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*, 6th 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 1.24¹) 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.

¹ 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	Courses (1988) 201	-	-	
Description	IP	Packets	Packets/s	Stop
Intel(R) 82567LM Gigabit Network Connection	fe80::844a:1239:ab19:74ea	28	0	Start Options Details
🔊 Microsoft	192.168.1.100	17	0	Start Options Details
<u>H</u> elp				<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"!
- 8. 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 window². 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 (Connection [Wireshark 1.6.2 (SVN R	ev 38931 from /trunk-1.6)]	C. St. Int. St. & completion, Name . Marcall, South	
	nalyze <u>S</u> tatistics Telephony <u>T</u> ool	s <u>I</u> nternals <u>H</u> elp		
	s 🔁 占 🔍 🗢 🔿 7	<u>।</u>	🛛 🔐 📧 🎨 🔆 🛛 💢	
Filter: http		Expression Clear Apply		
No. Time	Source	Destination Pro	tocol Length Info	A
813 43.946687	192.168.1.101	66.103.80.47 HT		
816 43.996668 826 44.457577	66.103.80.47 192.168.1.101	192.168.1.101 HT 204.9.163.166 HT	TP 60 HTTP/1.1 200 OK (text, TP 333 POST /api/v1.0/pnr?land	
828 44.507171	204.9.163.166		тр 271 нттр/1.1 200 ок	gaage-Enap ragmen
835 45.629833	192.168.1.101		TP 489 GET /wireshark-labs/IN	
837 45.646802 838 45.670226	128.119.245.12 192.168.1.101	192.168.1.101 HT 128.119.245.12 HT	TP 434 HTTP/1.1 200 OK (text, TP 429 GET /favicon.ico HTTP/1	
838 45.670226	128.119.245.12		TP 429 GET /1401CON.TCO HTTP/J TP 564 HTTP/1.1 404 Not Found	
840 45.724273	192.168.1.101	128.119.245.12 HT	TP 459 GET /favicon.ico HTTP/1	1.1
841 45.739188	128.119.245.12		TP 564 HTTP/1.1 404 Not Found	
847 48.670194	192.168.1.101 128.119.245.12		TP 459 GET /favicon.ico HTTP/1 TP 564 HTTP/1.1 404 Not Found	
848 48 689680				(cexe/nemry
🗄 Ethernet II, Src: HonHai 🕀 Internet Protocol Versio	wire (3912 bits), 489 by ipr_0d:ca:8f (00:22:68:0d on 4, Src: 192.168.1.101 otocol, Src Port: 57522 ()	<pre>m tes captured (3912 bits) :ca:8f), Dst: Cisco-Li_45:1 (192.168.1.101), Dst: 128.1 57522), Dst Port: http (80)</pre>	f:1b (00:22:6b:45:1f:1b) 19.245.12 (128.119.245.12)	•
 Frame 835: 489 bytes on Ethernet II, Src: HonHai Internet Protocol Versic Transmission Control Protoc GET /wireshark-labs/IN Host: gaia.cs.umass.ec User-Agent: Mozilla/5. Accept: text/html,app 	wire (3912 bits), 489 by Pr_Odica:8f (00:22:68:0d n4, Src: 192.168.1.101 btocol, Src Port: 57522 (1 scol wTRO-wireshark-file1.html Ju\r\n 0 (windows; U; windows N lication/xhtml+xml,applic.	tes captured (3912 bits) :ca:8f), Dst: Cisco-Li_45:1 (192.168.1.101), Dst: 128.1 57522), Dst Port: http (80) HTTP/1.1\r\n	f:1b (00:22:6b:45:1f:1b) 19.245.12 (128.119.245.12) , Seq: 1, Ack: 1, Len: 435 Gecko/20110902 Firefox/3.6.22 (.NET CLR =	, 3. 5. 30729)\r\n
 Frame &35: 489 bytes on Frame &35: 489 bytes on Ethernet II, Src: HonHai Internet Protocol Versic Transmission Control Prot Hypertext Transfer Proto GET /wireshark-labs/IN Host: gaia.cs.umass.ec User-Agent: Mozilla/5. Accept: text/html,appl Accept-Language: en-us Accept-charset: ISo-88 Keep-Alive: 115\r\n Connection: keep-alive 	<pre>wire (3912 bits), 489 by iPr_Odica:8f (00:22:68:0d on 4, src: 192.168.1.101 stocol, src Port: 57522 (: scol urRo-wireshark-file1.html ulr\n 0 (windows; u; windows N lication/xhtml+xml,applic: s,en;q=0.5\r\n deflate\r\n 359-1,utf-8;q=0.7,*;q=0.7</pre>	<pre>tes captured (3912 bits) :ca:8f), Dst: Cisco-Li_45:1 (192.168.1.101), Dst: 128.1 57522), Dst Port: http (80) HTTP/1.1\r\n T 6.1; en-US; rv:1.9.2.22) ation/xml;q=0.9,*/*;q=0.8\r</pre>	f:1b (00:22:6b:45:1f:1b) 19.245.12 (128.119.245.12) , Seq: 1, Ack: 1, Len: 435 Gecko/20110902 Firefox/3.6.22 (.NET CLR =	, 3. 5. 30729)∖r∖n
Frame 835: 489 bytes on Ethernet II, Src: HonHai Internet Protocol Versite Transmission Control Prot GET Wyrestark-labs/TH Host: gaia.cs.umass.ee User-Agent: Mozilla/S. Accept: text/html,app Accept-tencoding: gzip, Accept-charset: ISO-88 Keep-Alive: 115\r\n Connection: keep-alive \r\n	<pre>wire (3912 bits), 489 by iPr_Odica:8f (00:22:68:0d n4, Src: 192.168.1.101 btocol, Src Port: 57522 (1 cool utRo-wireshark-file1.html du\r\n 0 (windows; U; Windows N iication/xhtml+xml,applic; s,en;q=0.5\r\n ,deflate\r\n 359=1,utf-8;q=0.7,*;q=0.7 e\r\n</pre>	<pre>tes captured (3912 bits) :ca:8f), Dst: Cisco-Li_45:1 (192.168.1.101), Dst: 128.1 57522), Dst Port: http (80) HTTP/1.1\r\n T 6.1; en-US; rv:1.9.2.22) ation/xml;q=0.9,*/*;q=0.8\r</pre>	f:1b (00:22:6b:45:1f:1b) 19.245.12 (128.119.245.12) , Seq: 1, Ack: 1, Len: 435 Gecko/20110902 Firefox/3.6.22 (.NET CLR =	, 3. 5. 30729)∖r∖n
Frame 835: 489 bytes on Ethernet II, Src: HonHai Internet Protocol Versic Transmission Control Prot GET Wrieshark-labs/Th Host: gaia.cs.umass.ec User-Agent: Mozilla/S. Accept:text/hml,app Accept-Language: en-us Accept-Encoding: gzip, Accept-Charset: ISO-88 Keep-Alive: 115\r\n Connection: keep-alive \r\n [Full request URI: htt]	<pre>wire (3912 bits), 489 by iPr_Odica:8f (00:22:68:0d ptocol, src: 192.168.1.101 ptocol, src Port: 57522 (: cool wTRO-wireshark-file1.html du/r\n 0 (windows; U; windows N lication/xhtml+xml,applics ,en;q=0.5\r\n deflate\r\n 359-1.utf-8;q=0.7,*;q=0.7 e\r\n tp://gaia.cs.umass.edu/win tp://gaia.cs.umass.edu/win 122 68 0d ca 8f 08 00 45 80 16 89 b3 d9 41 b1 83 35 145 54 20 2f 77 69 72 65</pre>	<pre>tes captured (3912 bits) :ca:8f), Dst: Cisco-Li_45:1 (192.168.1.101), Dst: 128.1 57522), Dst Port: http (80) HTTP/1.1\r\n T 6.1; en-US; rv:1.9.2.22) ation/xml;q=0.9,*/*;q=0.8\r \r\n reshark-labs/INTRO-wireshar 0 00 ."KE" hE. 0 77).@e.W 18Pe.W 19Pe.W 19e.W 19Pe.W 19Pe.W 19Pe.W 19Pe.W 19Pe.W 19Pe.W 19PPe.W 19P</pre>	f:1b (00:22:6b:45:1f:1b) 19.245.12 (128.119.245.12) , Seq: 1, Ack: 1, Len: 435 Gecko/20110902 Firefox/3.6.22 (.NET CLR =	, 3. 5. 30729)∖r∖n

Congratulations! You've now completed the first lab.

Figure 5: Wireshark window after step 9

 $^{^{2}}$ 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 text³. 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 following⁴:

- 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.

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

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

<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)

File	- 14		'iresha				-										_									_ 🗆
	<u>E</u> dit	<u>V</u> iew	Go	<u>C</u> apti	ure	<u>A</u> naly:	ze S	itatistic	:s <u>H</u> ≀	elp		1	_													
	Ë.	0	1 <u>@</u>	((×	C) (-	x	¢,	8		4	4	\$	¢2	Ŧ	_₽			\$		€	Θ,	O,	÷
jiter:														▼ <u>E</u> >	pressio	n ⊆	lear	Apply								
lo. +		Time			Sourc	e				Desti	nation				Protoco	ol Infi	o									
	_		0000 0606			168. 119.					.119 .168				TCP TCP										S=14 :=1 W	
	3 (0.05	0729		192.	168.	.2.1	45		128	.119	.245	5.12		TCP	20	38 :	> htt	p	ACK]	Seq	=1	Ack=	1 Wi	n=65	535
			5906 8700			$\frac{168}{119}$.119				HTTP TCP										l-la Win=	
	6 (0.13	4167		128.	119.	. 245	.12		192.	.168	.2.1	L4 5 👘		TCP	[Τ	CP :	segme	int	of a	i rea	sse	mble	d PD	0]	0432
			0302 0487			119. 168.					.168				TCP TCP						i rea Sec				00] '62 W	in=6
	9 (0.21	3639		128.	119.	.245	.12		192.	.168	.2.1	L4 5 👘		TCP	[т	CP :	segme	ent	of a	a rea	SSE	mble	d PD)U]	
			5724 5947			$\frac{119}{168}$.168 .119				TCP TCP										9 seg 14 W	
	12 (0.23	1749		128.	119.	.245	.12		192.	.168	.2.1	L4 5		HTTP	[Τ	CP P	Retra	insr	nissi	ion]	HTT	Ρ/1.	1 20	00 ок	(te
			2145 0470			168. 168.					.119 .119				TCP HTTP						Seq HTTP			k=48	310 W	in=6
	15 (0.40	3428		128.	119.	.245	.12		192.	.168	.2.1	L4 5 👘		HTTP	HT	TP/:	1.1 4	04	Not	Foun	d (text			
			3932 9522			168. 168.					.66.: .119				TCP TCP			> htt > htt							S=14 35 W	
	18 3	3.38	3584		192.	168.	.2.1	45		168.	.66.:	12.2	224		TCP	20	39.0	> htt	:p	[SYN]	Seq				S=14	
			2197 89131			168. 119.					.66.: .168				TCP TCP			> htt > 203							S=14 Ack=	
			89258			168.					.119				TCP		_	> 203 > htt		- /					36 W	
			506 k II, S											:6d)	, Dst	: Lir	nksy	′sG_4	5:9	0:a8	(00:		:41:4	15:9		
Eti In Tr Hyj	heri teri ansr beri	net net niss text	II, S Proto <mark>ion C</mark> Trar	orc: ocol cont ont	Net , Sr <mark>rol</mark> r Pr	geai c: 1 Prot oto	r_61 192. <mark>toco</mark> col	:8e: 168. 1, s	6d 2.14 Src ((00: 45 () Port	09:5 192. : 20	b:63 168. 38	1:8e: .2.14 <mark>(2038</mark>	45), 3), I	Dst: Dst P	128. ort:	. 119	.245	.12	(12	8.119	:0c 9.24	45.12	2))
Etl In Tr. Hy C	nerri Insr Dert R(R(R) Nost Jser Acco Acco (cep	net net iss text eque eque eque c-Ag ept- ept- ept- o-Al	II, S Proto Inn C Tran resha st Me st UR st UR aia.c ent: text Langu Encoc Chars ive:	ocol ont osfe etho ersi s.u Moz /xm uage ing et: 300	Net , Sr rol r Pr labs d: C /wir on: mass illa illa ; er : gz ISC \r\r	:gear Prot oto /HT aET resh: HTT s.edu x/5.(ppli(n-us, zip,()-88) n	r_61 192. toco col TP-e p/1. u\r\ 0 (w cati ,en; defl 59-1	:8e: 168. 1, 5 1abs 1 (indo on/x q=0. ,utf	6d 2.14 rc (eal- :/HT ws; ml,; 5\r` ,r\n	(00: 15 () Port -lab (P-e U; ' appl (n	09:5 192. : 20 -fil ther wind icat	b:61 168 38 e3.1 eal- ows ion,	1:8e: .2.14 (2038 html -lab- NT ! /xhtm	45), 3), HTTI -fild 5.1; nl+xr	Dst:	128. ort: \r\n ml S; r\	.119 htt	.245 p (8	.12 0), 4)	(12 Seq Geck	8.119 : 1,	:0c 9.2/ AC	45.1; k: 1, 515 (2) Le	n: 45	52 2.0.1
Fr. Etl In Hy Hy	nerri Insr Dert R(R(R) Nost Jser Acco Acco (cep	net net niss text /wi eque equ	II, S Proto ion C Trar resha st Me st Ve aia.c ent: text Langu Encoo Chars	ocol ont osfe etho ersi s.u Moz /xm uage ing et: 300	Net , Sr rol r Pr labs d: C /wir on: mass illa illa ; er : gz ISC \r\r	:gear Prot oto /HT aET resh: HTT s.edu x/5.(ppli(n-us, zip,()-88) n	r_61 192. toco col TP-e p/1. u\r\ 0 (w cati ,en; defl 59-1	:8e: 168. 1, 5 1abs 1 (indo on/x q=0. ,utf	6d 2.14 rc (eal- :/HT ws; ml,; 5\r` ,r\n	(00: 15 () Port -lab (P-e U; ' appl (n	09:5 192. : 20 -fil ther wind icat	b:61 168 38 e3.1 eal- ows ion,	1:8e: .2.14 (2038 html -lab- NT ! /xhtm	45), 3), HTTI -fild 5.1; nl+xr	Dst: Dst P P/1.1 e3.ht en-U	128. ort: \r\n ml S; r\	.119 htt	.245 p (8	.12 0), 4)	(12 Seq Geck	8.119 : 1,	:0c 9.2/ AC	45.1; k: 1, 515 (2) Le	n: 45	2
Fr. Etl In Hy Hy I	nerr terr ansr aer R(R(R) Acc(Acc) Acc) Acc) Con	net net niss text /wi eque equ	II, S Proto Inn C Tran resha st Me st UR st UR aia.c ent: text Langu Encoc Chars ive:	ocol ont osfe etho ersi s.u Moz /xm uage ing et: 300	Net , Sr rol r Pr labs d: C /wir on: mass illa illa ; er : gz ISC \r\r	:gear Prot oto /HT aET resh: HTT s.edu x/5.(ppli(n-us, zip,()-88) n	r_61 192. toco col TP-e p/1. u\r\ 0 (w cati ,en; defl 59-1	:8e: 168. 1, 5 1abs 1 (indo on/x q=0. ,utf	6d 2.14 rc (eal- :/HT ws; ml,; 5\r` ,r\n	(00: 15 () Port -lab (P-e U; ' appl (n	09:5 192. : 20 -fil ther wind icat	b:61 168 38 e3.1 eal- ows ion,	1:8e: .2.14 (2038 html -lab- NT ! /xhtm	45), 3), HTTI -fild 5.1; nl+xr	Dst: Dst P P/1.1 e3.ht en-U	128. ort: \r\n ml S; r\	.119 htt	.245 p (8	.12 0), 4)	(12 Seq Geck	8.119 : 1,	:0c 9.2/ AC	45.1; k: 1, 515 (2) Le	n: 45) 52
Fr. Etl In Hy Hy	nerri ansr ansr Re Re Re Nost Jser Jser Jser Acco Acco (ceep Jonn (r \r R Acco (ceep Jonn (r \r) (a 33 (a 60 60	net net niss text eque eque eque eque eque eque eque equ	II, S Proto Trar resha st Me st Ve aia.c ent: text Langu Encoc Chars ive: ion:	src: cont sfe a asfe asfe a asfe as as asfe a a as as a a a a	Net , Sr r Pr labs d: C /wir on: masss ill; er : gz ISC \r\r\p-al 661 661 661 662 663 664 665 665	(gean c:: Proto (fHT) EET cesh (fHT) (cedu (a/5.) (cedu)(a/5.) (cedu (a/5.) (cedu)(r_61 192. toco col TP-e p/1. 0 (w cati ,en; ,en; fo fo fo 2 2 4 8 67 75	:8e: 168. 1, 5 1abs 1 1 1 1 1 1 1 1 1 1 1 1 1	6d 1 2.14 rc 1 ws; /HT 5\r\n -8;0 21 4 61 6 54 5 69 6 0a 5	(00: -1ab -1ab rP-e U; ' appl n -1ab -2 -2 -2 -2 -2 -2 -2 -2 -2 -2	09:5 192. 20 -fil ther wind icat 7,*; 54 466 466 31	b:62 168. 38 e3.1 eal- ows ion, q=0. 50 69 2e 73 72	1:8e .2.14 (2038 html -lab- lab- /xhtm .7\r 2d 66 66 6	45), 1 HTTI -fili 5.1; nl+xr \n 5 5 d 5 5 1	Dst: Dst P P/1.1 e3.ht en-U	128. ort: \r\n ml S; r\ xt/ht xt/ht eal- il H :equ	.119 htt /:1. tml; is/H lab. TTP/ ala 	245 p (8 8.1. q=0.	.12 0), 4) 9,t	(12 Seq Geck	8.119 : 1,	:0c 9.2/ AC	45.1; k: 1, 515 (2) Le	n: 45	2.0.

• 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:

			eshark	and we	Acaluse	Chabial	tica L	tele.												<u>- 0 ×</u>
Eile	Edit	<u>V</u> iew		apture	<u>A</u> nalyze	<u>S</u> tatis	×	lelp Raj	8	9	4	\$	¢2	Ŧ	Ŧ		Ţ	•	e,	O,
Eilter:											₹Ę	xpressior	n ⊆	lear <u>A</u>	pply					
No. +	•	Time		Source	:e			Destin	ation			Protoco	l Info							<u> </u>
	2 (3 ().000).050).050	606 729	Lin Net	gear_6 <sysg_ gear_6</sysg_ 	45:90 1:8e:	:a8 6d	Neto Link	jear_6 :sysG_	51:8e: _45:90	6d):a8	0×080 0×080 0×080)0 IP)0 IP							
	5 (6 (7 (9 (10 (112 (112 (113 (114 (115 (117 (118 9 20 1	0.055 0.128 0.134 0.150 0.213 0.215 0.231 0.232 0.320 0.403 0.403 0.423 0.579 0.383 0.392 0.38 0.38 0.38	700 167 302 487 639 724 947 749 145 470 428 932 522 584 197 9131	Linl Linl Linl Linl Net Net Net Net Net	cear_6 (sysG_ (sysG_ (sysG_ gear_6 (sysG_ gear_6 gear_6 gear_6 gear_6 gear_6 gear_6 gear_6 gear_6 gear_6 gear_6	45:90 45:90 45:90 1:8e: 45:90 1:8e: 45:90 1:8e: 1:8e: 1:8e: 1:8e: 1:8e: 1:8e: 1:8e: 1:8e: 1:8e: 1:8e:	: a8 : a8 : 6d : a8 : 6d : a8 : 6d : a8 : 6d : 6d : 6d : 6d : 6d : 6d : 6d : 6d	Neto Neto Neto Link Neto Link Link Link Link Link Neto	jear_6 jear_6 jear_6 sysG_ jear_6 sysG_ sysG_ sysG_ sysG_ sysG_ sysG_ sysG_ sysG_	51:8e: 51:8e: 51:8e: 51:8e: 51:8e: 51:8e: 45:90 51:8e: 45:90 51:8e: 45:90 45:90 45:90 45:90 51:8e: 51:8e:	6d 6d 6d 6d 6d 6d 6d 1:a8 6d 1:a8 1:a8 1:a8 1:a8 1:a8 1:a8 1:a8 1:a8	0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080	00 IP 00 IP							
	herr Dest Ac Sour Ac Type	dres dres ce: dres	I, Sr ion: I s: Li 0 0 Netge s: Ne 0	c: Ne _inksys hksys ar_61 tgear 300)	n wire tgear_ ysG_45 G_45:9 	_61:8e 5:90:a8 90:a8 	2:6d (00: (00: = 09:5 00:0 =	(00:0 0:0c:41 IG t LG t b:61 9:5b IG t	09:5b: 41:45:9 Dit: 1 Dit: 0 :8e:60 :61:86 Dit: 1	:61:86 5:90:a8 Indiv Globa d) e:6d) Indiv	a8)) idual lly u idual	addro nique addro	ess (addr ess (unica ess unica	ast) (fact ast)	ory c	lefaul		45:90): a8)
0000 0010 0020 0030 0040 0050 0060 0070 File: "C	01 f5 68 74 33 0a	ec 8 0c 0 ff 3 68 8 2e 8 48 8	87 e9 97 f6 a 9c 2 6b 55 72 58 74 5f 73	40 00 00 50 20 60 65 61 6d 60 74 3a	3 00 0 3 80 0 3 74 7 47 4 5 61 6 5 6C 2 6 2 6 2 4 2 0 4 1 20 6 TALS~1	6 38 4 c4 5 54 2 73 d 6c 8 54 7 61	65 58 20 2f 61 54 59 69	c0 a8 7d a6 2f 77 48 54 52 2d 50 2f 51 2e	02 9 27 9 69 7 54 5 66 6 31 2 63 7	0 45 1 80 0 50 2 65 0 2d 9 6c e 31 3 2e 3ytes 00	77 18 73 65 65 0d 75	AE. @ hark- there 3.htm .Host 21 D: 21	Pzt . .GE 1 .GE 1 .lab : eal- il H 1	Ве .×}.' т /wi s/нтт lab-f ттр/1 аiа.с	w .P. res P-e ile 1.					

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 printout⁵ 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?

⁵ 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 following⁶:

- 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:

⁶ 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).

🗖 eth	erne	t-ethe	real-tr	ace-1	- Wiresha	rk																_ 🗆 🗵
Eile	<u>E</u> dit	<u>V</u> iew	<u>G</u> o y	<u>C</u> apture	<u>A</u> nalyze	<u>S</u> tatis	itics <u>H</u>	elp														
E.	Ë.		<u></u>		D	8	×	¢.	8	ঐ	4		¢	Ŧ	⊉				Ð,	Θ,	0	FF
Eilter:											▼ <u>E</u>	xpression	n <u>⊂</u>	ear <u>A</u>	pply							
No. +		Time		So	urce			Destina				Protoco		·								<u> </u>
	2 0 3 0 4 5 7 8 9 10 11 12 13 14 15 16	17.44 17.46 17.46 17.49 17.49 17.50 17.50 17.50 17.52	.018 .028 .850	Li Am Am Te Am Li Am Li Li Li Li	Ditwic nksysG bitMic bitMic bitMic lebit bitMic bitMic bitMic bitMic nksysG nksysG nksysG nksysG bitMic bitMic bitMic bitMic bitMic bitMic bitMic bitMic	_da:af _a9:30 _a9:30 73:8d: _a9:30 _da:af _da:af _da:af _da:af _da:af _da:af _da:af	:73 1:68 1:68 1:68 1:68 1:68 1:68 1:68 1:68	Ambi Link: Link: Broau Link: Ambi Ambi Link: Ambi Link: Ambi	tMic_ sysG_ sysG_ tMic_ tMic_ sysG_ tMic_ tMic_	da:af da:af da:af da:af da:af da:af da:af a9:3d da:af a9:3d da:af a9:3d	:73 :73 :73 :68 :73 :68 :68 :68 :68 :73 :68 :68 :73 :68 :68	0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080 0×080	19 0 IP 0 IP 0 IP 0 IP 0 IP 0 IP 0 IP 0 IP	2.168	3.1.1	is a	1.1? at 00: .1.117	06:2	25:d	a:af	:73	105
 Fr Et Ad 	ame therr ldres Harr Prot Opco Seno Seno Tar	1 (4 net) ss Re dware toco dware toco ode: der N der J get N	12 by 13 solut 2 solut 2 size 1 size	tes c rc: A tion e: Et e: IF e: 6 e: 4 est (ddres ddress ddress	n wire mbitMi Protoco hernet (0x08) 0x0001 s: Amb : 192.: s: 00:	. 42 k a9:3 ol (re (0x00)00) itMic_ L68.1. 00:00_		capt (00: t) d:68 (192. 0:00	ured) d0:59 (00:d 168.1 (00:0	:a9:3 0:59: .105) 0:00:	d:68 a9:3), Ds1 d:68)		oadc	ast I	(ff:f	f:ff:	ff:f	f:ff			
0000 0010 0020	ff 08 00	- ff 00 00	FF FF 06 04 00 00	ff 00 00	: 192.: Ff 00 c 01 00 c 00 c0 a aula Wing\	0 59 0 59 8 01	a9 3 a9 3 01	d 68 d 68	08 00 c0 a8	5 00 1	59			r.=h.								

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 text⁷. 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 following⁸:

- 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.

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

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

<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: Packets: Sent = 10, Received = 10, Lost = 0 (0% loss), 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.

M icn	np-et	hereal-	trace-1	- Wire	shark																<u>- 🗆 ×</u>
Eile	<u>E</u> dit	⊻iew	<u>G</u> o <u>C</u> a	apture	<u>A</u> nalyz	e <u>S</u> tat	istics	Help													
	2	0	@)	C	, 🖁	x	eg	8	٩	4	⇔	¢	Ŧ	⊉		Ţ		•	Θ	0
<u>Filter</u> :	icm	пр									▼ E	xpression	<u>⊂</u> le	ear	Apply						
No		Time		Sour	се			Destir	nation			Protocol	Info	I							4
		0.001				1.101	-		.89.14			ICMP			ping)						
L		0.415 1.006			.89.1	.4.34 1.101			.168.1 .89.14			ICMP ICMP			ping) pinq)						
L		1.431			.89.1		•		168.1			ICMP			ping)						
		2.006				1.101			89.14			ICMP	Ech	10 (ping)	requ	est				
		2.324			.89.1				.168.1			ICMP			ping)						
-		3.006 3.321			.168.	1.101	•		.89.14 .168.1			ICMP ICMP			ping) ping)						
		4.006				1.101			89.14			ICMP			ping)						
	12 4	4.343	301	143	.89.1	4.34		192.	.168.1	.101		ICMP	Ech	10 (ping)	reply	y 👘				
		5.006				1.101			.89.14			ICMP			ping)						
		5.365 6.022			.89.1	.4.34 1.101			.168.1 .89.14			ICMP ICMP			ping) ping)						
		6.403			.89.1		•		168.1			ICMP			ping)						
		7.022				1.101			89.14			ICMP			ping)						
		7.423			.89.1				.168.1			ICMP			ping)						
		B.022 B.423			.168.	1.101	-		.89.14 .168.1			ICMP ICMP			ping) ping)						
L		9.022				1.101			.89.14			ICMP			ping)						
		9.432			.89.1				168.1			ICMP			ping)						
₹																					F
E Er	ame	3 (7	4 hvt/	es on	wire	P. 74	byte	s can	tured))											
											36:23), Dst	: Li	nks	vsG_da	a:af:7	3 (0)	0:00	5 : 25	:da:a	(f:73)
												Dst:									, i
		sion:		,																	
	Неа	der 1	ength	: 20	bytes	;															
±	Dif	feren	tiate	d ser	vices	; Fiel	ld: 0	x00 (I	DSCP (:00×0	Defa	ult; E	CN:	0×0	0)						
	Tot	al Le	ngth:	60																	
	Ide	ntifi	catio	n: Ox	d1fd	(5375	57)														
•	Fla	gs: O	×00																		
	Fra	gment	offse	et: 0																	
	тim	e to	live:	128																	
	Pro	tocol	: ICMP	P (0x	:01)																
÷	Неа	der c	hecksi	um: O	x093b) [cor	rect]													
	Sou	rce:	192.10	68.1.	101 ((192.1	.68.1	.101)													
	Des:	tinat	ion: 1	143.8	9.14.	34 (1	43.8	9.14.3	34)												
🗉 II	nteri	net c	ontro	l Mes	sage	Proto	lool														
4																					
0010	00	300	di fd	00 00	0 80	01 n	9 3b	CO 25	3 01 6	5 8f	59	. <	·		.e.Y						<u> </u>
0020	0e	22 [00 80	e4 5a	а О2	00 6	7 01	61 62	63 6	4 65_	66	ghijk	z c	j.ab	cdef						
0030			59 6a			6e 6		71 72	73 7	4 75	76	ghijk	lmn c	pqr	stuv						
0040	77	OT (52 63	64 6.	00	07 b	8 69					wabcd	erg r	T I							
Interne	et Con	trol Mes	sage Prol	tocol (ic	mn). 40	hytes					P: 22	2 D; 20 M;	:0								<u> </u>
ancon n	2000	011103	Jagerro			5,005					1.20	. Di 20 Mi									11.

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.

Elle Edit Yiew Go Capture Analyze Statistics Help Image: Statistics Media Image: Statistics Help Image: Statistics Help Image: Statistics Media Image: Statistics Help Image: Statistics Help Image: Statistics Media Image: Statistics Help Image: Statistics Help Image: Statistics Media Image: Statistics Help Image: Statistics Help Image: Statistics Media Image: Statistics Help Image: Statistics Help Image: Statistics Media Image: Statistics Help Image: Statistics Help Image: Statistics Media Image: Statistics Help Image: Statistics Help Image: Statistics Media Media		Q	Q,
Elter: icmp Expression Clear Apply No Time Source Destination Protocol Info 3 0.001656 192.168.1.101 143.89.14.34 ICMP Echo (ping) reguly 4 0.415098 143.89.14.34 192.168.1.101 ICMP Echo (ping) reguly 5 1.006279 192.168.1.101 143.89.14.34 ICMP Echo (ping) reguly 7 2.006328 192.168.1.101 143.89.14.34 ICMP Echo (ping) reguly 8 2.324479 143.89.14.34 192.168.1.101 ICMP Echo (ping) reguly 9 3.006356 192.168.1.101 143.89.14.34 ICMP Echo (ping) reguly 10 3.321121 143.89.14.34 192.168.1.101 ICMP Echo (ping) reguly 11 4.006398 192.168.1.101 143.89.14.34 ICMP Echo (ping) reguly		Θ,	Q,
No Time Source Destination Protocol Info 3 0.001656 192.168.1.101 143.89.14.34 ICMP Echo (ping) request 4 0.415098 143.89.14.34 192.168.1.101 ICMP Echo (ping) reply 5 1.006279 192.168.1.101 143.89.14.34 ICMP Echo (ping) reply 7 2.006328 192.168.1.101 143.89.14.34 ICMP Echo (ping) reply 7 2.006328 192.168.1.101 143.89.14.34 ICMP Echo (ping) reply 9 3.006356 192.168.1.101 143.89.14.34 ICMP Echo (ping) reply 9 3.006356 192.168.1.101 143.89.14.34 ICMP Echo (ping) reply 10 3.321121 143.89.14.34 192.168.1.101 ICMP Echo (ping) reply 11 4.006398 192.168.1.101 143.89.14.34 ICMP Echo (ping) reply			
3 0.001656 192.168.1.101 143.89.14.34 ICMP Echo (ping) request 4 0.415098 143.89.14.34 192.168.1.101 ICMP Echo (ping) reply 5 1.006279 192.168.1.101 143.89.14.34 ICMP Echo (ping) request 6 1.431684 143.89.14.34 192.168.1.101 ICMP Echo (ping) request 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) request 9 3.006356 192.168.1.101 143.89.14.34 ICMP Echo (ping) request 10 3.321121 143.89.14.34 192.168.1.101 ICMP Echo (ping) request 11 4.006398 192.168.1.101 143.89.14.34 ICMP Echo (ping) request			
4 0.415098 143.89.14.34 192.168.1.101 ICMP Echo (ping) reply 5 1.006279 192.168.1.101 143.89.14.34 ICMP Echo (ping) request 6 1.431684 143.89.14.34 192.168.1.101 ICMP Echo (ping) request 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) request 9 3.006356 192.168.1.101 143.89.14.34 ICMP Echo (ping) request 10 3.321121 143.89.14.34 192.168.1.101 ICMP Echo (ping) request 11 4.006398 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) request 6 1.431684 143.89.14.34 192.168.1.101 ICMP Echo (ping) request 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) request 9 3.006356 192.168.1.101 143.89.14.34 ICMP Echo (ping) request 10 3.321121 143.89.14.34 192.168.1.101 ICMP Echo (ping) request 11 4.006398 192.168.1.101 143.89.14.34 ICMP Echo (ping) request			
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) reply 10 3.21121 143.89.14.34 192.168.1.101 ICMP Echo (ping) reply 11 4.306398 192.168.1.101 143.89.14.34 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 10 3.321121 143.89.14.34 192.168.1.101 ICMP Echo (ping) reply 11 4.006398 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 10 3.321121 143.89.14.34 192.168.1.101 ICMP Echo (ping) reply 11 4.006398 192.168.1.101 143.89.14.34 ICMP Echo (ping) request			
9 3.006356 192.168.1.101 143.89.14.34 ICMP Echo (ping) request 10 3.321121 143.89.14.34 192.168.1.101 ICMP Echo (ping) reply 11 4.006398 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) request			
17 4 242201 142 90 14 24 102 169 1 101 TOMP Tobe (pipe) poply			
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			
17 7.022213 192.168.1.101 143.89.14.34 ISCHORE ECHO (ping) repry			
18 7.423214 143.89.14.34 192.168.1.101 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			
		- 1	
🗄 Frame 3 (74 bytes on wire, 74 bytes captured)			
Ethernet II, Src: DellComp_4f:36:23 (00:08:74:4f:36:23), Dst: LinksysG_da:af:73 (5:da:	af:73)
⊞ Internet Protocol, Src: 192.168.1.101 (192.168.1.101), Dst: 143.89.14.34 (143.89.1	14.34)		
Internet Control Message Protocol			
Type: 8 (Echo (ping) request)			
Code: 0			
Checksum: 0xe45a [correct]			
Identifier: 0x0200			
Sequence number: 26369 (0x6701)			
Data (32 bytes)			
0000 00 06 25 da af 73 00 08 74 4f 36 23 08 00 45 00%s to6#E.			
0010 00 3c d1 fd 00 00 80 01 09 3b c0 a8 01 65 8f 59			
0020 0e 22 08 00 e4 5a 02 00 67 01 61 62 63 64 65 66 ."z g.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			
Internet Control Message Protocol (icmp), 40 bytes P: 22 D: 20 M: 0			1.

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 printout⁹ 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:

⁹ 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?