computer running Wireshark by the computer with the ARP-requested Ethernet address. But there is yet another computer on this network, as indicated by packet 6 – another ARP request. Why is there no ARP reply (sent in response to the ARP request in packet 6) in the packet trace?

# Part 3. ICMP

In this lab, we'll explore several aspects of the ICMP protocol:

- ICMP messages generating by the Ping program;
- ICMP messages generated by the Traceroute program;
- the format and contents of an ICMP message.

Before attacking this lab, you're encouraged to review the ICMP material in section 4.4.3 of the text<sup>7</sup>. We present this lab in the context of the Microsoft Windows operating system. However, it is straightforward to translate the lab to a Unix or Linux environment.

### ICMP and Ping

Let's begin our ICMP adventure by capturing the packets generated by the Ping program. You may recall that the Ping program is simple tool that allows anyone (for example, a network administrator) to verify if a host is live or not. The Ping program in the source host sends a packet to the target IP address; if the target is live, the Ping program in the target host responds by sending a packet back to the source host. As you might have guessed (given that this lab is about ICMP), both of these Ping packets are ICMP packets.

Do the following<sup>8</sup>:

- Let's begin this adventure by opening the Windows Command Prompt application (which can be found in your Accessories folder).
- Start up the Wireshark packet sniffer, and begin Wireshark packet capture.
- The *ping* command is in c:\windows\system32, so type either "*ping -n 10 hostname*" or "*c:\windows\system32\ping -n 10 hostname*" in the MS-DOS command line (without quotation marks), where hostname is a host on another continent. If you're outside of Asia, you may want to enter www.ust.hk for the Web server at Hong Kong University of Science and Technology. The argument "-n 10" indicates that 10 ping messages should be sent. Then run the Ping program by typing return.
- When the Ping program terminates, stop the packet capture in Wireshark.

<sup>8</sup> If you are unable to run Wireshark live on a computer, you can download the zip file

<sup>&</sup>lt;sup>7</sup> References to figures and sections are for the 6<sup>th</sup> edition of our text, *Computer Networks, A Top-down Approach, 6<sup>th</sup> ed.*, J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2012.

<sup>&</sup>lt;u>http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip</u> and extract the file *ICMP-ethereal-trace-1*. The traces in this zip file were collected by Wireshark running on one of the author's computers, while performing the steps indicated in the Wireshark lab. Once you have downloaded the trace, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the *ICMP-ethereal-trace-1* trace file. You can then use this trace file to answer the questions below.

At the end of the experiment, your Command Prompt Window should look something like Figure 1. In this example, the source ping program is in Massachusetts and the destination Ping program is in Hong Kong. From this window we see that the source ping program sent 10 query packets and received 10 responses. Note also that for each response, the source calculates the round-trip time (RTT), which for the 10 packets is on average 375 msec.

📾 Command Prompt	
C:\WINDOWS\SYSTEM32>ping -n 10 www.ust.hk	
Pinging www.ust.hk [143.89.14.34] with 32 bytes of data:	
Reply from 143.89.14.34: bytes=32 time=415ms TTL=231 Reply from 143.89.14.34: bytes=32 time=425ms TTL=231 Reply from 143.89.14.34: bytes=32 time=318ms TTL=231 Reply from 143.89.14.34: bytes=32 time=314ms TTL=231 Reply from 143.89.14.34: bytes=32 time=336ms TTL=231 Reply from 143.89.14.34: bytes=32 time=359ms TTL=231 Reply from 143.89.14.34: bytes=32 time=381ms TTL=231 Reply from 143.89.14.34: bytes=32 time=401ms TTL=231 Reply from 143.89.14.34: bytes=32 time=400ms TTL=231	
Ping statistics for 143.89.14.34:	
Approximate round trip times in milli-seconds: Minimum = 314ms, Maximum = 425ms, Average = 375ms	
C:\WINDOWS\SYSTEM32> C:\WINDOWS\SYSTEM32> C:\WINDOWS\SYSTEM32>	

Figure 1 Command Prompt window after entering Ping command.

Figure 2 provides a screenshot of the Wireshark output, after "icmp" has been entered into the filter display window. Note that the packet listing shows 20 packets: the 10 Ping queries sent by the source and the 10 Ping responses received by the source. Also note that the source's IP address is a private address (behind a NAT) of the form 192.168/12; the destination's IP address is that of the Web server at HKUST. Now let's zoom in on the first packet (sent by the client); in the figure below, the packet contents area provides information about this packet. We see that the IP datagram within this packet has protocol number 01, which is the protocol number for ICMP. This means that the payload of the IP datagram is an ICMP packet.

<b>M</b> icr	np-et	hereal-	trace-1	- Wire	shark															- 🗆 ×
Eile	<u>E</u> dit	⊻iew	<u>G</u> o <u>C</u>	apture	<u>A</u> nalyze	<u>S</u> tatis	itics <u>F</u>	<u>t</u> elp												
	ë.	0	<b>@</b> (	<b>)</b>	D	8	x	e,	<b>—</b>	٩	4	⇔	ŝ	Ŧ	⊉		J	€,	Θ	0
Eilter	icm	пр									▼ E	xpression	<u>⊂</u> le	ar	Apply					
No. •		Time		Sour	te			Dest	ination			Protocol	Info							
	3	0.001	656 098	192	.168.1	.101		143	.89.14	1.34		ICMP	Ech		ping)	reques	t			
	5 :	1.006	279	192	.168.1	.101		143	.89.14	1.34		ICMP	Ech	io (	ping)	reques	t			
	6 :	1.431	684	143	.89.14	.34		192	.168.1	L.101		ICMP	Ech	οÇ	ping)	reply				
	- 7	2.006	328	192	.168.1	.101		143	.89.14	1.34		ICMP	Ech	0 (	ping)	reques	st			
	9	2.524 3.006	479 356	192	.09.14 .168.1	. 101		143	.89.14	1.34		ICMP	Ech		ping)	reques	t			
	10	3.321	121	143	.89.14	.34		192	.168.1	1.101		ICMP	Ech	ō (	ping)	reply	-			
	11 .	4.006	398	192	.168.1	.101		143	.89.14	1.34		ICMP	Ech	o (	ping)	reques	st			
	12 4	4.343 5.006	301 454	143	.89.14	.34		192	.168.1	L.101 1 34		ICMP	ECN		ping)	reply	+			
	14	5.365	480	143	.89.14	.34		192	.168.1	L.101		ICMP	Ech	ю (	ping)	reply	, <b>c</b>			
	15 (	6.022	116	192	.168.1	.101		143	.89.14	1.34		ICMP	Ech	юČ	ping)	reques	st			
	16	6.403	470	143	.89.14	.34		192	.168.1	L.101		ICMP	Ech	0 9	ping)	reply				
-	18	7.022 7.473	213 214	143	89 14	34		197	168 1	+.34   101		TCMP	ECH		ping) ning)	reques	, L			
	19 :	B.022	249	192	.168.1	.101		143	.89.14	1.34		ICMP	Ech	ŏÒ	ping)	reques	t			
	20 3	8.423	018	143	.89.14	.34		192	.168.1	L.101		ICMP	Ech	οÒ	ping)	reply				
	21	9.022	254	192	.168.1	.101		143	.89.14	1.34		ICMP	Ech	0 (	ping)	reques	st			
	22 '	9.432	063	143	.89.14	.34		192	.168.1	1.101		ICMP	ECN	0 (	ping)	reply				
4																				Þ
🕀 FI	ame	3 (7	4 byt	es on	wire,	74 k	oytes	cap	tured)	)										
. ∃ E	ther	net I	I, Sr	c: De	ll⊂omp	_4f:3	36:23	(00	:08:74	4:4f:3	36:23	), Dst	:Lir	nksy	ysG_da	:af:73	(00:	06:25	:da:	af:73)
ΞI	nteri	net P	rotoc	ol, s	rc: 19	2.168	3.1.1	01 (	192.1	68.1.1	LO1),	Dst:	143.8	39.1	L4.34	(143.8	9.14.	34)		
	Ven	sion:	4																	
	Неа	der l	ength	: 20	bytes															
÷	Dif	feren	tiate	d ser	vices	Field	x0 :b	00 (	DSCP	0×00:	Defa	ult; E	CN: (	0×0	2)					
	TOT	al Le	ngth:	60																
	Ide	ntifi	catio	n: Ox	d1fd (	53757	7)													
+	Fla	gs: O	x00																	
	Fra	gment	otts	et: 0																
	Tim	e to	live:	128																
L	Pro	tocol	: ICM	P (UX	01)	F	7													
•	неа	der c	necks	um: U 69 1 1	XU930 101 (1	LCOPT	rectj	1.01.5												
	Doci	rce: tipot	192.I	08.I. 147 0	101 (1 0 14 2	92.10 4 (14	12 90	101) 14	243											
H I	nteri	net C	ontro	143.8 1 Mes	sage P	roto	+3.89 col	.14.	54)											
4																				
0010	00	30 0	41 fd	00.00	80.0	í na	3h	-0 2	8 01 6	5 8f	59		c	•						<u> </u>
0020	Ŭē	22	00 80	e4 5a	1 02 00	0 <u>6</u> 7	01	51 6	2 6 <u>3 6</u>	64 <u>65</u>	66		z. <u>q</u>	. ab	cdef					
0030	67	68 6	59 6a	6b 60	- 6d 60	≘ 6f	70	71 7.	2 73 7	4 75	76	ghijk Wabed	lmn ö	pqr	stuv					
0040	77	-01- (	12 05	04 0.	-00-0.	- 08	-09					wabcu	er y Ti							-
Intern	et Con	trol Mes	sage Pro	tocol (ici	mp), 40 by	/tes					P: 22	2 D: 20 M	:0							1.

Figure 2 Wireshark output for Ping program with Internet Protocol expanded.

Figure 3 focuses on the same ICMP but has expanded the ICMP protocol information in the packet contents window. Observe that this ICMP packet is of Type 8 and Code 0 - a so-called ICMP "echo request" packet. (See Figure 4.23 of text.) Also note that this ICMP packet contains a checksum, an identifier, and a sequence number.

📶 icmp-ethereal-trace-1	- Wireshark				
<u>File E</u> dit <u>V</u> iew <u>G</u> o <u>C</u> ap	pture <u>A</u> nalyze <u>S</u> tatistics <u>H</u>	<u>l</u> elp			
	💓 🗁 🔣 🗙	% ≛ 🖣 🌣	🗢 轮 🔂 🕹		Θ, 0,
Eilter: icmp		<b>▼</b> [	xpression ⊆lear Apply		
No Time	Source	Destination	Protocol Info		<u> </u>
3 0.001656	192.168.1.101	143.89.14.34	ICMP Echo (ping)	request	
5 1,006279	192.168.1.101	143.89.14.34	ICMP ECHO (ping)	reply	
6 1.431684	143.89.14.34	192.168.1.101	ICMP Echo (ping)	reply	
7 2.006328	192.168.1.101	143.89.14.34	ICMP Echo (ping)	request	
8 2.324479	143.89.14.34	192.168.1.101	ICMP Echo (ping)	reply	
9 3.006356	192.168.1.101	143.89.14.34	ICMP Echo (ping)	request	
11 4 006398	192 168 1 101	143 89 14 34	ICMP Echo (ping)	repty	
12 4.343301	143.89.14.34	192.168.1.101	ICMP Echo (ping)	reply	
13 5.006454	192.168.1.101	143.89.14.34	ICMP Echo (ping)	request	
14 5.365480	143.89.14.34	192.168.1.101	ICMP Echo (ping)	reply	
15 6.022116	192.168.1.101	143.89.14.34	ICMP Echo (ping)	request	
16 6.403470	143.89.14.34	192.168.1.101	ICMP Echo (ping)	reply	
19 7 422213	142 80 14 24	107 168 1 101	ICMP Echo (ping)	request	
19 8.022249	192.168.1.101	143.89.14.34	ICMP Echo (ping)	request	
20 8.423018	143.89.14.34	192.168.1.101	ICMP Echo (ping)	reply	
21 9.022254	192.168.1.101	143.89.14.34	ICMP Echo (ping)	request	
22 9.432063	143.89.14.34	192.168.1.101	ICMP Echo (ping)	reply	
4					
E Ename 3 (74 byte	s on wire 74 butes	cantured)			
Ethernet II Src	• DellComp 4f•36•73	(00.08.74.4f.36.23	) Dat: Linksvag da	f.73 (00.06.25)	da • af • 72)
I Internet Protoco	. Dericomp <u>4</u> 1.30.23 1 src· 192 168 1 1	(00.08.74.41.30.2. 01 (197 168 1 101)	Dst · 143 89 14 34	(143 89 14 34)	.ua.ar./5)
Internet Control	Message Protocol	UI (192.100.1.101),	030. 143.09.14.34	(143.09.14.34)	
Type: 8 (Echo	(ning) request)				
Codo: 0	(ping) requesty				
Code. 0	E. [connoct]				
Tdentifien, Ov	Ja [COFFECU]				
Tuent Ther: 0x	.0200				
Sequence numbe	r: 20309 (UX6/UI)				
Data (32 bytes	)				
0000 00 06 25 da a 0010 00 3c d1 fd ( 0020 0e 22 08 00 a 0030 67 68 69 63 6	af 73 00 08 74 4f 3 00 00 80 01 09 3b 0 24 5a 02 00 67 01 0 5b 6c 6d 6a 6f 70	36 23 08 00 45 00 20 a8 01 65 8f 59 51 62 63 64 65 66 71 72 73 74 75 76	%s t06#E. .<;e.Y ."Z g.abcdef		
0040 77 61 62 63 6	54 65 66 67 68 69	- <u></u> 70 74 75 70	wabcdefg hi		
Internet Control Message Proto	ocol (icmp), 40 bytes	P	22 D: 20 M: 0		li.

Figure 3 Wireshark capture of ping packet with ICMP packet expanded.



#### What to Hand In:

You should hand in a screen shot of the Command Prompt window similar to Figure 1 above. Whenever possible, when answering a question below, you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. Annotate the printout<sup>9</sup> to explain your answer. To print a packet, use *File->Print*, choose *Selected packet only*, choose *Packet summary line*, and select the minimum amount of packet detail that you need to answer the question.

You should answer the following questions:

<sup>&</sup>lt;sup>9</sup> What do we mean by "annotate"? If you hand in a paper copy, please highlight where in the printout you've found the answer and add some text (preferably with a colored pen) noting what you found in what you 've highlight. If you hand in an electronic copy, it would be great if you could also highlight and annotate.

- 1. What is the IP address of your host? What is the IP address of the destination host?
- 2. Why is it that an ICMP packet does not have source and destination port numbers?
- 3. Examine one of the ping request packets sent by your host. What are the ICMP type and code numbers? What other fields does this ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?
- 4. Examine the corresponding ping reply packet. What are the ICMP type and code numbers? What other fields does this ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?