计算机网络实验手册

华东师范大学软件工程学院

目　录

实验一 Protocol Layer 2

实验二 Ethernet 10

实验三 IPV4 18

实验四 ARP 27

实验五 UDP 37

实验六 TCP 43

实验七 Socket Programming 52

附录 华东师范大学软件工程学院实验报告 57

实验一 Protocol Layer

Objective

To learn how protocols and layering are represented in packets. They are key concepts for structuring networks that are covered in §1.3 and §1.4 of your text. Review those sections before doing the lab.

Requirements

Wireshark: This lab uses the Wireshark software tool to capture and examine a packet trace. A packet trace is a record of traffic at a location on the network, as if a snapshot was taken of all the bits that passed across a particular wire. The packet trace records a timestamp for each packet, along with the bits that make up the packet, from the lower-layer headers to the higher-layer contents. Wireshark runs on most operating systems, including Windows, Mac and Linux. It provides a graphical UI that shows the sequence of packets and the meaning of the bits when interpreted as protocol headers and data. It color-codes packets by their type, and has various ways to filter and analyze packets to let you investigate the behavior of network protocols. Wireshark is widely used to troubleshoot networks. You can download it from www.wireshark.org if it is not already installed on your computer. We highly recommend that you watch the short, 5 minute video "Introduction to Wireshark" that is on the site.

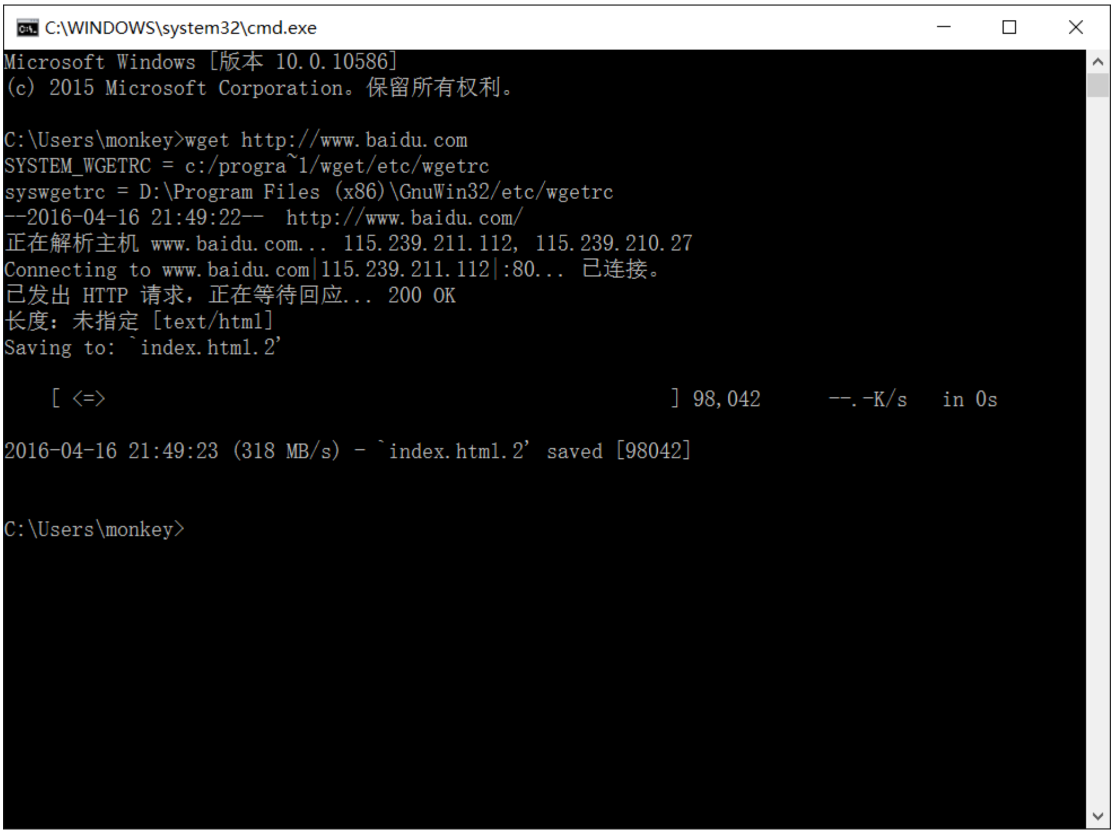
wget / curl: This lab uses wget (Linux and Windows) and curl (Mac) to fetch web resources. wget and curl are command-line programs that let you fetch a URL. Unlike a web browser, which fetches and executes entire pages, wget and curl give you control over exactly which URLs you fetch and when you fetch them. Under Linux, wget can be installed via your package manager. Under Windows, wget is available as a binary; look for download information on http://www.gnu.org/software/wget/. Under Mac, curl comes installed with the OS. Both have many options (try "wget --help" or "curl --help" to see) but a URL can be fetched simply with "wget *URL*" or "curl *URL* ".

Step 1: Capture a Trace

Proceed as follows to capture a trace of network traffic; alternatively, you may use a supplied trace. We want this trace to look at the protocol structure of packets. A simple Web fetch of a URL from a server of your choice to your computer, which is the client, will serve as traffic.

1. *Pick a URL and fetch it with* wget *or* curl. For example, "wget http://www.google.com" or "curl http://www.google.com". This will

fetch the resource and either write it to a file (`wget`) or to the screen (`curl`). You are checking to see that the fetch works and retrieves some content. A successful example is shown below (with added highlighting) for `wget`. You want a single response with status code “200 OK”. If the fetch does not work then try a different URL; if no URLs seem to work then debug your use of `wget`/`curl` or your Internet connectivity.

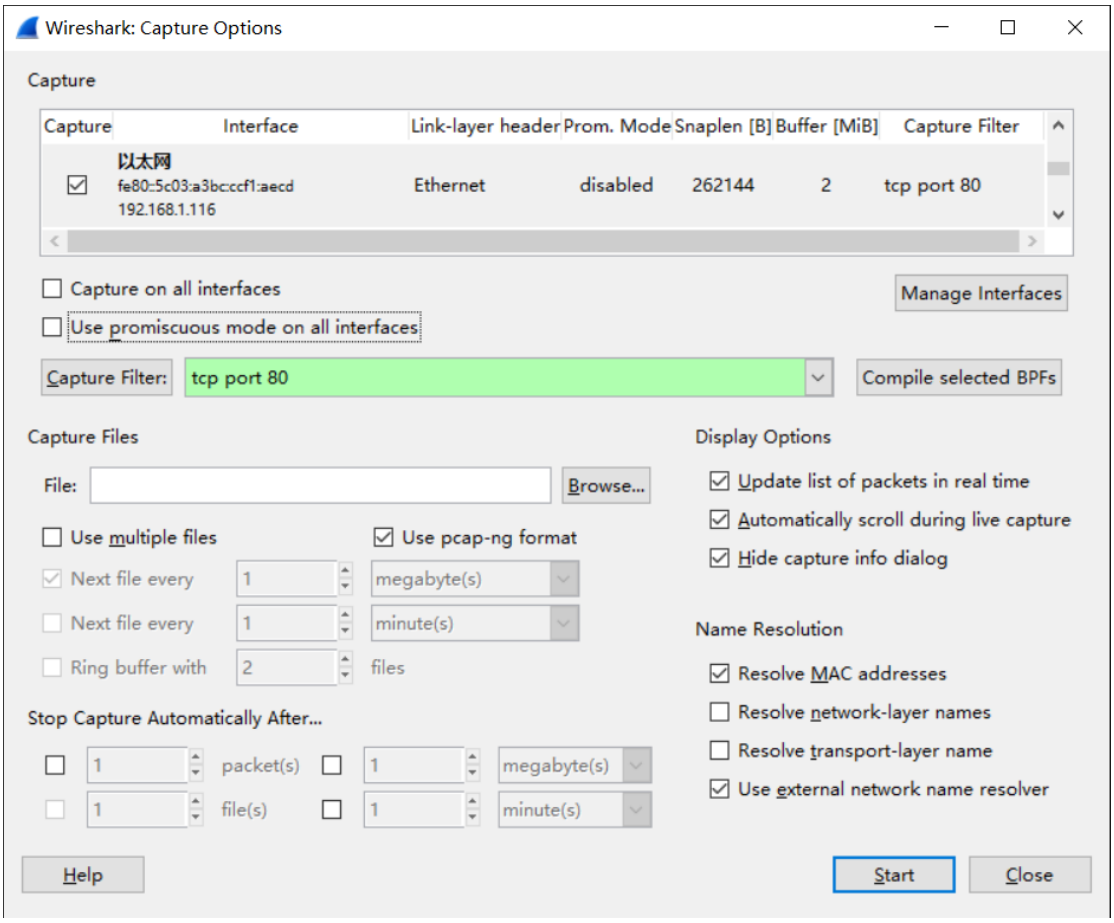


```
Microsoft Windows [版本 10.0.10586]
(c) 2015 Microsoft Corporation。保留所有权利。

C:\Users\monkey>wget http://www.baidu.com
SYSTEM_WGETRC = c:/progra~1/wget/etc/wgetrc
syswgetrc = D:\Program Files (x86)\GnuWin32/etc/wgetrc
--2016-04-16 21:49:22--  http://www.baidu.com/
正在解析主机 www.baidu.com... 115.239.211.112, 115.239.210.27
Connecting to www.baidu.com|115.239.211.112|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：未指定 [text/html]
Saving to: `index.html.2'

    [ <=>                                                ] 98,042      --.-K/s    in 0s

2016-04-16 21:49:23 (318 MB/s) - `index.html.2' saved [98042]


C:\Users\monkey>
```

Figure 1: Using `wget` to fetch a URL

2. *Close unnecessary browser tabs and windows*. By minimizing browser activity you will stop your computer from fetching unnecessary web content, and avoid incidental traffic in the trace.
3. *Launch Wireshark and start a capture with a filter of* “`tcp port 80`” and check “enable network name resolution”. This filter will record only standard web traffic and not other kinds of packets that your computer may send. The checking will translate the addresses of the computers sending and receiving packets into names, which should help you to recognize whether the packets are going to or from your computer. Your capture window should be similar to the one pictured below, other than our highlighting. Select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful. Uncheck “capture packets in promiscuous mode”. This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets

sent to/from your computer. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive. On Wireshark 1.8, the capture filter box is present directly on the options screen, but on Wireshark 1.9, you set a capture filter by double-clicking on the interface.



Wireshark: Capture Options

Capture

Capture	Interface	Link-layer header	Prom. Mode	Snaplen [B]	Buffer [MiB]	Capture Filter
☑	以太网 fe80::5c03:a3bc:ccf1:aecd 192.168.1.116	Ethernet	disabled	262144	2	tcp port 80

☐ Capture on all interfaces

☐ Use promiscuous mode on all interfaces

Capture Filter: tcp port 80

Capture Files

File:

☐ Use multiple files ☑ Use pcap-ng format

☑ Next file every 1 megabyte(s)

☐ Next file every 1 minute(s)

☐ Ring buffer with 2 files

Stop Capture Automatically After...

☐ 1 packet(s) ☐ 1 megabyte(s)

☐ 1 file(s) ☐ 1 minute(s)

Display Options

☑ Update list of packets in real time

☑ Automatically scroll during live capture

☑ Hide capture info dialog

Name Resolution

☑ Resolve MAC addresses

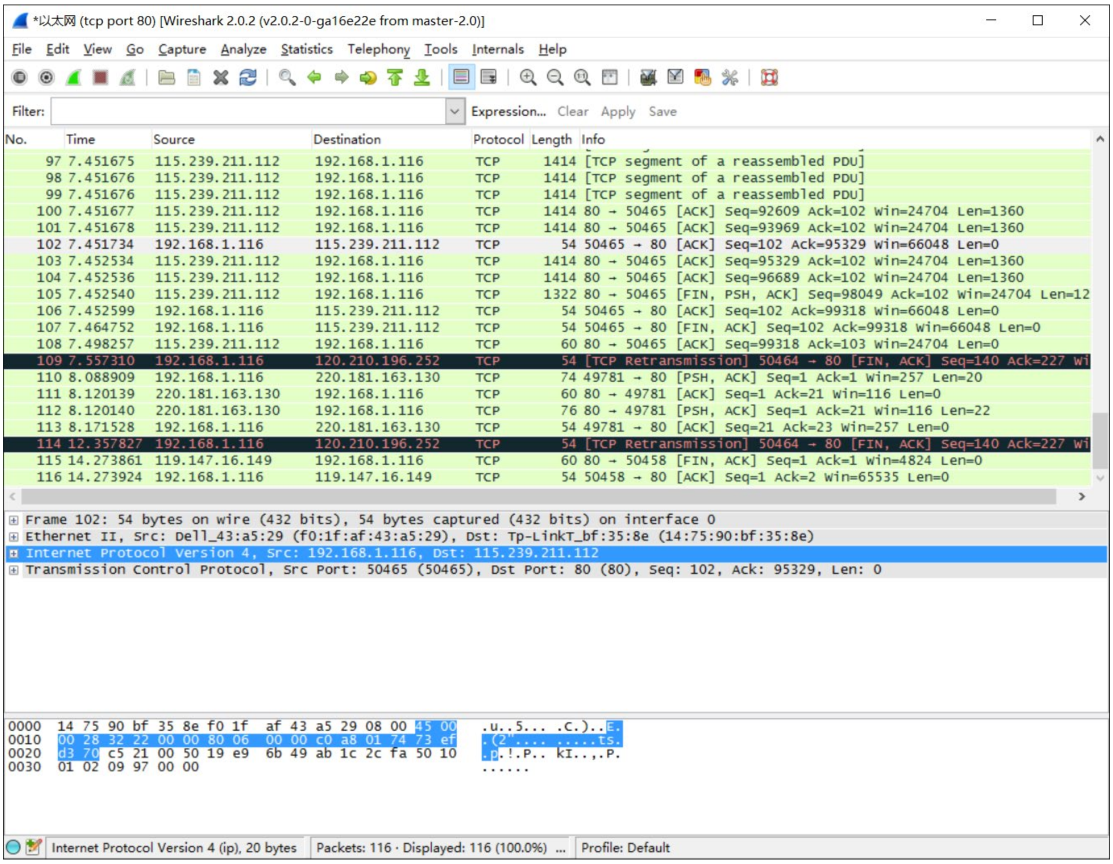
☐ Resolve network-layer names

☐ Resolve transport-layer name

☑ Use external network name resolver

Figure 2: Setting up the capture options

4. *When the capture is started, repeat the web fetch using* `wget/curl` *above.* This time, the packets will be recorded by Wireshark as the content is transferred.
5. *After the fetch is successful, return to Wireshark and use the menus or buttons to stop the trace.* If you have succeeded, the upper Wireshark window will show multiple packets, and most likely it will be full. How many packets are captured will depend on the size of the web page, but there should be at least 8 packets in the trace, and typically 20-100, and many of these packets will be colored green. An example is shown below. Congratulations, you have captured a trace!


*以太网 (tcp port 80) [Wireshark 2.0.2 (v2.0.2-0-ga16e22e from master-2.0)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time       Source           Destination      Protocol Length Info
  97 7.451675   115.239.211.112  192.168.1.116    TCP      1414 [TCP segment of a reassembled PDU]
  98 7.451676   115.239.211.112  192.168.1.116    TCP      1414 [TCP segment of a reassembled PDU]
  99 7.451676   115.239.211.112  192.168.1.116    TCP      1414 [TCP segment of a reassembled PDU]
 100 7.451677   115.239.211.112  192.168.1.116    TCP      1414 80 → 50465 [ACK] Seq=92609 Ack=102 Win=24704 Len=1360
 101 7.451678   115.239.211.112  192.168.1.116    TCP      1414 80 → 50465 [ACK] Seq=93969 Ack=102 Win=24704 Len=1360
 102 7.451734   192.168.1.116    115.239.211.112  TCP        54 50465 → 80 [ACK] Seq=102 Ack=95329 Win=66048 Len=0
 103 7.452534   115.239.211.112  192.168.1.116    TCP      1414 80 → 50465 [ACK] Seq=95329 Ack=102 Win=24704 Len=1360
 104 7.452536   115.239.211.112  192.168.1.116    TCP      1414 80 → 50465 [ACK] Seq=96689 Ack=102 Win=24704 Len=1360
 105 7.452540   115.239.211.112  192.168.1.116    TCP      1322 80 → 50465 [FIN, PSH, ACK] Seq=98049 Ack=102 Win=24704 Len=12
 106 7.452599   192.168.1.116    115.239.211.112  TCP        54 50465 → 80 [ACK] Seq=102 Ack=99318 Win=66048 Len=0
 107 7.464752   192.168.1.116    115.239.211.112  TCP        54 50465 → 80 [FIN, ACK] Seq=102 Ack=99318 Win=66048 Len=0
 108 7.498257   115.239.211.112  192.168.1.116    TCP        60 80 → 50465 [ACK] Seq=99318 Ack=103 Win=24704 Len=0
 109 7.557310   192.168.1.116    120.210.196.252  TCP        54 [TCP Retransmission] 50464 → 80 [FIN, ACK] Seq=140 Ack=227 Wi
 110 8.088909   192.168.1.116    220.181.163.130  TCP        74 49781 → 80 [PSH, ACK] Seq=1 Ack=1 Win=257 Len=20
 111 8.120139   220.181.163.130  192.168.1.116    TCP        60 80 → 49781 [ACK] Seq=1 Ack=21 Win=116 Len=0
 112 8.120140   220.181.163.130  192.168.1.116    TCP        76 80 → 49781 [PSH, ACK] Seq=1 Ack=21 Win=116 Len=22
 113 8.171528   192.168.1.116    220.181.163.130  TCP        54 49781 → 80 [ACK] Seq=21 Ack=23 Win=257 Len=0
 114 12.357827  192.168.1.116    120.210.196.252  TCP        54 [TCP Retransmission] 50464 → 80 [FIN, ACK] Seq=140 Ack=227 Wi
 115 14.273861  119.147.16.149   192.168.1.116    TCP        60 80 → 50458 [FIN, ACK] Seq=1 Ack=1 Win=4824 Len=0
 116 14.273924  192.168.1.116    119.147.16.149   TCP        54 50458 → 80 [ACK] Seq=1 Ack=2 Win=65535 Len=0

⊞ Frame 102: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
⊞ Ethernet II, Src: Dell_43:a5:29 (f0:1f:af:43:a5:29), Dst: Tp-LinkT_bf:35:8e (14:75:90:bf:35:8e)
⊞ Internet Protocol Version 4, Src: 192.168.1.116, Dst: 115.239.211.112
⊞ Transmission Control Protocol, Src Port: 50465 (50465), Dst Port: 80 (80), Seq: 102, Ack: 95329, Len: 0

0000  14 75 90 bf 35 8e f0 1f  af 43 a5 29 08 00 45 00   .u..5... .C.)..E.
0010  00 28 32 22 00 00 80 06  00 00 c0 a8 01 74 73 ef   .(2".... .....ts.
0020  d3 70 c5 21 00 50 19 e9  6b 49 ab 1c 2c fa 50 10   .p.!.P.. kI..,.P.
0030  01 02 09 97 00 00                                  ......

Internet Protocol Version 4 (ip), 20 bytes | Packets: 116 · Displayed: 116 (100.0%) ... | Profile: Default
```

Figure 3: Packet trace of `wget` traffic

Step 2: Inspect the Trace

Wireshark will let us select a packet (from the top panel) and view its protocol layers, in terms of both header fields (in the middle panel) and the bytes that make up the packet (in the bottom panel). In the figure above, the first packet is selected (shown in blue). Note that we are using "packet" as a general term here. Strictly speaking, a unit of information at the link layer is called a frame. At the network layer it is called a packet, at the transport layer a segment, and at the application layer a message. Wireshark is gathering frames and presenting us with the higher-layer packet, segment, and message structures it can recognize that are carried within the frames. We will often use "packet" for convenience, as each frame contains one packet and it is often the packet or higher-layer details that are of interest.

Select a packet for which the Protocol column is "HTTP" and the Info column says it is a GET. It is the packet that carries the web (HTTP) request sent from your computer to the server. (You can click the column headings to sort by that value, though it should not be difficult to find an HTTP packet by inspection.) Let's have a closer look to see how the packet structure reflects the protocols that are in use.

Since we are fetching a web page, we know that the protocol layers being used are as shown below. That is, HTTP is the application layer web protocol used to fetch URLs. Like many Internet applications, it runs on top of the TCP/IP transport and network layer protocols. The link and physical layer protocols depend on your network, but are typically combined in the form of Ethernet (shown) if your computer is wired, or 802.11 (not shown) if your computer is wireless.

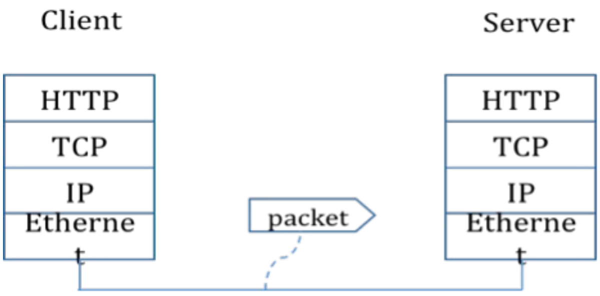


Figure 4: Protocol stack for a web fetch

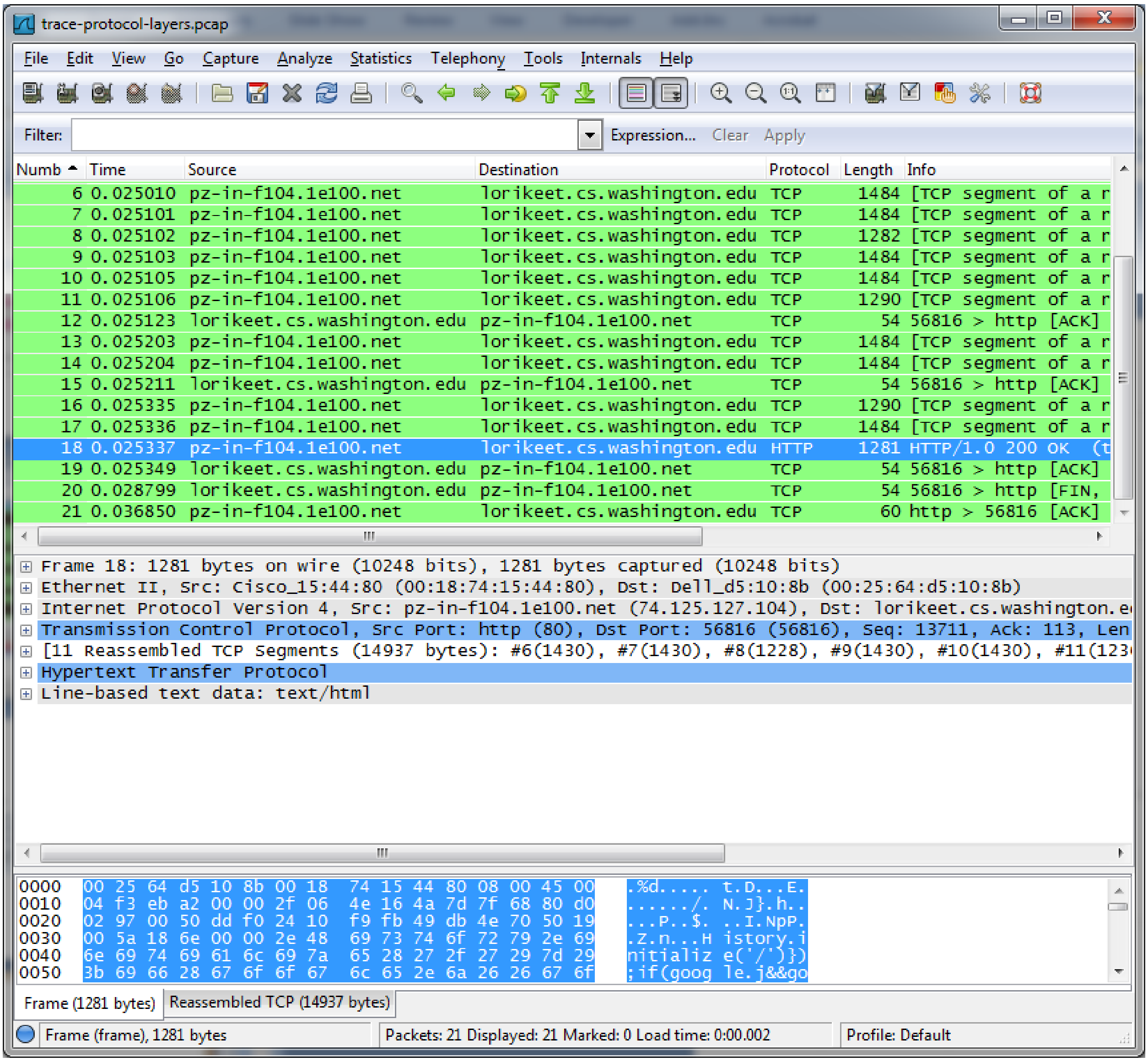
With the HTTP GET packet selected, look closely to see the similarities and differences between it and our protocol stack as described next. The protocol blocks are listed in the middle panel. You can expand each block (by clicking on the "+" expander or icon) to see its details.

- The first Wireshark block is "Frame". This is not a protocol, it is a record that describes overall information about the packet, including when it was captured and how many bits long it is.
- The second block is "Ethernet". This matches our diagram! Note that you may have taken a trace on a computer using 802.11 yet still see an Ethernet block instead of an 802.11 block. Why? It happens because we asked Wireshark to capture traffic in Ethernet format on the capture options, so it converted the real 802.11 header into a pseudo-Ethernet header.
- Then come IP, TCP, and HTTP, which are just as we wanted. Note that the order is from the bottom of the protocol stack upwards. This is because as packets are passed down the stack, the header information of the lower layer protocol is added to the front of the information from the higher layer protocol, as in Fig. 1-15 of your text. That is, the lower layer protocols come first in the packet "on the wire".

Now find another HTTP packet, the response from the server to your computer, and look at the structure of this packet for the differences compared to the HTTP GET packet. This packet should have "200 OK" in the Info field, denoting a successful fetch. In our trace, there are two extra blocks in the detail panel as seen in the next figure.

- The first extra block says "[11 reassembled TCP segments …]". Details in your capture will vary, but this block is describing more than the packet itself. Most likely, the web response was sent across the network as a series of packets that were put together after they arrived at the computer. The packet labeled HTTP is the last packet in the web response, and the block lists packets that are joined together to obtain the complete web response. Each of these packets is shown as having protocol TCP even though the packets carry part of an HTTP response. Only the final packet is shown as having protocol HTTP when the complete HTTP message may be understood, and it lists the packets that are joined together to make the HTTP response.

- The second extra block says “Line-based text data …”. Details in your capture will vary, but this block is describing the contents of the web page that was fetched. In our case it is of type text/html, though it could easily have been text/xml, image/jpeg, or many other types. As with the Frame record, this is not a true protocol. Instead, it is a description of packet contents that Wireshark is producing to help us understand the network traffic.


trace-protocol-layers.pcap
File
Edit
View
Go
Capture
Analyze
Statistics
Telephony
Tools
Internals
Help
Filter:
Expression...
Clear
Apply
Frame (1281 bytes)
Reassembled TCP (14937 bytes)
Frame (frame), 1281 bytes
Packets: 21 Displayed: 21 Marked: 0 Load time: 0:00.002
Profile: Default

```
Numb  Time     Source                    Destination               Protocol Length Info
   6 0.025010 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP    1484 [TCP segment of a r
   7 0.025101 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP    1484 [TCP segment of a r
   8 0.025102 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP    1282 [TCP segment of a r
   9 0.025103 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP    1484 [TCP segment of a r
  10 0.025105 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP    1484 [TCP segment of a r
  11 0.025106 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP    1290 [TCP segment of a r
  12 0.025123 lorikeet.cs.washington.edu pz-in-f104.1e100.net     TCP      54 56816 > http [ACK]
  13 0.025203 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP    1484 [TCP segment of a r
  14 0.025204 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP    1484 [TCP segment of a r
  15 0.025211 lorikeet.cs.washington.edu pz-in-f104.1e100.net     TCP      54 56816 > http [ACK]
  16 0.025335 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP    1290 [TCP segment of a r
  17 0.025336 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP    1484 [TCP segment of a r
  18 0.025337 pz-in-f104.1e100.net      lorikeet.cs.washington.edu HTTP   1281 HTTP/1.0 200 OK  (t
  19 0.025349 lorikeet.cs.washington.edu pz-in-f104.1e100.net     TCP      54 56816 > http [ACK]
  20 0.028799 lorikeet.cs.washington.edu pz-in-f104.1e100.net     TCP      54 56816 > http [FIN,
  21 0.036850 pz-in-f104.1e100.net      lorikeet.cs.washington.edu TCP      60 http > 56816 [ACK]
```

```
⊞ Frame 18: 1281 bytes on wire (10248 bits), 1281 bytes captured (10248 bits)
⊞ Ethernet II, Src: Cisco_15:44:80 (00:18:74:15:44:80), Dst: Dell_d5:10:8b (00:25:64:d5:10:8b)
⊞ Internet Protocol Version 4, Src: pz-in-f104.1e100.net (74.125.127.104), Dst: lorikeet.cs.washington.e
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 56816 (56816), Seq: 13711, Ack: 113, Len
⊞ [11 Reassembled TCP Segments (14937 bytes): #6(1430), #7(1430), #8(1228), #9(1430), #10(1430), #11(123
⊞ Hypertext Transfer Protocol
⊞ Line-based text data: text/html
```

```
0000  00 25 64 d5 10 8b 00 18  74 15 44 80 08 00 45 00   .%d..... t.D...E.
0010  04 f3 eb a2 00 00 2f 06  4e 16 4a 7d 7f 68 80 d0   ....../. N.J}.h..
0020  02 97 00 50 dd f0 24 10  f9 fb 49 db 4e 70 50 19   ...P..$. ..I.NpP.
0030  00 5a 18 6e 00 00 2e 48  69 73 74 6f 72 79 2e 69   .Z.n...H istory.i
0040  6e 69 74 69 61 6c 69 7a  65 28 27 2f 27 29 7d 29   nitializ e('/')})
0050  3b 69 66 28 67 6f 6f 67  6c 65 2e 6a 26 26 67 6f   ;if(goog le.j&&go
```

Figure 5: Inspecting a HTTP “200 OK” response

Step 3: Packet Structure

To show your understanding of packet structure, draw a figure of an HTTP GET packet that shows *the position and size in bytes of the TCP, IP and Ethernet protocol headers.* Your figure can simply show the overall packet as a long, thin rectangle. Leftmost elements are the first sent on the wire. On this drawing, show the range of the Ethernet header and the Ethernet payload that IP passed to Ethernet to send over the network. To show the nesting structure of protocol layers, note the range of the IP header and the IP payload. You may have questions about the fields in each protocol as you look at them. We will explore these protocols and fields in detail in future labs.

To work out sizes, observe that when you click on a protocol block in the middle panel (the block itself, not the "+" expander) then Wireshark will highlight the bytes it corresponds to in the packet in the lower panel and display the length at the bottom of the window. For instance, clicking on the IP version 4 header of a packet in our trace shows us that the length is 20 bytes. (Your trace will be different if it is IPv6, and may be different even with IPv4 depending on various options.) You may also use the overall packet size shown in the Length column or Frame detail block.

Turn-in: Hand in your packet drawing.

Step 4: Protocol Overhead

Estimate the download protocol overhead, or percentage of the download bytes taken up by protocol overhead. To do this, consider HTTP data (headers and message) to be useful data for the network to carry, and lower layer headers (TCP, IP, and Ethernet) to be the overhead. We would like this overhead to be small, so that most bits are used to carry content that applications care about. To work this out, first look at only the packets in the download direction for a single web fetch. You might sort on the Destination column to find them. The packets should start with a short TCP packet described as a SYN ACK, which is the beginning of a connection. They will be followed by mostly longer packets in the middle (of roughly 1 to 1.5KB), of which the last one is an HTTP packet. This is the main portion of the download. And they will likely end with a short TCP packet that is part of ending the connection. For each packet, you can inspect how much overhead it has in the form of Ethernet / IP / TCP headers, and how much useful HTTP data it carries in the TCP payload. You may also look at the HTTP packet in Wireshark to learn how much data is in the TCP payloads over all download packets.

Turn-in: Your estimate of download protocol overhead as defined above. Tell us whether you find this overhead to be significant.

Step 5: Demultiplexing Keys

When an Ethernet frame arrives at a computer, the Ethernet layer must hand the packet that it contains to the next higher layer to be processed. The act of finding the right higher layer to process received packets is called demultiplexing. We know that in our case the higher layer is IP. But how does the Ethernet protocol know this? After all, the higher-layer could have been another protocol entirely (such as ARP). We have the same issue at the IP layer – IP must be able to determine that the contents of IP message is a TCP packet so that it can hand it to the TCP protocol to process. The answer is that protocols use information in their header known as a "demultiplexing key" to determine the higher layer.

Look at the Ethernet and IP headers of a download packet in detail to answer the following questions:

1. *Which Ethernet header field is the demultiplexing key that tells it the next higher layer is IP? What value is used in this field to indicate "IP"?*
2. *Which IP header field is the demultiplexing key that tells it the next higher layer is TCP? What value is used in this field to indicate "TCP"?*

Turn-in: Hand in your answers to the above questions.

Explore on your own

We encourage you to explore protocols and layering once you have completed this lab. Some ideas:

- Look at a short TCP packet that carries no higher-layer data. To what entity is this packet destined? After all, if it carries no higher-layer data then it does not seem very useful to a higher layer protocol such as HTTP!
- In a classic layered model, one message from a higher layer has a header appended by the lower layer and becomes one new message. But this is not always the case. Above, we saw a trace in which the web response (one HTTP message comprised of an HTTP header and an HTTP payload) was converted into multiple lower layer messages (being multiple TCP packets). Imagine that you have drawn the packet structure (as in step 2) for the first and last TCP packet carrying the web response. How will the drawings differ?
- In the classic layered model described above, lower layers append headers to the messages passed down from higher layers. How will this model change if a lower layer adds encryption?
- In the classic layered model described above, lower layers append headers to the messages passed down from higher layers. How will this model change if a lower layer adds compression?

[END]

实验二 Ethernet

Objective

To explore the details of Ethernet frames. Ethernet is a popular link layer protocol that is covered in §4.3 of your text; modern computers connect to Ethernet switches (§4.3.4) rather than use classic Ethernet (§4.3.2). Review section §4.3 before doing this lab.

Requirements

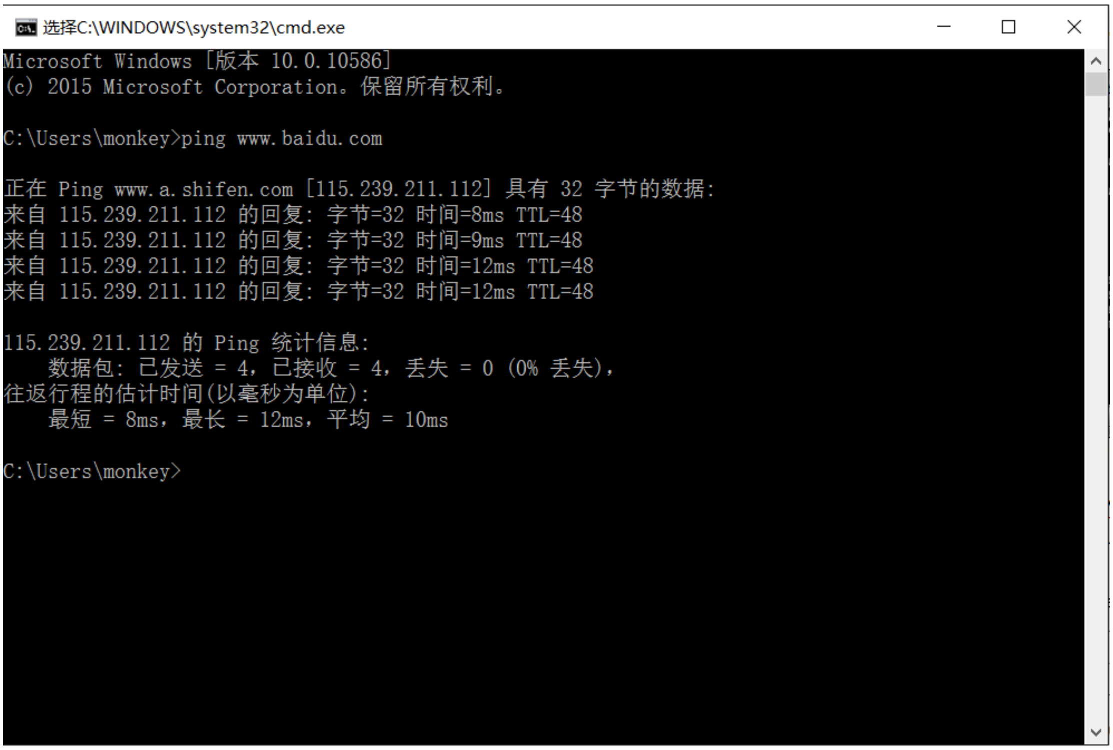
Wireshark: This lab uses the Wireshark software tool to capture and examine a packet trace. A packet trace is a record of traffic at a location on the network, as if a snapshot was taken of all the bits that passed across a particular wire. The packet trace records a timestamp for each packet, along with the bits that make up the packet, from the lower-layer headers to the higher-layer contents. Wireshark runs on most operating systems, including Windows, Mac and Linux. It provides a graphical UI that shows the sequence of packets and the meaning of the bits when interpreted as protocol headers and data. It color-codes packets by their type, and has various ways to filter and analyze packets to let you investigate the behavior of network protocols. Wireshark is widely used to troubleshoot networks. You can download it from www.wireshark.org if it is not already installed on your computer. We highly recommend that you watch the short, 5 minute video "Introduction to Wireshark" that is on the site.
ping: This lab uses "`ping`" to send and receive messages. `ping` is a standard command-line utility for checking that another computer is responsive. It is widely used for network troubleshooting and comes pre-installed on Window, Linux, and Mac. While `ping` has various options, simply issuing the command "`ping www.bing.com`" will cause your computer to send a small number of ICMP ping requests to the remote computer (here www.bing.com), each of which should elicit an ICMP ping response.

Step 1: Capture a Trace

Proceed as follows to capture a trace of ping packets; alternatively you may use a supplied trace. We will use ping simply as an easy way to collect a small trace. Perhaps surprisingly, you can capture a trace for this lab from a computer connected to the Internet using either wired Ethernet or wireless 802.11.

1. *Pick a remote web server or other publicly reachable Internet host and use* `ping` *to send some ping messages and check that it sends replies*. For example, "`ping`

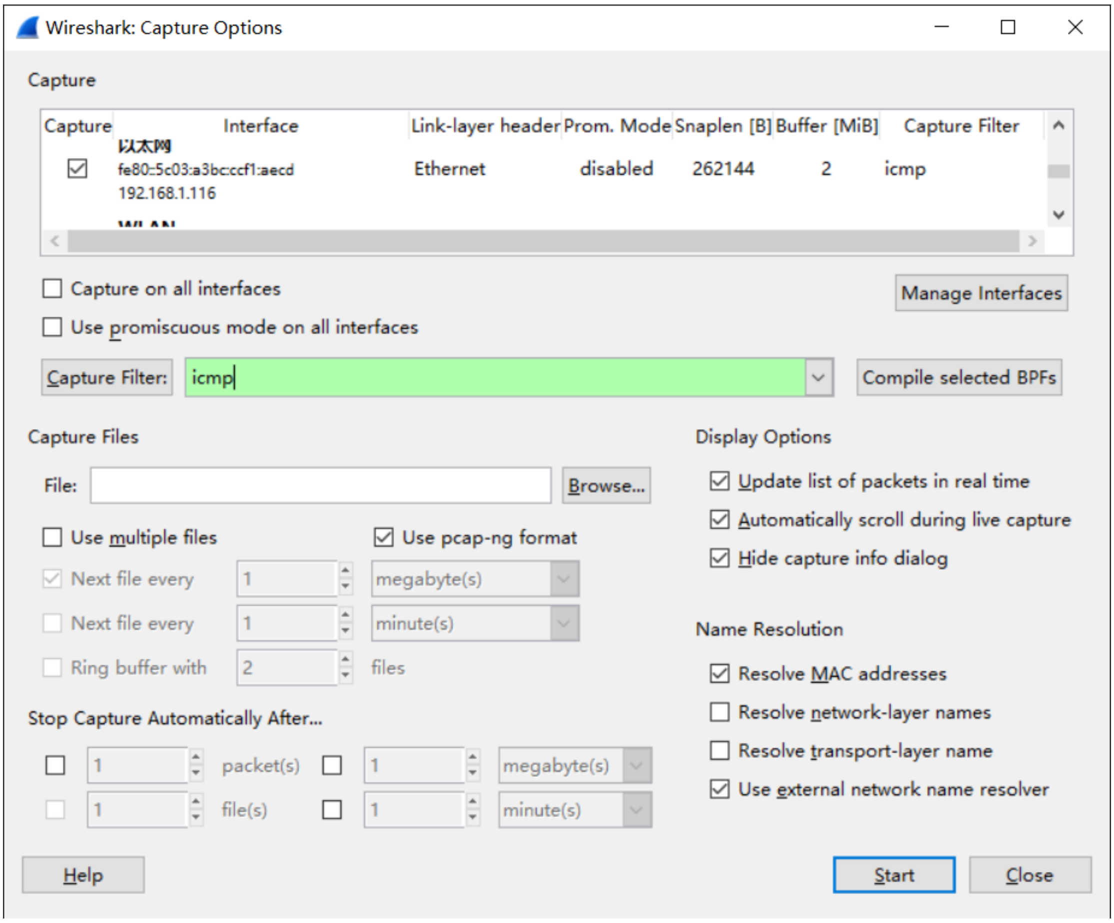
www.bing.com". You should see several replies indicating that the pings reached the remote host and were returned. The figure below shows a successful example. Note that some versions of ping will continue to bounce messages off of a remote server until you tell the program to stop by signaling it with ^C. If your ping test does not succeed then try another server.



```
Microsoft Windows [版本 10.0.10586]
(c) 2015 Microsoft Corporation。保留所有权利。

C:\Users\monkey>ping www.baidu.com

正在 Ping www.a.shifen.com [115.239.211.112] 具有 32 字节的数据:
来自 115.239.211.112 的回复: 字节=32 时间=8ms TTL=48
来自 115.239.211.112 的回复: 字节=32 时间=9ms TTL=48
来自 115.239.211.112 的回复: 字节=32 时间=12ms TTL=48
来自 115.239.211.112 的回复: 字节=32 时间=12ms TTL=48

115.239.211.112 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
往返行程的估计时间(以毫秒为单位):
    最短 = 8ms，最长 = 12ms，平均 = 10ms

C:\Users\monkey>
```

Figure 6: Using ping to bounce messages off a remote host

2. *Launch Wireshark and start a capture of Ethernet frames with a filter of "icmp", making sure that "enable MAC name resolution" is checked.* The latter will translate Ethernet (MAC) addresses to provide vendor information. Also check that the Link-layer header type pulldown says "Ethernet". Your capture window should be similar to the one pictured below, other than our highlighting. Select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful. Uncheck "capture packets in promiscuous mode". This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets sent to/from your computer. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive. On Wireshark 1.8, the capture filter box is present directly on the options screen, but on Wireshark 1.9, you set a capture filter by double-clicking on the interface.



Wireshark: Capture Options

Capture

Capture	Interface	Link-layer header	Prom. Mode	Snaplen [B]	Buffer [MiB]	Capture Filter
☑	以太网 fe80::5c03:a3bc:ccf1:aecd 192.168.1.116	Ethernet	disabled	262144	2	icmp
	WLAN					

□ Capture on all interfaces

□ Use promiscuous mode on all interfaces

Capture Filter: icmp

Capture Files

File:

□ Use multiple files ☑ Use pcap-ng format

☑ Next file every 1 megabyte(s)

□ Next file every 1 minute(s)

□ Ring buffer with 2 files

Stop Capture Automatically After...

□ 1 packet(s) □ 1 megabyte(s)

□ 1 file(s) □ 1 minute(s)

Display Options

☑ Update list of packets in real time

☑ Automatically scroll during live capture

☑ Hide capture info dialog

Name Resolution

☑ Resolve MAC addresses

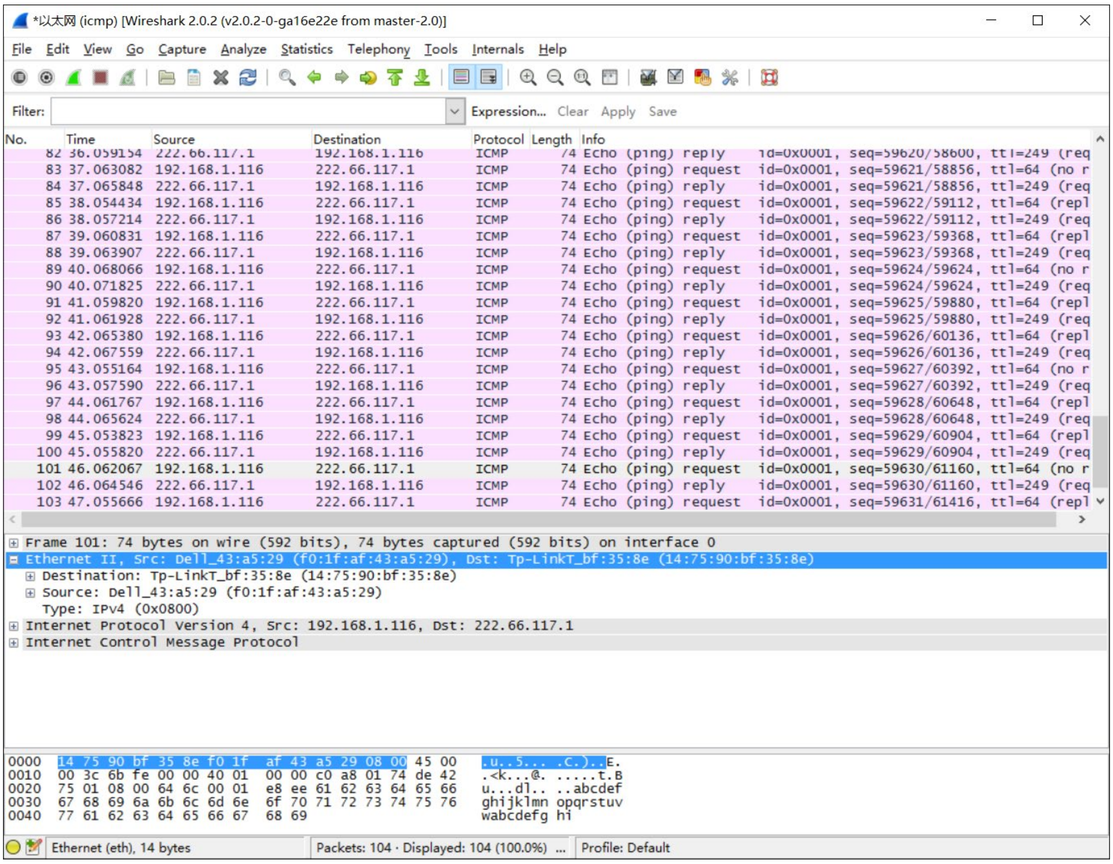
□ Resolve network-layer names

□ Resolve transport-layer name

☑ Use external network name resolver

Figure 7: Setting the capture options for `ping` traffic

3. *When the capture is started, repeat the* `ping` *command above.* This time, the packets will also be recorded by Wireshark.
4. *After the* `ping` *command is complete, return to Wireshark and use the menus or buttons to stop the trace.* You should now have a short trace similar to that shown in the figure below. If you do not succeed in capturing a trace then use the supplied one. Note that the trace we supply begins with ping messages, and then has other kinds of Ethernet frames.



*以太网 (icmp) [Wireshark 2.0.2 (v2.0.2-0-ga16e22e from master-2.0)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time       Source         Destination    Protocol Length Info
  82 36.059154 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59620/58600, ttl=249 (req
  83 37.063082 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59621/58856, ttl=64 (no r
  84 37.065848 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59621/58856, ttl=249 (req
  85 38.054434 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59622/59112, ttl=64 (repl
  86 38.057214 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59622/59112, ttl=249 (req
  87 39.060831 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59623/59368, ttl=64 (repl
  88 39.063907 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59623/59368, ttl=249 (req
  89 40.068066 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59624/59624, ttl=64 (no r
  90 40.071825 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59624/59624, ttl=249 (req
  91 41.059820 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59625/59880, ttl=64 (repl
  92 41.061928 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59625/59880, ttl=249 (req
  93 42.065380 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59626/60136, ttl=64 (repl
  94 42.067559 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59626/60136, ttl=249 (req
  95 43.055164 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59627/60392, ttl=64 (no r
  96 43.057590 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59627/60392, ttl=249 (req
  97 44.061767 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59628/60648, ttl=64 (repl
  98 44.065624 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59628/60648, ttl=249 (req
  99 45.053823 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59629/60904, ttl=64 (repl
 100 45.055820 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59629/60904, ttl=249 (req
 101 46.062067 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59630/61160, ttl=64 (no r
 102 46.064546 222.66.117.1   192.168.1.116  ICMP     74 Echo (ping) reply    id=0x0001, seq=59630/61160, ttl=249 (req
 103 47.055666 192.168.1.116  222.66.117.1   ICMP     74 Echo (ping) request  id=0x0001, seq=59631/61416, ttl=64 (repl
```

```
⊞ Frame 101: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
⊟ Ethernet II, Src: Dell_43:a5:29 (f0:1f:af:43:a5:29), Dst: Tp-LinkT_bf:35:8e (14:75:90:bf:35:8e)
  ⊞ Destination: Tp-LinkT_bf:35:8e (14:75:90:bf:35:8e)
  ⊞ Source: Dell_43:a5:29 (f0:1f:af:43:a5:29)
    Type: IPv4 (0x0800)
⊞ Internet Protocol Version 4, Src: 192.168.1.116, Dst: 222.66.117.1
⊞ Internet Control Message Protocol
```

```
0000  14 75 90 bf 35 8e f0 1f  af 43 a5 29 08 00 45 00   .u..5... .C.)..E.
0010  00 3c 6b fe 00 00 40 01  00 00 c0 a8 01 74 de 42   .<k...@. .....t.B
0020  75 01 08 00 64 6c 00 01  e8 ee 61 62 63 64 65 66   u...dl.. ..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
```

Ethernet (eth), 14 bytes | Packets: 104 · Displayed: 104 (100.0%) ... | Profile: Default

Figure 8: Trace of `ping` traffic, showing Ethernet details of the first packet

Step 2: Inspect the Trace

Select any packet in the trace (in the top panel) to see details of its structure (in the middle panel) and the bytes that make up the packet (in the bottom panel). Now we can inspect the details of the packets. In the figure, we have selected the first packet in the trace. Note that we are using the term "packet" in a loose way. Each record captured by Wireshark more correctly corresponds to a single frame in Ethernet format that carries a packet as its payload; Wireshark interprets as much structure as it can.

In the middle panel, expand the Ethernet header fields (using the "+" expander or icon) to see their details. Our interest is the Ethernet header, and you may ignore the higher layer protocols (which are IP and ICMP in this case). You can click on the Ethernet header to see the bytes that correspond to it in the packet highlighted in the bottom panel. We have performed both steps in the figure.

If you are capturing traffic over an 802.11 interface, you may wonder why you have an Ethernet header at all, instead of an 802.11 header. This happens because we asked Wireshark to capture packets in Ethernet format on the capture options (in Figure 2). In this case, the OS software converted the real 802.11 header into a pseudo-Ethernet header. We are seeing the pseudo-Ethernet header.

Compare the fields you see with the picture of an Ethernet frame in Fig. 4-14 of your text. You will see both similarities and differences:

- There are two kinds of Ethernet shown in your book, IEEE 802.3 and DIX Ethernet. IEEE 802.3 is rare and you are not likely to see it. The frames in the figure and likely your capture are DIX Ethernet, called “Ethernet II” in Wireshark.
- There is no preamble in the fields shown in Wireshark. The preamble is a physical layer mechanism to help the NIC identify the start of a frame. It carries no useful data and is not received like other fields.
- There is a destination address and a source address. Wireshark is decoding some of these bits in the OUI (Organizationally Unique Identifier) portion of the address to tell us the vendor of the NIC, e.g., Dell for the source address.
- There is a Type field. For the ping messages, the Ethernet type is IP, meaning the Ethernet payload carries an IP packet. (There is no Length field as in the IEEE 802.3 format. Instead, the length of a DIX Ethernet frame is determined by the hardware of a receiving computer, which looks for valid frames that start with a preamble and end with a correct checksum, and passed up to higher layers along with the packet.)
- There is no Data field per se – the data starts with the IP header right after the Ethernet header.
- There is no pad. A pad will be present at the end if the frame would otherwise be less than 64 bytes, the minimum Ethernet frame size.
- There is no checksum in most traces, even though it really does exist. Typically, Ethernet hardware that is sending or receiving frames computes or checks this field and adds or strips it. Thus it is simply not visible to the OS or Wireshark in most capture setups.
- There are also no VLAN fields such as the Tag shown in Fig. 4-49. If VLANs are in use, the VLAN tags are normally added and removed by switch ports so they will not be visible at host computers using the network.

Step 3: Ethernet Frame Structure

To show your understanding of the Ethernet frame format, draw a figure of the ping message that shows the position and size in bytes of the Ethernet header fields. Your figure can simply show the frame as a long, thin rectangle. The leftmost fields come first in the packet and are sent on the wire first. On this drawing, show the range of the Ethernet header and the Ethernet payload. Add a dashed box at the end to represent the 4-byte checksum; we know it is there even if Wireshark does not show us this field.

To work out sizes, observe that when you click on a protocol block in the middle panel (the block itself, not the “+” expander) then Wireshark will highlight the bytes it corresponds to in the packet in the lower panel and display the length at the bottom of the window. You may also use the overall packet size shown in the Length column or Frame detail block.

Turn-in: Hand in your drawing of an Ethernet frame.

Step 4: Scope of Ethernet Addresses

Each Ethernet frame carries a source and destination address. One of these addresses is that of your computer. It is the source for frames that are sent, and the destination for frames that are received. But what is the other address? Assuming you pinged a remote Internet server, it cannot be the Ethernet address of the remote server because an Ethernet frame is only addressed to go within one LAN. Instead, it will be the Ethernet address of the router or default gateway, such as your AP in the case of 802.11. This is the device that connects your LAN to the rest of the Internet. In contrast, the IP addresses in the IP block of each packet do indicate the overall source and destination endpoints. They are your computer and the remote server.

Draw a figure that shows the relative positions of your computer, the router, and the remote server. Label your computer and the router with their Ethernet addresses. Label your computer and the remote server with their IP addresses. Show where the Ethernet and the rest of the Internet fit on the drawing.

Turn-in: Hand in your drawing.

Step 5: Broadcast Frames

The trace that you gathered above captured unicast Ethernet traffic sent between a specific source and destination, e.g., your computer to the router. It is also possible to send multicast or broadcast Ethernet traffic, destined for a group of computers or all computers on the Ethernet, respectively. We can tell from the address whether it is unicast, multicast, or broadcast. Broadcast traffic is sent to a reserved Ethernet address that has all bits set to “1”. Multicast traffic is sent to addresses that have a “1” in the first bit sent on the wire; broadcast is a special case of multicast. Broadcast and multicast traffic is widely used for discovery protocols, e.g., a packet sent to everyone in an effort to find the local printer.

Start a capture for broadcast and multicast Ethernet frames with a filter of “`ether multicast`”, *wait up to 30 seconds to record background traffic, and then stop the capture. If you do not capture any packets with this filter then use the trace that we supplied.* On most Ethernets, there is a steady chatter of background traffic as computers exchange messages to maintain network state, which is why we try to capture traffic without running any other programs. The capture filter of “`ether multicast`” will capture both multicast and broadcast Ethernet frames, but not regular unicast frames. You may have to wait a little while for these packets to be captured, but on most LANs with multiple computers you will see at least a packet every few seconds.

Examine the multicast and broadcast packets that you captured, looking at the details of the source and destination addresses. Most likely one has the broadcast Ethernet address, as broadcast frames tend to be more common than multicast frames. Look at a broadcast frame to see what address is used for broadcast by Ethernet. Expand the Ethernet address fields of either broadcast or multicast frames to see which bit is set to distinguish broadcast/multicast or group traffic from unicast traffic.

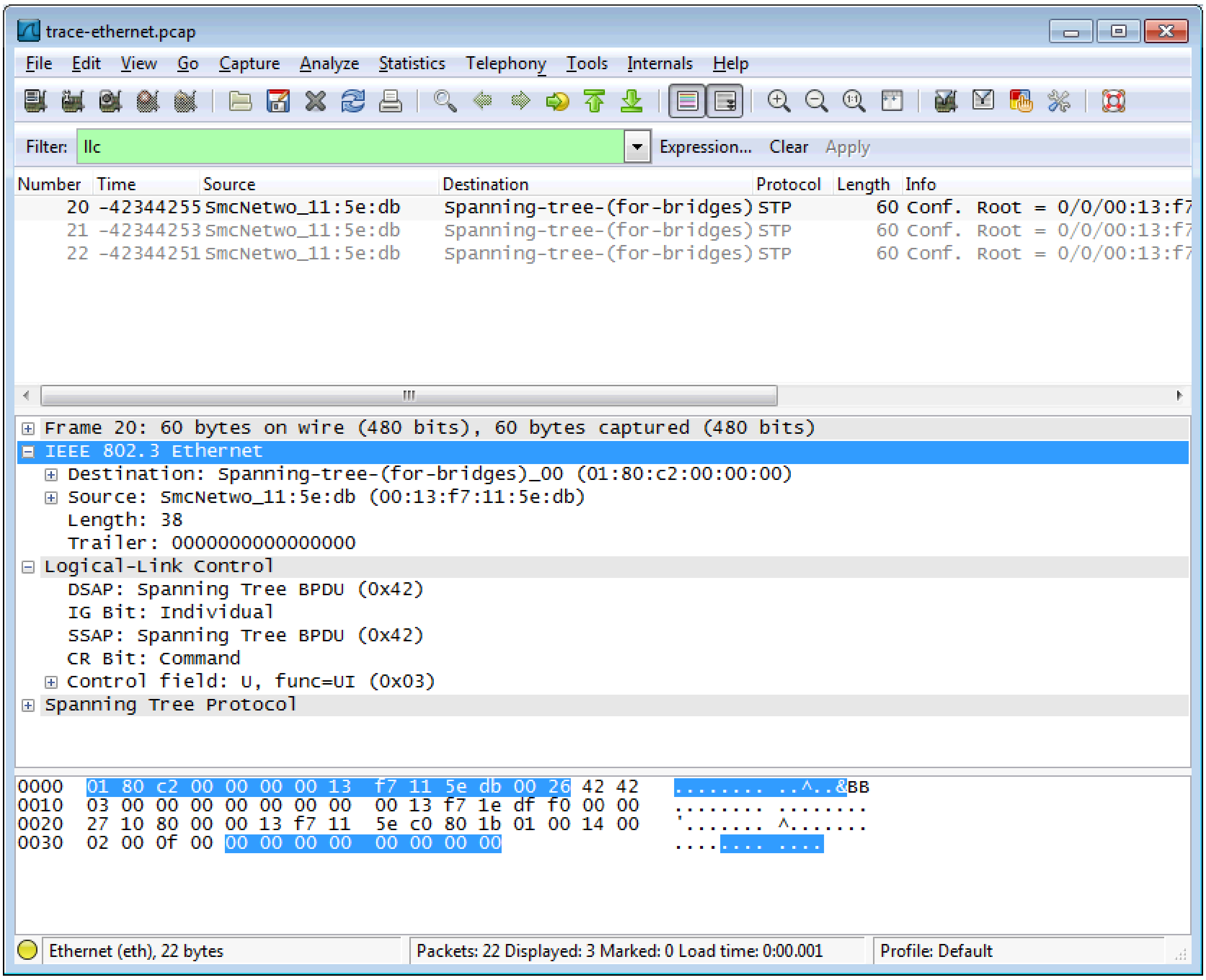
Answer the following questions:

1. *What is the broadcast Ethernet address, written in standard form as Wireshark displays it?*
2. *Which bit of the Ethernet address is used to determine whether it is unicast or multicast/broadcast?*

Turn-in: Hand in your answers to the above questions.

Explore on your own (IEEE 802.3)

We encourage you to explore Ethernet on your own once you have completed this lab. As one possibility, recall that there are two types of Ethernet frame, IEEE 802.3 and DIX Ethernet. DIX is common and what we considered above, while IEEE 802.3 is rare. If you are rather lucky, you may see some IEEE 802.3 frames in the trace you have captured. If not, then there are some of these packets in the trace that we supplied. To search for IEEE 802.3 packets, enter a display filter (above the top panel of the Wireshark window) of “llc” (that was lowercase “LLC”) because the IEEE 802.3 format has the LLC protocol on top of it. LLC is also present on top of IEEE 802.11 wireless, but it is not present on DIX Ethernet.


trace-ethernet.pcap
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: llc
Expression... Clear Apply

```
Number Time       Source              Destination                 Protocol Length Info
    20 -42344255 SmcNetwo_11:5e:db   Spanning-tree-(for-bridges) STP          60 Conf. Root = 0/0/00:13:f7
    21 -42344253 SmcNetwo_11:5e:db   Spanning-tree-(for-bridges) STP          60 Conf. Root = 0/0/00:13:f7
    22 -42344251 SmcNetwo_11:5e:db   Spanning-tree-(for-bridges) STP          60 Conf. Root = 0/0/00:13:f7
```

```
⊞ Frame 20: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
⊟ IEEE 802.3 Ethernet
  ⊞ Destination: Spanning-tree-(for-bridges)_00 (01:80:c2:00:00:00)
  ⊞ Source: SmcNetwo_11:5e:db (00:13:f7:11:5e:db)
    Length: 38
    Trailer: 0000000000000000
⊟ Logical-Link Control
    DSAP: Spanning Tree BPDU (0x42)
    IG Bit: Individual
    SSAP: Spanning Tree BPDU (0x42)
    CR Bit: Command
  ⊞ Control field: U, func=UI (0x03)
⊞ Spanning Tree Protocol
```

```
0000  01 80 c2 00 00 00 00 13  f7 11 5e db 00 26 42 42   ........ ..^..&BB
0010  03 00 00 00 00 00 00 00  00 13 f7 1e df f0 00 00   ........ ........
0020  27 10 80 00 00 13 f7 11  5e c0 80 1b 01 00 14 00   '....... ^.......
0030  02 00 0f 00 00 00 00 00  00 00 00 00               ........ ....
```

Ethernet (eth), 22 bytes | Packets: 22 Displayed: 3 Marked: 0 Load time: 0:00.001 | Profile: Default

Figure 9: IEEE 802.3 frames with Ethernet and LLC header detail

Have a look at the details of an IEEE 802.3 frame, including the LLC header. The figure shows the details for our trace. Observe that the Type field is now a Length field. In our example, the frame is short enough that there is also padding of zeros identified as a Trailer or Padding. The changes lead to a few questions for you to ponder:

1. How long are the combined IEEE 802.3 and LLC headers compared to the DIX Ethernet headers? You can use Wireshark to work this out. Note that the Trailer/Padding and Checksum may be shown as part of the header, but they come at the end of the frame.
2. How does the receiving computer know whether the frame is DIX Ethernet or IEEE 802.3? Hint: you may need to both use Wireshark to look at packet examples and read your text near where the Ethernet formats are described.
3. If IEEE 802.3 has no Type field, then how is the next higher layer determined? Use Wireshark to look for the demultiplexing key.

Turn-in: Your answers to the above questions.

[END]

实验三 IPV4

Objective

To learn about the details of IP (Internet Protocol). IP is the network layer protocol used throughout the Internet. We will examine IP version 4, since it is ubiquitously deployed, while the IP version 6 is partly deployed. IP is covered in §5.6.1 to §5.6.3 of your text. Review those sections before doing this lab.

Requirements

Wireshark: This lab uses the Wireshark software tool to capture and examine a packet trace. A packet trace is a record of traffic at a location on the network, as if a snapshot was taken of all the bits that passed across a particular wire. The packet trace records a timestamp for each packet, along with the bits that make up the packet, from the lower-layer headers to the higher-layer contents. Wireshark runs on most operating systems, including Windows, Mac and Linux. It provides a graphical UI that shows the sequence of packets and the meaning of the bits when interpreted as protocol headers and data. It color-codes packets by their type, and has various ways to filter and analyze packets to let you investigate the behavior of network protocols. Wireshark is widely used to troubleshoot networks. You can download it from www.wireshark.org if it is not already installed on your computer. We highly recommend that you watch the short, 5 minute video “Introduction to Wireshark” that is on the site.

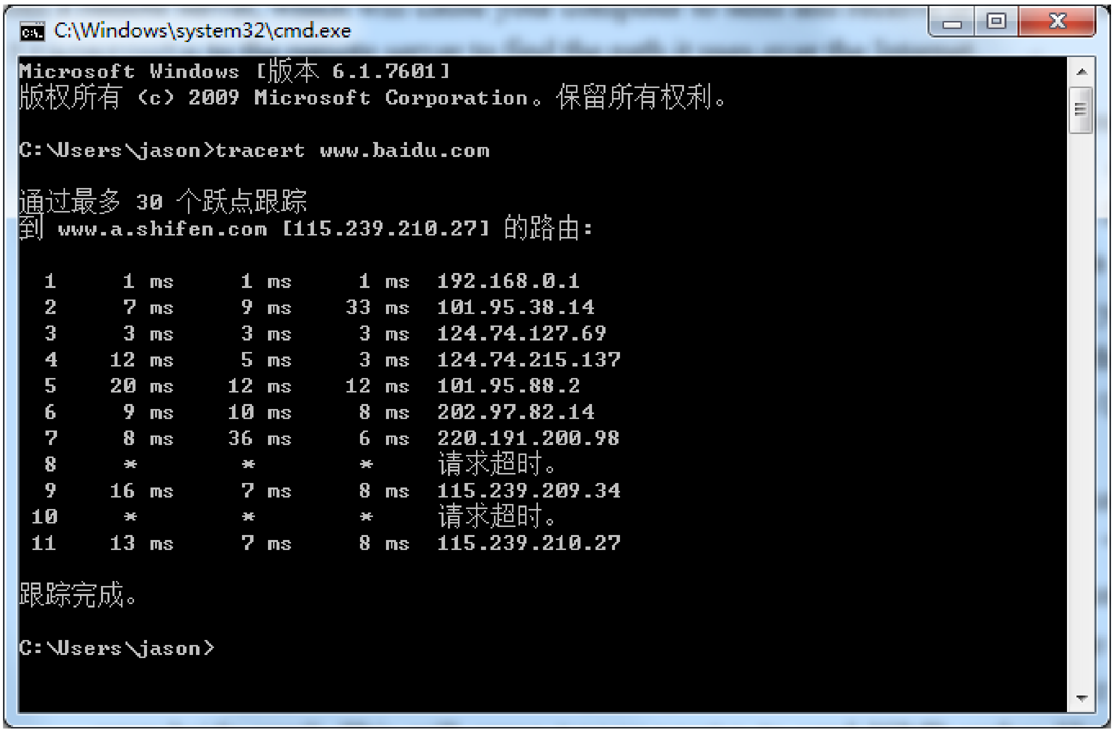
wget / curl: This lab uses `wget` (Linux and Windows) and `curl` (Mac) to fetch web resources. `wget` and `curl` are command-line programs that let you fetch a URL. Unlike a web browser, which fetches and executes entire pages, `wget` and `curl` give you control over exactly which URLs you fetch and when you fetch them. Under Linux, `wget` can be installed via your package manager. Under Windows, `wget` is available as a binary; look for download information on http://www.gnu.org/software/wget/. Under Mac, `curl` comes installed with the OS. Both have many options (try “`wget --help`” or “`curl --help`” to see) but a URL can be fetched simply with “`wget` *`URL`*” or “`curl` *`URL`*”.

traceroute / tracert: This lab uses “`traceroute`” to find the router level path from your computer to a remote Internet host. `traceroute` is a standard command-line utility for discovering the Internet paths that your computer uses. It is widely used for network troubleshooting. It comes pre-installed on Window and Mac, and can be installed using your package manager on Linux. On Windows, it is called “`tracert`”. It has various options, but simply issuing the command “`traceroute www.uwa.edu.au`” will cause your computer to find and print the path to the remote computer (here www.uwa.edu.au).

Step 1: Capture a Trace

Proceed as follows to capture a trace assuming that your computer has IPv4 connectivity; alternatively, you may use a supplied trace. The trace we want to gather is a simple web fetch from a remote server, which will cause your computer to send and receive IP packets, followed by a `traceroute` to the remote server to find the path it uses over the Internet.

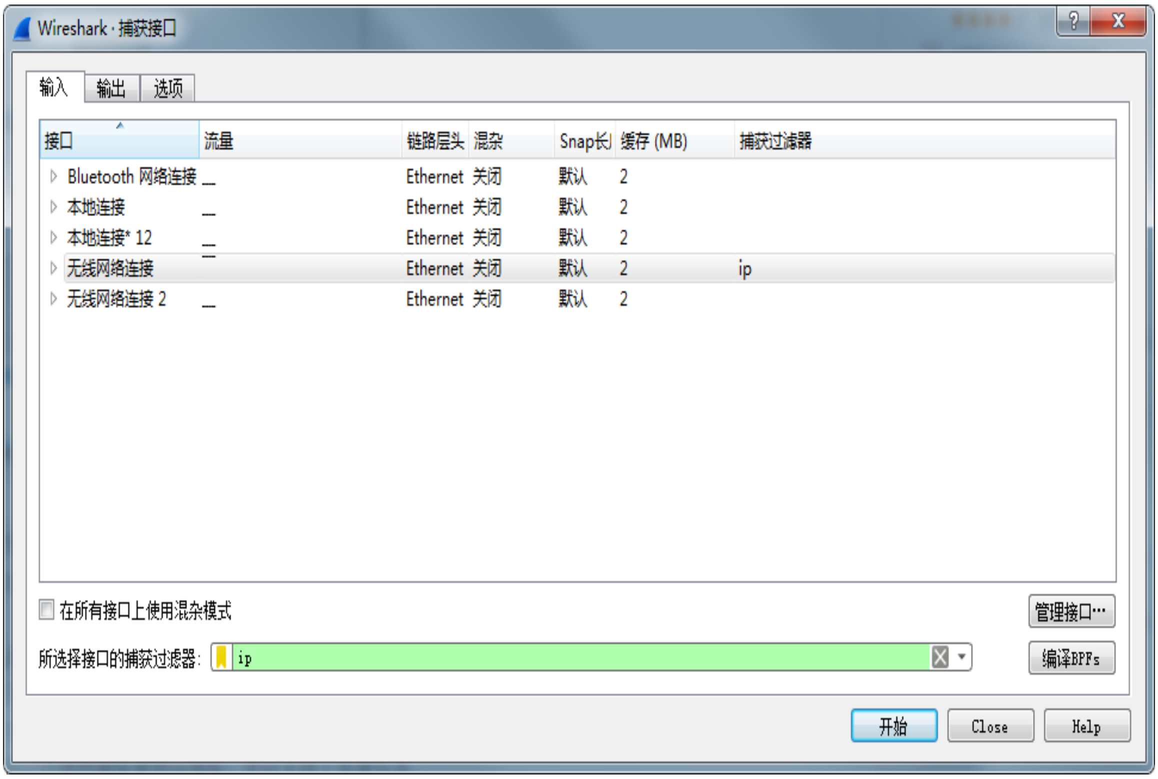
1. *Pick a URL at a remote server, e.g.,*http://www.baidu.com *and check that you can fetch the contents with* `wget` *or* `curl`*, e.g., "*`wget` http://www.baidu.com*" or "*`curl` http://www.baidu.com*".* This will fetch the resource and either write it to a file (`wget`) or to the screen (`curl`). With `wget`, you want a single response with status code "200 OK". If the fetch does not work then try a different URL; keep in mind that you may be referring to a URL by a shortcut for which browsers must do work to find the intended content, e.g., http://mit.edu may really be http://web.mit.edu/index.html. If no URLs seem to work then debug your use of `wget/curl` or your Internet connectivity.
2. *Perform a* `traceroute` *to the same remote server to check that you can discover information about the network path.* On Windows, type, e.g., "`tracert` www.baidu.com". On Linux / Mac, type, e.g., "`traceroute` www.baidu.com". If you are on Linux / Mac and behind a NAT (as most home users or virtual machine users) then use the –I option (that was a capital i) to traceroute, e.g., "`traceroute –I` www.baidu.com". This will cause `traceroute` to send ICMP probes like `tracert` instead of its usual UDP probes; ICMP probes are better able to pass through NAT boxes. A successful example is shown below; save the output as you will need it for later steps. Note that `traceroute` may take up to a minute to run. Each line shows information about the next IP hop from the computer running `traceroute` towards the target destination. The lines with "*"s indicate that there was no response from the network to identity that segment of the Internet path. Some unidentified segments are to be expected. However, if `traceroute` is not working correctly then nearly all the path will be "*"s. In this case, try a different remote server, experiment with `traceroute`, or use the supplied traces.



```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\jason>tracert www.baidu.com

通过最多 30 个跃点跟踪
到 www.a.shifen.com [115.239.210.27] 的路由:

  1     1 ms     1 ms     1 ms  192.168.0.1
  2     7 ms     9 ms    33 ms  101.95.38.14
  3     3 ms     3 ms     3 ms  124.74.127.69
  4    12 ms     5 ms     3 ms  124.74.215.137
  5    20 ms    12 ms    12 ms  101.95.88.2
  6     9 ms    10 ms     8 ms  202.97.82.14
  7     8 ms    36 ms     6 ms  220.191.200.98
  8     *        *        *     请求超时。
  9    16 ms     7 ms     8 ms  115.239.209.34
 10     *        *        *     请求超时。
 11    13 ms     7 ms     8 ms  115.239.210.27

跟踪完成。

C:\Users\jason>
```

Figure 10: Running traceroute (as `tracert` on Windows)

3. *Launch Wireshark and start a capture with a filter of* "`tcp port 80`". *Make sure to check "enable network name resolution".* We use this filter to record only standard web traffic. Name resolution will translate the IP addresses of the computers sending and receiving packets into names. It will help you to recognize whether the packets are going to or from your computer. Your capture window should be similar to the one pictured below, other than our highlighting. Select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful. Uncheck "capture packets in promiscuous mode". This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets sent to/from your computer. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive. On Wireshark 1.8, the capture filter box is present directly on the options screen, but on Wireshark 1.9, you set a capture filter by double-clicking on the interface.



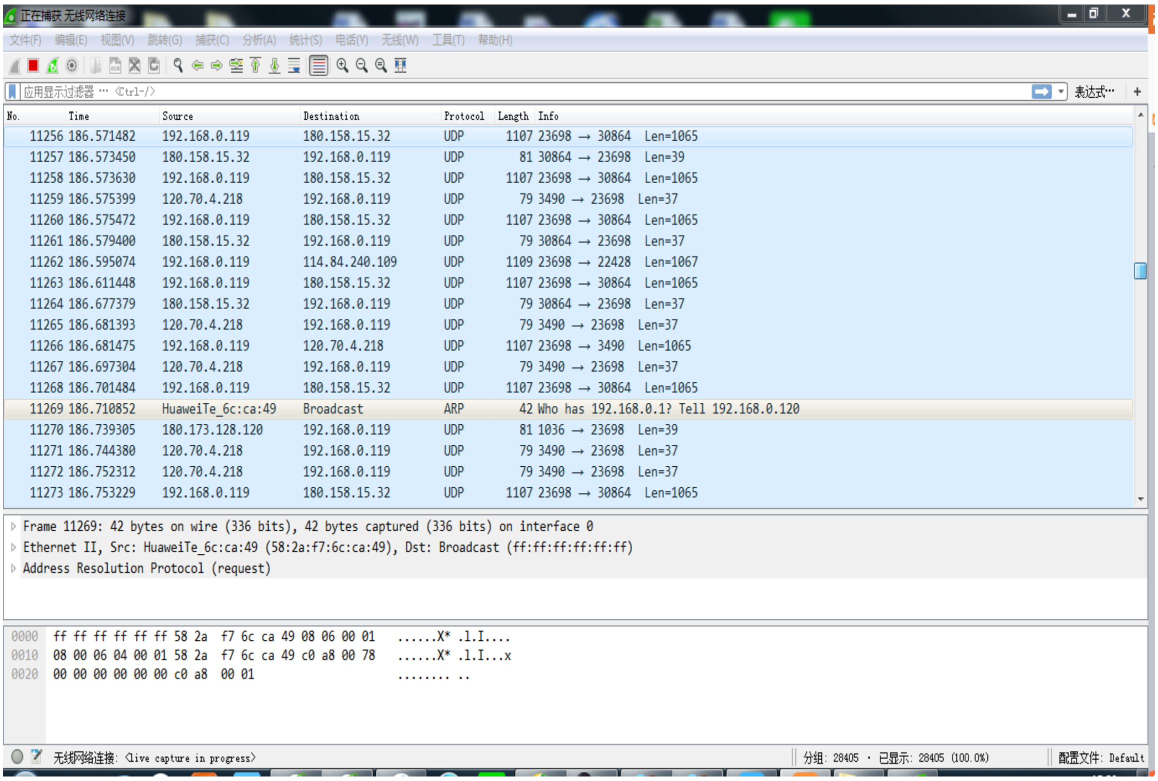
接口	流量	链路层头	混杂	Snap长	缓存 (MB)	捕获过滤器
Bluetooth 网络连接	_	Ethernet	关闭	默认	2	
本地连接	_	Ethernet	关闭	默认	2	
本地连接* 12	_	Ethernet	关闭	默认	2	
无线网络连接	_	Ethernet	关闭	默认	2	ip
无线网络连接 2	_	Ethernet	关闭	默认	2	

□ 在所有接口上使用混杂模式

所选择接口的捕获过滤器: ip

Figure 11: Setting up the capture options

4. *After the capture is started, repeat the* `wget/curl` *command above.* This time, the packets will also be recorded by Wireshark.
5. *After the command is complete, return to Wireshark and stop the trace.* You should now have a short trace similar to that shown in the figure below, along with the output of a traceroute you ran earlier to the corresponding server.


正在捕获 无线网络连接
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
应用显示过滤器 … <Ctrl-/>
表达式…
+
无线网络连接: <live capture in progress>
分组: 28405 · 已显示: 28405 (100.0%)
配置文件: Default

```
No.    Time        Source             Destination      Protocol Length Info
 11256 186.571482  192.168.0.119      180.158.15.32    UDP       1107 23698 → 30864  Len=1065
 11257 186.573450  180.158.15.32      192.168.0.119    UDP         81 30864 → 23698  Len=39
 11258 186.573630  192.168.0.119      180.158.15.32    UDP       1107 23698 → 30864  Len=1065
 11259 186.575399  120.70.4.218       192.168.0.119    UDP         79 3490 → 23698  Len=37
 11260 186.575472  192.168.0.119      180.158.15.32    UDP       1107 23698 → 30864  Len=1065
 11261 186.579400  180.158.15.32      192.168.0.119    UDP         79 30864 → 23698  Len=37
 11262 186.595074  192.168.0.119      114.84.240.109   UDP       1109 23698 → 22428  Len=1067
 11263 186.611448  192.168.0.119      180.158.15.32    UDP       1107 23698 → 30864  Len=1065
 11264 186.677379  180.158.15.32      192.168.0.119    UDP         79 30864 → 23698  Len=37
 11265 186.681393  120.70.4.218       192.168.0.119    UDP         79 3490 → 23698  Len=37
 11266 186.681475  192.168.0.119      120.70.4.218     UDP       1107 23698 → 3490  Len=1065
 11267 186.697304  120.70.4.218       192.168.0.119    UDP         79 3490 → 23698  Len=37
 11268 186.701484  192.168.0.119      180.158.15.32    UDP       1107 23698 → 30864  Len=1065
 11269 186.710852  HuaweiTe_6c:ca:49  Broadcast        ARP         42 Who has 192.168.0.1? Tell 192.168.0.120
 11270 186.739305  180.173.128.120    192.168.0.119    UDP         81 1036 → 23698  Len=39
 11271 186.744380  120.70.4.218       192.168.0.119    UDP         79 3490 → 23698  Len=37
 11272 186.752312  120.70.4.218       192.168.0.119    UDP         79 3490 → 23698  Len=37
 11273 186.753229  192.168.0.119      180.158.15.32    UDP       1107 23698 → 30864  Len=1065
```

```
▷ Frame 11269: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▷ Ethernet II, Src: HuaweiTe_6c:ca:49 (58:2a:f7:6c:ca:49), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▷ Address Resolution Protocol (request)
```

```
0000  ff ff ff ff ff ff 58 2a  f7 6c ca 49 08 06 00 01   ......X* .l.I....
0010  08 00 06 04 00 01 58 2a  f7 6c ca 49 c0 a8 00 78   ......X* .l.I...x
0020  00 00 00 00 00 00 c0 a8  00 01                     ........ ..
```

Figure 12: Trace of `wget`/`curl` traffic showing the details of the IP header

Step 2: Inspect the Trace

Select any packet in the trace and expand the IP header fields (using the "+" expander or icon) to see the details. You can simply click on a packet to select it (in the top panel). You will see details of its structure (in the middle panel) and the bytes that make up the packet (in the bottom panel). Our interest is the IP header, and you may ignore the other higher and lower layer protocols. When you click on parts of the IP header, you will see the bytes that correspond to the part highlighted in the bottom panel. We have expanded the IP header and clicked on all the IP header fields in the figure above.

Let us go over the fields in turn:

- The version field is set to 4. This is "IPv4" after all.
- Then there is the header length field. Observe by looking at the bytes selected in the packet data that version and header length are both packed into a single byte.
- The Differentiated Services field contains bit flags to indicate whether the packet should be handled with quality of service and congestion indications at routers.
- Then there is the Total Length field.
- Next is the Identification field, which is used for grouping fragments, when a large IP packet is sent as multiple smaller pieces called fragments. It is followed by the Flags and the Fragment offset fields, which also relate to fragmentation. Observe they share bytes.
- Then there is the Time to live or TTL field, followed by the Protocol field.
- Next comes the header checksum. Is your header checksum carrying 0 and flagged as

incorrect for IP packets sent from your computer to the remote server? On some computers, the operating system software leaves the header checksum blank (zero) for the NIC to compute and fill in as the packet is sent. This is called protocol offloading. It happens after Wireshark sees the packet, which causes Wireshark to believe that the checksum is wrong and flag it with a different color to signal a problem. A similar issue may happen for the TCP checksum. You can remove these false errors if they are occurring by telling Wireshark not to validate the checksums. Select "Preferences" from the Wireshark menus and expand the "Protocols" area. Look under the list until you come to IPv4. Uncheck "Validate checksum if possible". Similarly, you may uncheck checksum validation for TCP if applicable to your case.

- The last fields in the header are the normally the source and destination address. It is possible for there to be IP options, but these are unlikely in standard web traffic.
- The IP header is followed by the IP payload. This makes up the rest of the packet, starting with the next higher layer header, TCP in our case, but not including any link layer trailer (e.g., Ethernet padding).

Step 3: IP Packet Structure

To show your understanding of IP, sketch a figure of an IP packet you studied. It should show the position and size in bytes of the IP header fields as you can observe using Wireshark. Since you cannot easily determine sub-byte sizes, group any IP fields that are packed into the same bytes. Your figure can simply show the frame as a long, thin rectangle. Try not to look at the figure of an IPv4 packet in your text; check it afterwards to note and investigate any differences.

To work out sizes, observe that when you click on a protocol block in the middle panel (the block itself, not the "+" expander) Wireshark will highlight the corresponding bytes in the packet in the lower panel, and display the length at the bottom of the window. You may also use the overall packet size shown in the Length column or Frame detail block. Note that this method will not tell you sub-byte positions.

By looking at the IP packets in your trace, answer these questions:

1. *What are the IP addresses of your computer and the remote server?*
2. *Does the Total Length field include the IP header plus IP payload, or just the IP payload?*
3. *How does the value of the Identification field change or stay the same for different packets?* For instance, does it hold the same value for all packets in a TCP connection or does it differ for each packet? Is it the same in both directions? Can you see any pattern if the value does change?
4. *What is the initial value of the TTL field for packets sent from your computer? Is it the maximum possible value, or some lower value?*
5. *How can you tell from looking at a packet that it has not been fragmented?* Most often IP packets in normal operation are not fragmented. But the receiver must have a way to be sure. Hint: you may need to read your text to confirm a guess.
6. *What is the length of the IP Header and how is this encoded in the header length field?* Hint: notice that only 4 bits are used for this field, as the version takes up the other 4 bits

of the byte. You may guess and check your text.

Turn-in: Hand in your drawing of an IP packet and the answers to the questions above.

Step 4: Internet Paths

The source and destination IP addresses in an IP packet denote the endpoints of an Internet path, not the IP routers on the network path the packet travels from the source to the destination. `traceroute` is a utility for discovering this path. It works by eliciting responses (ICMP TTL Exceeded messages) from the router 1 hop away from the source towards the destination, then 2 hops away from the source, then 3 hops, and so forth until the destination is reached. The responses will identify the IP address of the router. The output from `traceroute` normally prints the information for one hop per line, including the measured round trip times and IP address and DNS names of the router. The DNS name is handy for working out the organization to which the router belongs. Since `traceroute` takes advantage of common router implementations, there is no guarantee that it will work for all routers along the path, and it is usual to see "*" responses when it fails for some portions of the path.

Using the `traceroute` *output, sketch a drawing of the network path. If you are using the supplied trace, note that we have provided the corresponding* `traceroute` *output as a separate file. Show your computer (lefthand side) and the remote server (righthand side), both with IP addresses, as well as the routers along the path between them numbered by their distance on hops from the start of the path.* You can find the IP address of your computer and the remote server on the packets in the trace that you captured. The output of `traceroute` will tell you the hop number for each router.

To finish your drawing, label the routers along the path with the name of the real-world organization to which they belong. To do this, you will need to interpret the domain names of the routers given by `traceroute`*. If you are unsure, label the routers with the domain name of what you take to be the organization. Ignore or leave blank any routers for which there is no domain name (or no IP address).*

This is not an exact science, so we will give some examples. Suppose that `traceroute` identifies a router along the path by the domain name `arouter.cac.washington.edu`. Normally, we can ignore at least the first part of the name, since it identifies different computers in the same organization and not different organizations. Thus we can ignore at least "`arouter`" in the domain name. For generic top-level domains, like "`.com`" and "`.edu`", the last two domains give the domain name of the organization. So for our example, it is "`washington.edu`". To translate this domain name into the real-world name of an organization, we might search for it on the web. You will quickly find that `washington.edu` is the University of Washington. This means that "`cac`" portion is an internal structure in the University of Washington, and not important for the organization name. You would write "University of Washington" on your figure for any routers with domain names of the form `*.washington.edu`.

Alternatively, consider a router with a domain name like `arouter.syd.aarnet.net.au`. Again, we ignore at least the "`arouter`" part as indicating a computer within a specific organization. For country-code top-level domains like "`.au`" (for Australia) the last three domains in the name will normally give the organization. In this case the organization's domain name is `aarnet.net.au`. Using a web search, we find this domain represents AARNET, Australia's research and education network. The "`syd`" portion is internal structure, and a good guess is that it means the router is located in the Sydney part of AARNET. So for all routers with domain names of the form `*.aarnet.net.au`, you would write "AARNET" on your figure. While there are no guarantees, you should be able to reason similarly and at least give the domain name of the organizations near the ends of the path.

Turn-in: Hand in your drawing, and `traceroute` output if it was not supplied to you.

Step 5: IP Header Checksum

We will now look at the IP header checksum calculation by validating a packet. The checksum algorithm adds the header bytes 16 bits at a time. It is computed so that re-computing the sum across the entire IP header (including the checksum value) will produce the result of zero. A complicating factor for us is that this is done using 1s complement arithmetic, rather than 2s complement arithmetic that is normally used for computing. The steps below explain how to perform the necessary computation.

From the trace, pick a packet sent from the remote server to your computer and check that you have a non-zero value in the checksum field. The checksum value sent over the network will be non-zero, so if you have a zero value it is because of the capture setup. Try a packet that has an IP header of 20 bytes, the minimum header size when there are no options, to make this exercise easier.

Follow these steps to check that the checksum value is correct:

1. Divide the header into 10 two byte (16 bit) words. Each word will be 4 hexadecimal digits shown in the packet data panel in the bottom of the Wireshark window, e.g., `05 8c`
2. Add these 10 words using regular addition. You may add them with a hexadecimal calculator (Google to find one), or convert them to decimal, add them, and convert them back to hexadecimal. Do whatever is easiest.
3. To compute the 1s complement sum from your addition so far, take any leading digits (beyond the 4 digits of the word size) and add them back to the remainder. For example: `5a432` will become `a432 + 5 = a437.`
4. The end result should be `0xffff`. This is actually zero in 1s complement form, or more precisely `0xffff` is -0 (negative zero) while `0x0000` is +0 (positive zero).

If you cannot get your sum to come out and are sure that the checksum must be wrong, you can get Wireshark to check it. See whether it says "[correct]" already. If it does not then use the menus to go to Preferences, expand Protocols, choose IPv4 from the list, and check "validate header checksum". Now Wireshark will check the checksum and tell you if it is correct.

Turn-in: Hand in your sums. Next to each word you add note the IPv4 fields to which it corresponds.

Explore on your own

We encourage you to explore IP on your own once you have completed this lab. Some ideas:

- Read about and experiment with IPv6. Modern operating systems already include support for IPv6, so you may be able to capture IPv6 traffic on your network. You can also "join the IPv6" backbone by tunneling to an IPv6 provider.
- Learn about tunnels, which wrap an IP packet within another IP header.
- Read about IP geolocation. It is the process of assigning a geographical location to an IP address using measurements or clues from its name administrative databases. Try a geolocation service.
- Learn about IPsec or IP security. It provides confidentiality and authentication for IP packets, and is often used as part of VPNs.

实验四 ARP

Objective

To see how ARP (Address Resolution Protocol) works. ARP is an essential glue protocol that is used to join Ethernet and IP. It is covered in §5.6.4 of your text. Review the text section before doing this lab.

Requirements

Wireshark: This lab uses the Wireshark software tool to capture and examine a packet trace. A packet trace is a record of traffic at a location on the network, as if a snapshot was taken of all the bits that passed across a particular wire. The packet trace records a timestamp for each packet, along with the bits that make up the packet, from the lower-layer headers to the higher-layer contents. Wireshark runs on most operating systems, including Windows, Mac and Linux. It provides a graphical UI that shows the sequence of packets and the meaning of the bits when interpreted as protocol headers and data. It color-codes packets by their type, and has various ways to filter and analyze packets to let you investigate the behavior of network protocols. Wireshark is widely used to troubleshoot networks. You can download it from www.wireshark.org if it is not already installed on your computer. We highly recommend that you watch the short, 5 minute video "Introduction to Wireshark" that is on the site.

arp: This lab uses the "`arp`" command-line utility to inspect and clear the cache used by the ARP protocol on your computer. `arp` is installed as part of the operating system on Windows, Linux, and Mac computers, but uses different arguments. It requires administrator privileges to clear the cache.

ifconfig / ipconfig: This lab uses the "`ipconfig`" (Windows) or "`ifconfig`" (Mac/Linux) command-line utility to inspect the state of your computer's network interface. `ifconfig/ipconfig` is installed as part of the operating system on Windows, Linux, and Mac computers.

route / netstat: This lab uses the "`route`" or "`netstat`" command-line utility to inspect the routes used by your computer. A key route is the default route (or route to prefix 0.0.0.0) that uses the default gateway to reach remote parts of the Internet. Both "`route`" and "`netstat`" are installed as part of the operating system across Windows and Mac/Linux, but there are many variations on the command-line parameters that must be used.

Browser: This lab uses a web browser to find or fetch pages as a workload. Any web browser will do.

Network Setup

We want to observe the ARP protocol in action. Recall that ARP is used to find the Ethernet address that corresponds to a local IP address to which your computer wants to send a packet. A typical example of a local IP address is that of the local router or default gateway that connects your computer to the rest of the Internet. Your computer caches these translations in an ARP cache so that the ARP protocol need only be used occasionally to do the translation. The setup from the viewpoint of your computer is as shown in the example below.

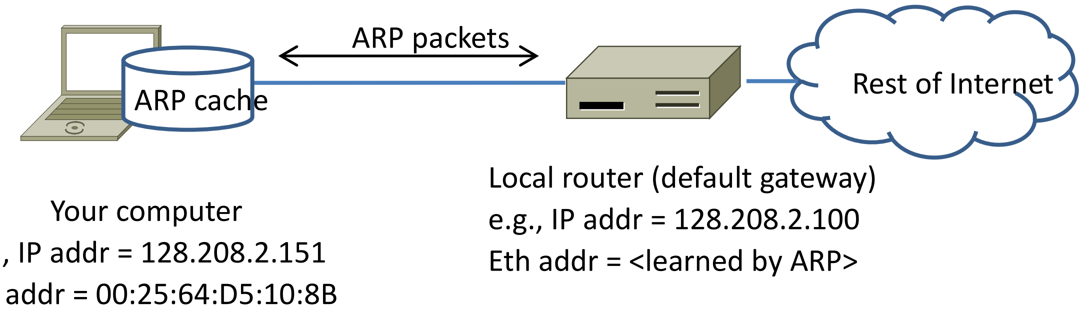
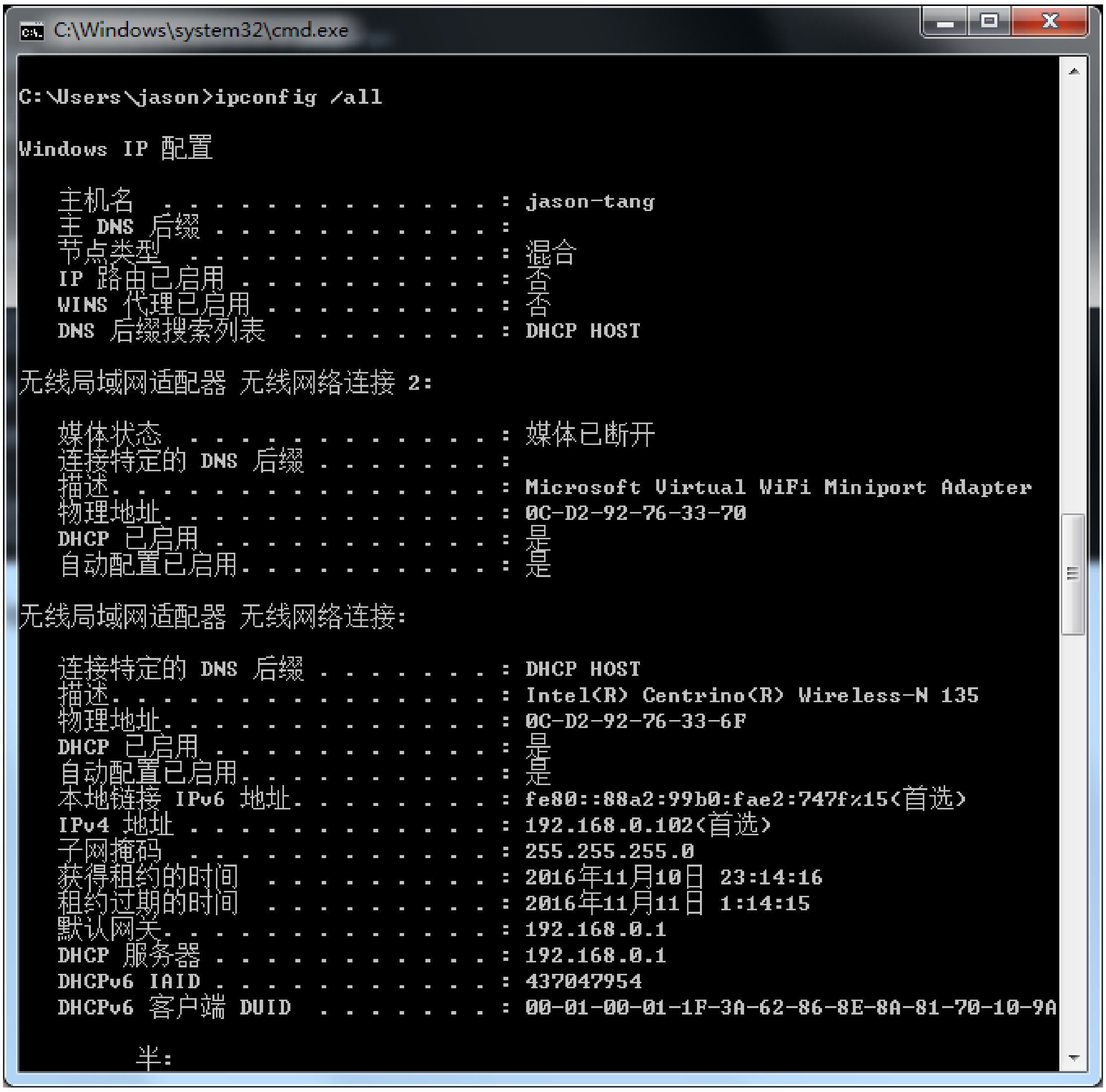

ARP packets
ARP cache
Rest of Internet
Local router (default gateway)
e.g., IP addr = 128.208.2.100
Eth addr = <learned by ARP>
Your computer
e.g., IP addr = 128.208.2.151
Eth addr = 00:25:64:D5:10:8B

Figure 13: Network setup under which we will study ARP

Step 1: Capture a Trace

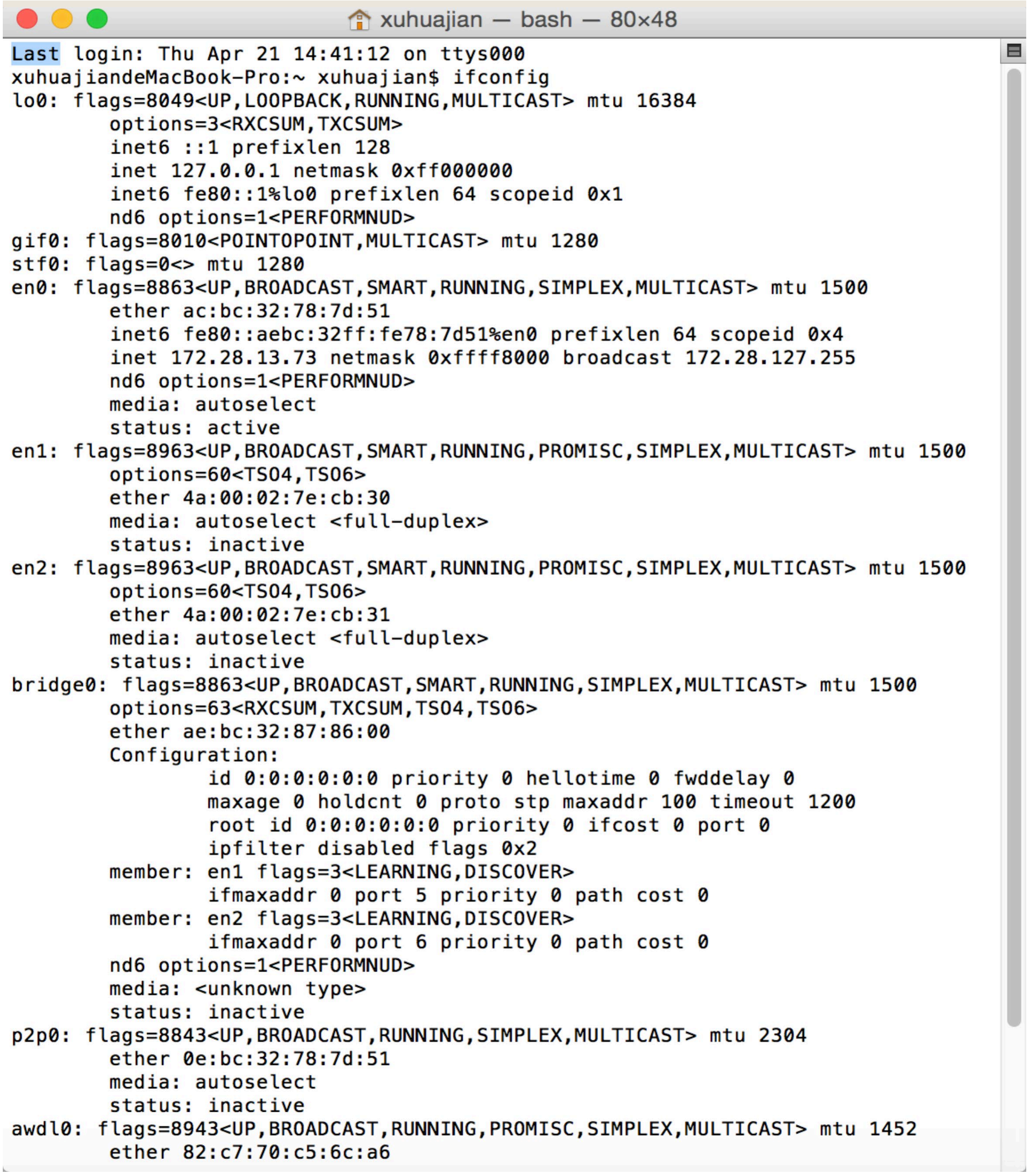
Proceed as follows to capture a trace of ARP traffic; alternatively, you may use a supplied trace. To gather ARP packets, we will cause your computer to send traffic to the local router when it does not know the router's Ethernet address – your computer will then use ARP to discover the Ethernet address.

5. *Find the Ethernet address of the main network interface of your computer with the* `ifconfig` / `ipconfig` *command*. You will want to know this address for later analysis. On Windows, bring up a command-line shell and type "`ipconfig /all`". On Mac/Linux, bring up a command-line shell and type "`ifconfig`". Among the output will be a section for the main interface of the computer (likely an Ethernet interface) and its Ethernet address. Common names for the interface are "eth0", "en0", or "Ethernet adapter". Two examples are shown below, with our added highlighting.



```
C:\Users\jason>ipconfig /all

Windows IP 配置

   主机名  . . . . . . . . . . . . . : jason-tang
   主 DNS 后缀 . . . . . . . . . . . :
   节点类型  . . . . . . . . . . . . : 混合
   IP 路由已启用 . . . . . . . . . . : 否
   WINS 代理已启用 . . . . . . . . . : 否
   DNS 后缀搜索列表  . . . . . . . . : DHCP HOST

无线局域网适配器 无线网络连接 2:

   媒体状态  . . . . . . . . . . . . : 媒体已断开
   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : Microsoft Virtual WiFi Miniport Adapter
   物理地址. . . . . . . . . . . . . : 0C-D2-92-76-33-70
   DHCP 已启用 . . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . . : 是

无线局域网适配器 无线网络连接:

   连接特定的 DNS 后缀 . . . . . . . : DHCP HOST
   描述. . . . . . . . . . . . . . . : Intel(R) Centrino(R) Wireless-N 135
   物理地址. . . . . . . . . . . . . : 0C-D2-92-76-33-6F
   DHCP 已启用 . . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . . : 是
   本地链接 IPv6 地址. . . . . . . . : fe80::88a2:99b0:fae2:747f%15(首选)
   IPv4 地址 . . . . . . . . . . . . : 192.168.0.102(首选)
   子网掩码  . . . . . . . . . . . . : 255.255.255.0
   获得租约的时间  . . . . . . . . . : 2016年11月10日 23:14:16
   租约过期的时间  . . . . . . . . . : 2016年11月11日 1:14:15
   默认网关. . . . . . . . . . . . . : 192.168.0.1
   DHCP 服务器 . . . . . . . . . . . : 192.168.0.1
   DHCPv6 IAID . . . . . . . . . . . : 437047954
   DHCPv6 客户端 DUID  . . . . . . . : 00-01-00-01-1F-3A-62-86-8E-8A-81-70-10-9A

         半:
```

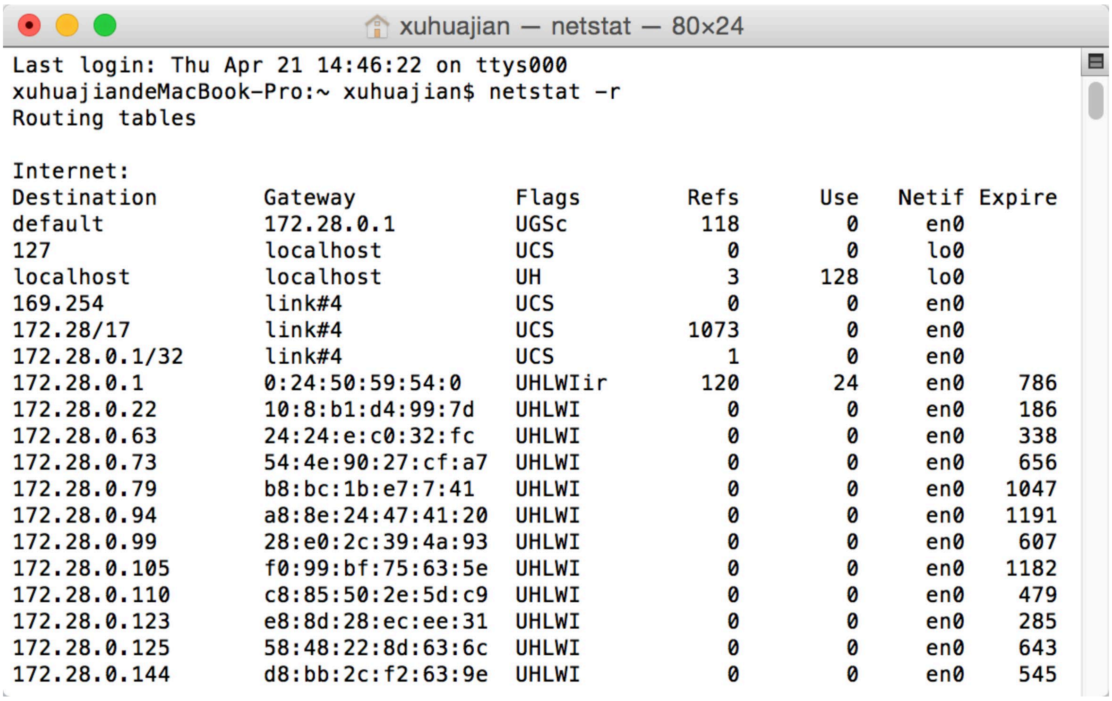
Figure 14: Finding the computer's Ethernet address with `ipconfig` (Windows)



```
Last login: Thu Apr 21 14:41:12 on ttys000
xuhuajiandeMacBook-Pro:~ xuhuajian$ ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        options=3<RXCSUM,TXCSUM>
        inet6 ::1 prefixlen 128
        inet 127.0.0.1 netmask 0xff000000
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
        nd6 options=1<PERFORMNUD>
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether ac:bc:32:78:7d:51
        inet6 fe80::aebc:32ff:fe78:7d51%en0 prefixlen 64 scopeid 0x4
        inet 172.28.13.73 netmask 0xffff8000 broadcast 172.28.127.255
        nd6 options=1<PERFORMNUD>
        media: autoselect
        status: active
en1: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        options=60<TSO4,TSO6>
        ether 4a:00:02:7e:cb:30
        media: autoselect <full-duplex>
        status: inactive
en2: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        options=60<TSO4,TSO6>
        ether 4a:00:02:7e:cb:31
        media: autoselect <full-duplex>
        status: inactive
bridge0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=63<RXCSUM,TXCSUM,TSO4,TSO6>
        ether ae:bc:32:87:86:00
        Configuration:
                id 0:0:0:0:0:0 priority 0 hellotime 0 fwddelay 0
                maxage 0 holdcnt 0 proto stp maxaddr 100 timeout 1200
                root id 0:0:0:0:0:0 priority 0 ifcost 0 port 0
                ipfilter disabled flags 0x2
        member: en1 flags=3<LEARNING,DISCOVER>
                ifmaxaddr 0 port 5 priority 0 path cost 0
        member: en2 flags=3<LEARNING,DISCOVER>
                ifmaxaddr 0 port 6 priority 0 path cost 0
        nd6 options=1<PERFORMNUD>
        media: <unknown type>
        status: inactive
p2p0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 2304
        ether 0e:bc:32:78:7d:51
        media: autoselect
        status: inactive
awdl0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1452
        ether 82:c7:70:c5:6c:a6
```

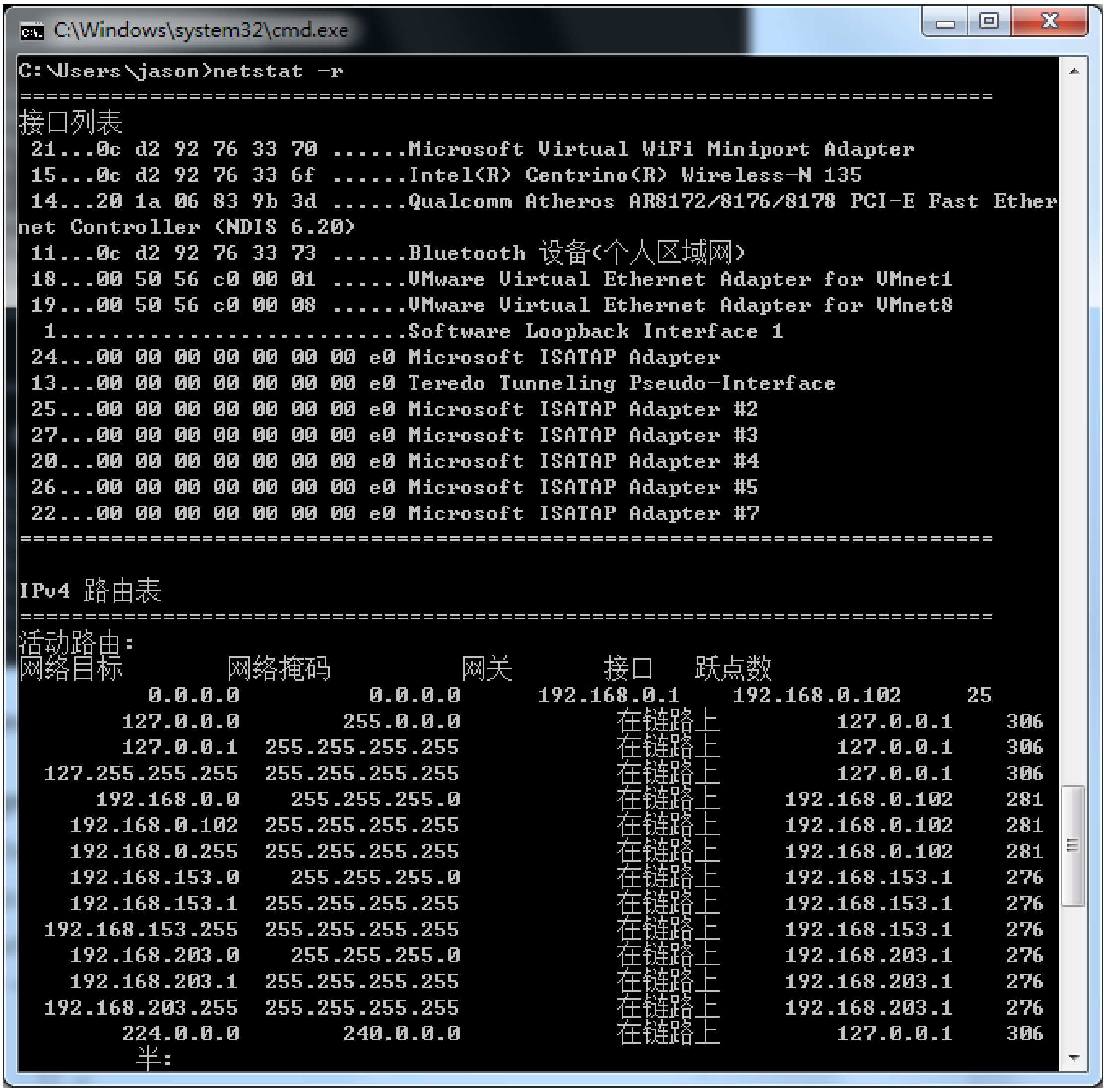
Figure 15: Finding the computer’s Ethernet address with `ifconfig` (Mac)

6. *Find the IP address of the local router or default gateway that your computer uses to reach the rest of the Internet using the* `netstat` */* `route` *command.* You should be able to use the `netstat` command (“`netstat -r`” on Windows, Mac and Linux, may require ctrl-C to stop). Alternatively, you can use the route command (“`route print`” on Windows, “`route`” on Linux, “`route -n get default`” on Mac). In either case you are looking for the gateway IP address that corresponds to the destination of default or 0.0.0.0. Two examples are shown below for `netstat`, with our added highlighting.



```
Last login: Thu Apr 21 14:46:22 on ttys000
xuhuajiandeMacBook-Pro:~ xuhuajian$ netstat -r
Routing tables

Internet:
Destination        Gateway            Flags        Refs      Use   Netif Expire
default            172.28.0.1         UGSc          118        0     en0
127                localhost          UCS             0        0     lo0
localhost          localhost          UH              3      128     lo0
169.254            link#4             UCS             0        0     en0
172.28/17          link#4             UCS          1073        0     en0
172.28.0.1/32      link#4             UCS             1        0     en0
172.28.0.1         0:24:50:59:54:0    UHLWIir       120       24     en0    786
172.28.0.22        10:8:b1:d4:99:7d   UHLWI           0        0     en0    186
172.28.0.63        24:24:e:c0:32:fc   UHLWI           0        0     en0    338
172.28.0.73        54:4e:90:27:cf:a7  UHLWI           0        0     en0    656
172.28.0.79        b8:bc:1b:e7:7:41   UHLWI           0        0     en0   1047
172.28.0.94        a8:8e:24:47:41:20  UHLWI           0        0     en0   1191
172.28.0.99        28:e0:2c:39:4a:93  UHLWI           0        0     en0    607
172.28.0.105       f0:99:bf:75:63:5e  UHLWI           0        0     en0   1182
172.28.0.110       c8:85:50:2e:5d:c9  UHLWI           0        0     en0    479
172.28.0.123       e8:8d:28:ec:ee:31  UHLWI           0        0     en0    285
172.28.0.125       58:48:22:8d:63:6c  UHLWI           0        0     en0    643
172.28.0.144       d8:bb:2c:f2:63:9e  UHLWI           0        0     en0    545
```

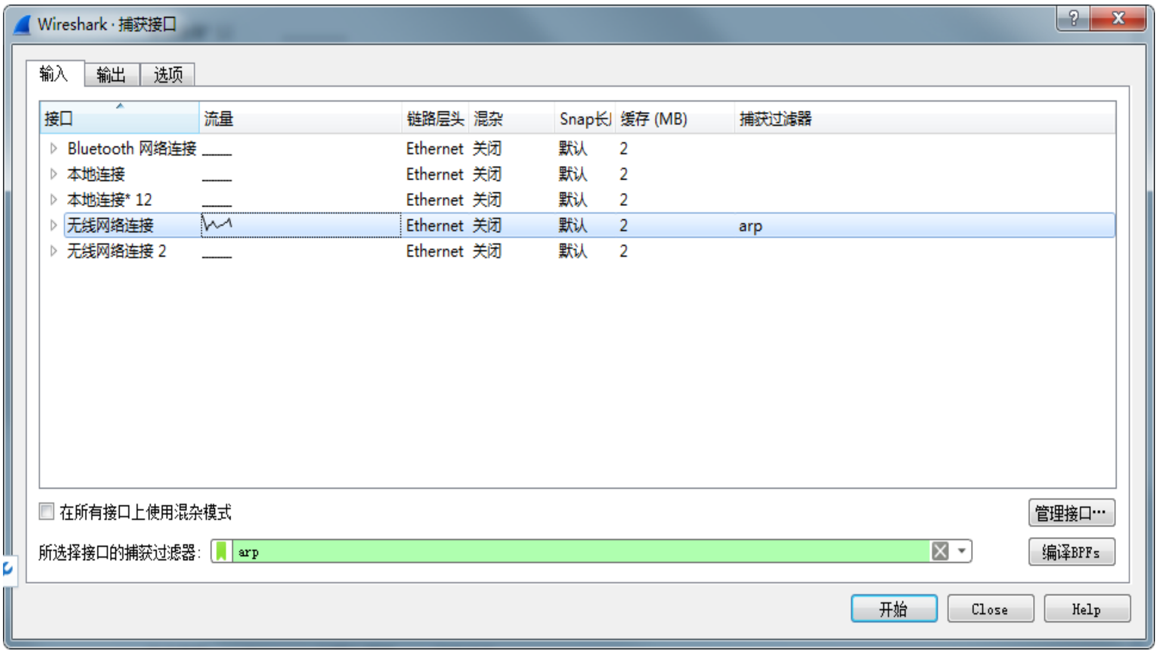
Figure 4: Finding the default gateway IP address with `netstat` (Mac)



```
C:\Users\jason>netstat -r
===========================================================================
接口列表
 21...0c d2 92 76 33 70 ......Microsoft Virtual WiFi Miniport Adapter
 15...0c d2 92 76 33 6f ......Intel(R) Centrino(R) Wireless-N 135
 14...20 1a 06 83 9b 3d ......Qualcomm Atheros AR8172/8176/8178 PCI-E Fast Ether
net Controller (NDIS 6.20)
 11...0c d2 92 76 33 73 ......Bluetooth 设备(个人区域网)
 18...00 50 56 c0 00 01 ......VMware Virtual Ethernet Adapter for VMnet1
 19...00 50 56 c0 00 08 ......VMware Virtual Ethernet Adapter for VMnet8
  1...........................Software Loopback Interface 1
 24...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter
 13...00 00 00 00 00 00 00 e0 Teredo Tunneling Pseudo-Interface
 25...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter #2
 27...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter #3
 20...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter #4
 26...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter #5
 22...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter #7
===========================================================================

IPv4 路由表
===========================================================================
活动路由:
网络目标        网络掩码          网关       接口   跃点数
          0.0.0.0          0.0.0.0      192.168.0.1    192.168.0.102     25
        127.0.0.0        255.0.0.0            在链路上         127.0.0.1    306
        127.0.0.1  255.255.255.255            在链路上         127.0.0.1    306
  127.255.255.255  255.255.255.255            在链路上         127.0.0.1    306
      192.168.0.0    255.255.255.0            在链路上     192.168.0.102    281
    192.168.0.102  255.255.255.255            在链路上     192.168.0.102    281
    192.168.0.255  255.255.255.255            在链路上     192.168.0.102    281
    192.168.153.0    255.255.255.0            在链路上     192.168.153.1    276
    192.168.153.1  255.255.255.255            在链路上     192.168.153.1    276
  192.168.153.255  255.255.255.255            在链路上     192.168.153.1    276
    192.168.203.0    255.255.255.0            在链路上     192.168.203.1    276
    192.168.203.1  255.255.255.255            在链路上     192.168.203.1    276
  192.168.203.255  255.255.255.255            在链路上     192.168.203.1    276
        224.0.0.0        240.0.0.0            在链路上         127.0.0.1    306
         半:
```

Figure 5: Finding the default gateway IP address with `netstat` (Windows)

7. *Launch Wireshark and start a capture with a filter of* “`arp`”. Your capture window should be similar to the one pictured below, other than our highlighting. Select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful. Uncheck “capture packets in promiscuous mode”. This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets sent to/from your computer. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive. On Wireshark 1.8, the capture filter box is present directly on the options screen, but on Wireshark 1.9, you set a capture filter by double-clicking on the interface.



接口	流量	链路层头	混杂	Snap长	缓存 (MB)	捕获过滤器
Bluetooth 网络连接		Ethernet	关闭	默认	2	
本地连接		Ethernet	关闭	默认	2	
本地连接* 12		Ethernet	关闭	默认	2	
无线网络连接		Ethernet	关闭	默认	2	arp
无线网络连接 2		Ethernet	关闭	默认	2	

☐ 在所有接口上使用混杂模式

所选择接口的捕获过滤器: arp

Figure 16: Setting up the capture options

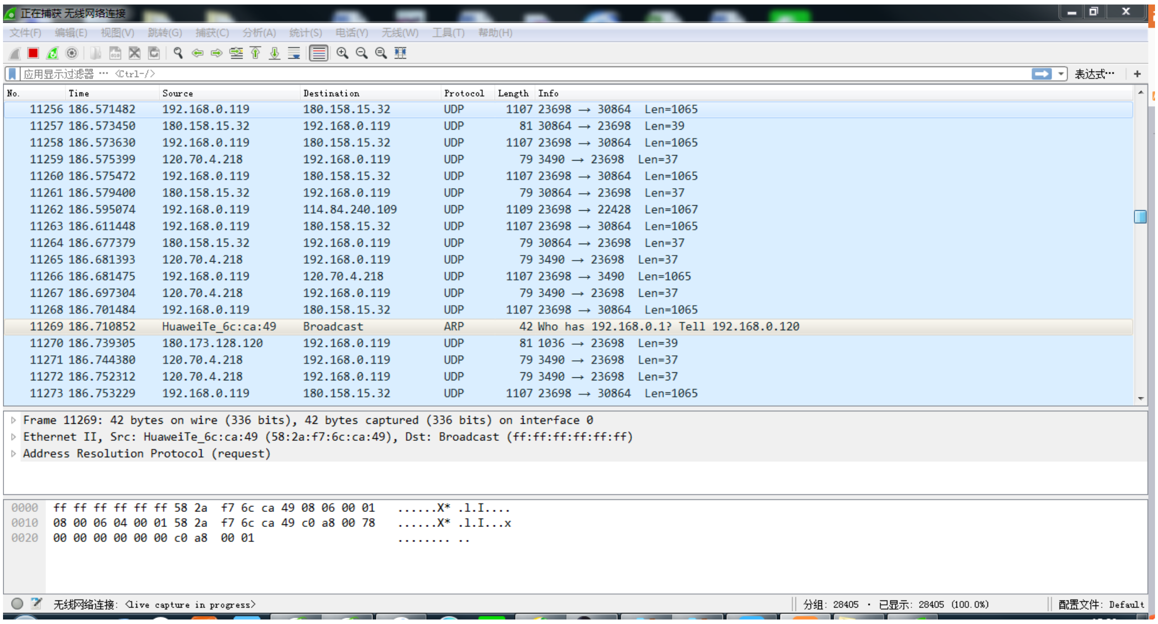
8. *When the capture is started, use the "*`arp`*" command to clear the default gateway from the ARP cache.* Using the command "`arp -a`" will show you the contents of the ARP cache as a check that you can run "`arp`". You should see an entry for the IP address of the default gateway. To clear this entry, use the arp command with different arguments ("`arp -d`" on Windows, "`arp -d -a`" on Mac, "`arp -d xx.xx.xx.xx`" where xx.xx.xx.xx is the IP address of the default gateway on Linux). This usage of `arp` will need administrator privileges to run, so you may run as a privileged user on Windows or issue "`sudo arp -d xx.xx.xx.xx`" on Linux/Mac. Note that the command should run without error but the ARP entry may not appear to be cleared if you check with "`arp -a`". This is because your computer will send ARP packets to repopulate this entry as soon as you need to send a packet to a remote IP address, and that can happen very quickly due to background activity on the computer.
9. *Now that you have cleared your ARP cache, fetch a remote page with your Web browser.* This will cause ARP to find the Ethernet address of the default gateway so that the packets can be sent. These ARP packets will be captured by Wireshark. You might clear the ARP cache and fetch a document a couple of times. Hopefully there will also be other ARP packets sent by other computers on the local network that will be captured. These packets are likely to be present if there are other computers on your local network. In fact, if you have a busy computer and extensive local network then you may capture many ARP packets. The ARP traffic of other computers will be captured when the ARP packets are sent to the broadcast address, since in this case they are destined for all computers including the one on which you are running Wireshark. Because ARP activity happens slowly, you may need to wait up to 30 seconds to observe some of this background ARP traffic.
10. *Once you have captured some ARP traffic, stop the capture.* You will need the trace, plus the Ethernet address of your computer and the IP address of the default gateway for the

next steps.

Step 2: Inspect the Trace

Now we can look at an ARP exchange! Since there may be many ARP packets in your trace, we'll first narrow our view to only the ARP packets that are sent directly from or to your computer.

Set a display filter for packets with the Ethernet address of your computer. You can do this by entering an expression in the blank "Filter:" box near the top of the Wireshark window and clicking "Apply". The filter to enter depends on your Ethernet address. For example, if your Ethernet address is 01:02:03:04:05:06 then enter a filter expression of "`eth.addr==01:02:03:04:05:06`". Note the double equal sign. If you are using the supplied trace, it comes with an additional text file giving the Ethernet address and default gateway IP address. After applying this filter your capture should look something like the figure below, in which we have expanded the ARP protocol details.


正在捕获 无线网络连接
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
应用显示过滤器 … <Ctrl-/>
表达式…
+

```
No.   Time         Source             Destination      Protocol Length Info
11256 186.571482   192.168.0.119      180.158.15.32    UDP      1107 23698 → 30864  Len=1065
11257 186.573450   180.158.15.32      192.168.0.119    UDP        81 30864 → 23698  Len=39
11258 186.573630   192.168.0.119      180.158.15.32    UDP      1107 23698 → 30864  Len=1065
11259 186.575399   120.70.4.218       192.168.0.119    UDP        79 3490 → 23698  Len=37
11260 186.575472   192.168.0.119      180.158.15.32    UDP      1107 23698 → 30864  Len=1065
11261 186.579400   180.158.15.32      192.168.0.119    UDP        79 30864 → 23698  Len=37
11262 186.595074   192.168.0.119      114.84.240.109   UDP      1109 23698 → 22428  Len=1067
11263 186.611448   192.168.0.119      180.158.15.32    UDP      1107 23698 → 30864  Len=1065
11264 186.677379   180.158.15.32      192.168.0.119    UDP        79 30864 → 23698  Len=37
11265 186.681393   120.70.4.218       192.168.0.119    UDP        79 3490 → 23698  Len=37
11266 186.681475   192.168.0.119      120.70.4.218     UDP      1107 23698 → 3490  Len=1065
11267 186.697304   120.70.4.218       192.168.0.119    UDP        79 3490 → 23698  Len=37
11268 186.701484   192.168.0.119      180.158.15.32    UDP      1107 23698 → 30864  Len=1065
11269 186.710852   HuaweiTe_6c:ca:49  Broadcast        ARP        42 Who has 192.168.0.1? Tell 192.168.0.120
11270 186.739305   180.173.128.120    192.168.0.119    UDP        81 1036 → 23698  Len=39
11271 186.744380   120.70.4.218       192.168.0.119    UDP        79 3490 → 23698  Len=37
11272 186.752312   120.70.4.218       192.168.0.119    UDP        79 3490 → 23698  Len=37
11273 186.753229   192.168.0.119      180.158.15.32    UDP      1107 23698 → 30864  Len=1065

▷ Frame 11269: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▷ Ethernet II, Src: HuaweiTe_6c:ca:49 (58:2a:f7:6c:ca:49), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▷ Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 58 2a  f7 6c ca 49 08 06 00 01   ......X* .l.I....
0010  08 00 06 04 00 01 58 2a  f7 6c ca 49 c0 a8 00 78   ......X* .l.I...x
0020  00 00 00 00 00 00 c0 a8  00 01                     ........ ..

无线网络连接: <live capture in progress>     分组: 28405 · 已显示: 28405 (100.0%)     配置文件: Default
```

Figure 17: Capture of ARP packets, showing details of a request

Find and select an ARP request for the default gateway and examine its fields. There are two kinds of ARP packets, a request and a reply, and we will look at each one in turn. The Info line for the request will start with "Who has …". You want to look for one of these packets that asks for the MAC address of the default gateway, e.g., "Who has xx.xx.xx.xx …" where xx.xx.xx.xx is your default gateway. You can click on the + expander or icon for the Address Resolution Protocol block to view the fields:

- Hardware and Protocol type are set to constants that tell us the hardware is Ethernet and the protocol is IP. This matches the ARP translation from IP to Ethernet address.
- Hardware and Protocol size are set to 6 and 4, respectively. These are the sizes of Ethernet and IP addresses in bytes.
- The opcode field tells us that this is a request.
- Next come the four key fields, the sender MAC (Ethernet) and IP and the target MAC

(Ethernet) and IP. These fields are filled in as much as possible. For a request, the sender knows their MAC and IP address and fills them in. The sender also knows the target IP address – it is the IP address for which an Ethernet address is wanted. But the sender does not know the target MAC address, so it does not fill it in.

Next, select an ARP reply and examine its fields. The reply will answer a request and have an Info line of the form “xx.xx.xx.xx is at yy:yy:yy:yy:yy:yy”:

- The Hardware and Protocol type and sizes are as set as before.
- The opcode field has a different value that tells us that this is a reply.
- Next come the four key fields, the sender MAC (Ethernet) and IP and the target MAC (Ethernet) and IP just as before. These fields are reversed from the corresponding request, since the old target is the new sender (and vice versa). The fields should now be all filled in since both computers have supplied their addresses.

Step 3: ARP request and reply

To show your understanding of an ARP exchange, draw a figure that shows the ARP request and reply packets sent between your computer and the default gateway. Make it for the case we examined of your computer doing an ARP for the default gateway. Label one packet the request and the other the reply. Give the sender and target MAC and IP addresses for each packet; you can use Wireshark to inspect the packets to get these values. Finally, circle the sought after Ethernet address on your drawing to show where it comes from in the exchange.

Turn-in: Hand in your drawing of the ARP exchange.

Step 4: Details of ARP over Ethernet

To look at further details of ARP, examine an ARP request and ARP reply to answer these questions:

1. *What opcode is used to indicate a request? What about a reply?*
2. *How large is the ARP header for a request? What about for a reply?*
3. *What value is carried on a request for the unknown target MAC address?*

ARP packets are carried in Ethernet frames, and the values of the Ethernet header fields are chosen to support ARP. For instance, you may wonder how an ARP request packet is delivered to the target computer so that it can reply and tell the requestor its MAC address. The answer is that the ARP request is (normally) broadcast at the Ethernet layer so that it is received by all computers on the local network including the target. Look specifically at the destination Ethernet address of a request: it is set to `ff:ff:ff:ff:ff:ff`, the broadcast address. So the target receives the request and recognizes that it is the intended recipient of the message; other computers that receive the request know that it is not meant for them. Only the target responds with a reply. However, anyone who receives an ARP packet can learn a mapping from it: the sender MAC and sender IP pair.

Examine an ARP request and reply to answer these questions:

4. *What Ethernet Type value which indicates that ARP is the higher layer protocol?*
5. *Is the ARP reply broadcast (like the ARP request) or not?*

Turn-in: Hand in your answers to the above questions.

Explore on your own

We encourage you to explore ARP on your own once you have completed this lab. One suggestion is to look at other ARP packets that may have been recorded in your trace; we only examined an ARP request by your computer and the ARP reply from the default gateway.

To see if there is other ARP activity, make sure to clear any Ethernet address filter that is set. Other ARP packets may exhibit any of the following kinds of behavior for you to explore:

- ARP requests broadcast by other computers. The other computers on the local network are also using ARP. Since requests are broadcast, your computer will receive their requests.
- ARP replies sent by your computer. If another computer happens to ARP for the IP address of your computer, then your computer will send an ARP reply to tell it the answer.
- Gratuitous ARPs in which your computer sends a request or reply about itself. This is helpful when a computer or link comes up to make sure that no-one else is using the same IP address. Gratuitous ARPs have the same sender and target IP address, and they have an Info field in Wireshark that identified them as gratuitous.
- Other ARP requests sent by your computer and the corresponding ARP reply. Your computer may need to ARP for other hosts besides the default gateway after you flush its ARP cache.

实验五　UDP

Objective

To look at the details of UDP (User Datagram Protocol). UDP is a transport protocol used throughout the Internet as an alternative to TCP when reliability is not required. It is covered in §6.4 of your text. Review that section before doing this lab.

Requirements

Wireshark: This lab uses the Wireshark software tool to capture and examine a packet trace. A packet trace is a record of traffic at a location on the network, as if a snapshot was taken of all the bits that passed across a particular wire.　The packet trace records a timestamp for each packet, along with the bits that make up the packet, from the lower-layer headers to the higher-layer contents. Wireshark runs on most operating systems, including Windows, Mac and Linux. It provides a graphical UI that shows the sequence of packets and the meaning of the bits when interpreted as protocol headers and data. It color-codes packets by their type, and has various ways to filter and analyze packets to let you investigate the behavior of network protocols. Wireshark is widely used to troubleshoot networks. You can download it from www.wireshark.org if it is not already installed on your computer. We highly recommend that you watch the short, 5 minute video "Introduction to Wireshark" that is on the site.

ifconfig / ipconfig: This lab uses the "`ipconfig`" (Windows) or "`ifconfig`" (Mac/Linux) command-line utility to inspect the state of your computer's network interface. `ifconfig/ipconfig` is installed as part of the operating system on Windows, Linux, and Mac computers.

Browser: This lab uses a web browser to find or fetch pages as a workload. Any web browser will do.

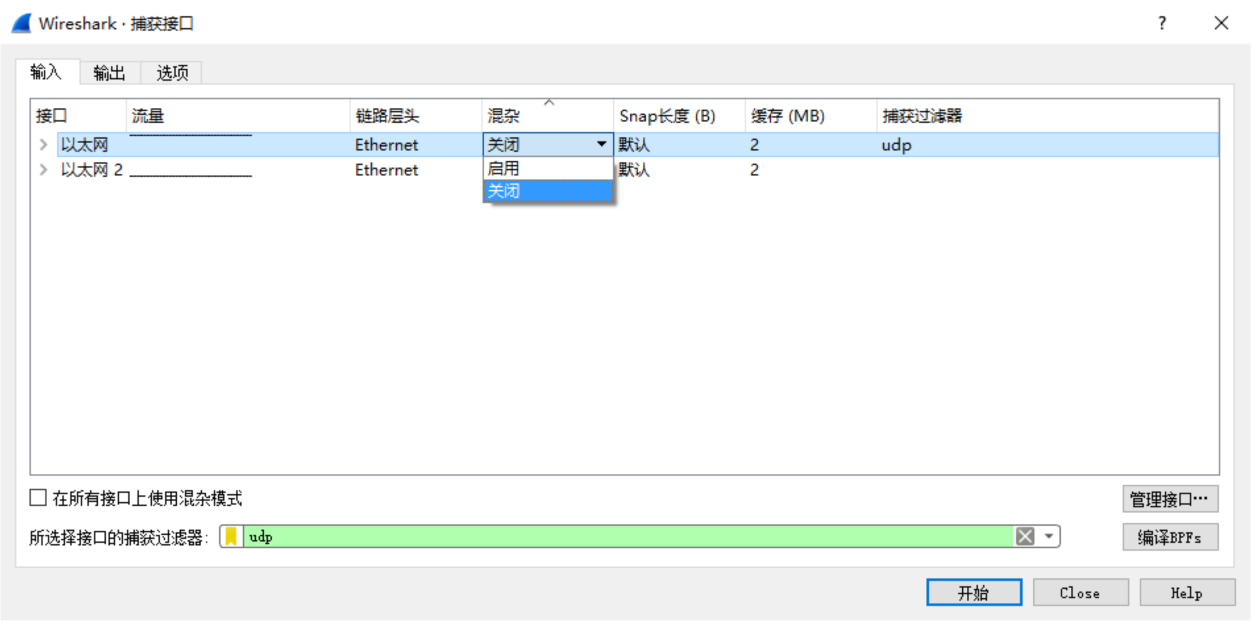
Step 1: Capture a Trace

There are many ways to cause your computer to send and receive UDP messages since UDP is widely used as a transport protocol. The easiest options are to:

- Do nothing but wait for a while. UDP is used for many “system protocols” that typically run in the background and produce small amounts of traffic, e.g., DHCP for IP address assignment and NTP for time synchronization.
- Use your browser to visit sites. UDP is used by DNS for resolving domain names to IP addresses, so visiting fresh sites will cause DNS traffic to be sent. Be careful not to visit unsafe sites; pick recommended sites or sites you know about but have not visited recently. Simply browsing the web is likely to cause a steady stream of DNS traffic.
- Start up a voice-over-IP call with your favorite client. UDP is used by RTP, which is the protocol commonly used to carry media samples in a voice or video call over the Internet.

Proceed as follows to capture a trace of UDP traffic; alternatively, you may use a supplied trace:

6. *Launch Wireshark and start a capture with a filter of* “`udp`“. Press ctrl+k to set up capture options. Your option window should be similar to the one pictured below. Select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful. Switch “capture packets in promiscuous mode（混杂模式）” to off. This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets sent to/from your computer. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive.



Wireshark · 捕获接口

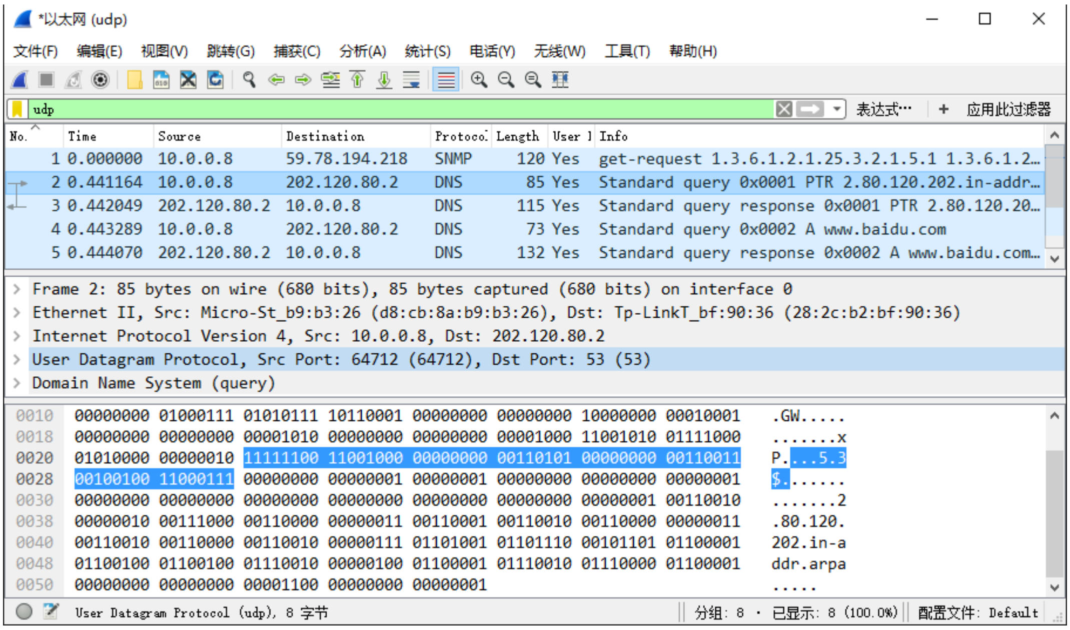
接口	流量	链路层头	混杂	Snap长度 (B)	缓存 (MB)	捕获过滤器
以太网		Ethernet	关闭	默认	2	udp
以太网 2		Ethernet		默认	2	

☐ 在所有接口上使用混杂模式

所选择接口的捕获过滤器: udp

Figure 1: Setting up the capture options

7. *When the capture is started, perform some activities that will generate UDP traffic.* We described several options above, e.g., browse the web.
8. *Wait a little while (say 60 seconds) after you have stopped your activity to also observe any background UDP traffic*. It is likely that you will observe a trickle of UDP traffic because system activity often uses UDP to communicate. We want to see some of this activity.
9. *Use the Wireshark menus or buttons to stop the capture*. You should now have a trace with possibly many UDP packets. Our example is shown below. We have selected a packet and expanded the detail of the UDP header.



*以太网 (udp)

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
表达式…
+
应用此过滤器

udp

```
No. Time     Source        Destination    Protoco Length User I Info
  1 0.000000 10.0.0.8      59.78.194.218  SNMP     120 Yes  get-request 1.3.6.1.2.1.25.3.2.1.5.1 1.3.6.1.2…
  2 0.441164 10.0.0.8      202.120.80.2   DNS       85 Yes  Standard query 0x0001 PTR 2.80.120.202.in-addr…
  3 0.442049 202.120.80.2  10.0.0.8       DNS      115 Yes  Standard query response 0x0001 PTR 2.80.120.20…
  4 0.443289 10.0.0.8      202.120.80.2   DNS       73 Yes  Standard query 0x0002 A www.baidu.com
  5 0.444070 202.120.80.2  10.0.0.8       DNS      132 Yes  Standard query response 0x0002 A www.baidu.com…

> Frame 2: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface 0
> Ethernet II, Src: Micro-St_b9:b3:26 (d8:cb:8a:b9:b3:26), Dst: Tp-LinkT_bf:90:36 (28:2c:b2:bf:90:36)
> Internet Protocol Version 4, Src: 10.0.0.8, Dst: 202.120.80.2
> User Datagram Protocol, Src Port: 64712 (64712), Dst Port: 53 (53)
> Domain Name System (query)

0010  00000000 01000111 01010111 10110001 00000000 00000000 10000000 00010001   .GW.....
0018  00000000 00000000 00001010 00000000 00000000 00001000 11001010 01111000   .......x
0020  01010000 00000010 11111100 11001000 00000000 00110101 00000000 00110011   P....5.3
0028  00100100 11000111 00000000 00000001 00000001 00000000 00000000 00000001   $.......
0030  00000000 00000000 00000000 00000000 00000000 00000000 00000001 00110010   .......2
0038  00000010 00111000 00110000 00000011 00110001 00110010 00110000 00000011   .80.120.
0040  00110010 00110000 00110010 00000111 01101001 01101110 00101101 01100001   202.in-a
0048  01100100 01100100 01110010 00000100 01100001 01110010 01110000 01100001   ddr.arpa
0050  00000000 00000000 00001100 00000000 00000001                              .....

User Datagram Protocol (udp), 8 字节 || 分组: 8 · 已显示: 8 (100.0%) || 配置文件: Default
```

Figure 2: Trace of UDP traffic showing the details of the UDP header

Step 2: Inspect the Trace

Different computers are likely to capture different kinds of UDP traffic depending on the network setup and local activity. Observe that the protocol column is likely to show multiple protocols, none of which is UDP. This is because the listed protocol is an application protocol layered on top of UDP. Wireshark gives the name of the application protocol, not the (UDP) transport protocol unless Wireshark cannot determine the application protocol. However, even if the packets are listed as an application protocol, they will have a UDP protocol header for us to study, following the IP and lower-layer protocol headers.

Select different packets in the trace (in the top panel) and browse the expanded UDP header (in the middle panel). You will see that it contains the following fields:

- Source Port, the port from which the UDP message is sent. It is given as a number and possibly a text name; names are given to port values that are registered for use with a specific application.
- Destination Port. This is the port number and possibly name to which the UDP message is destined. Ports are the only form of addressing in UDP. There computer is identified using the IP address in the lower IP layer.
- Length. The length of the UDP message.
- Checksum. A checksum over the message that is used to validate its contents. Is your checksum carrying 0 and flagged as incorrect for UDP messages sent from your computer? On some computers, the operating system software leaves the checksum blank (zero) for the NIC to compute and fill in as the packet is sent. This is called protocol offloading. It happens after Wireshark sees the packet, which causes Wireshark to believe that the checksum is wrong and flag it with a different color to signal a problem. You can remove these false errors if they are occurring by telling Wireshark not to validate the checksums. Select “Preferences” from the Wireshark menus and expand the “Protocols” area. Look under the list until you come to UDP. Uncheck “Validate checksum if possible”.

That is it. The UDP header has different values for different messages, but as you can see, it is short and sweet. The remainder of the message is the UDP payload that is normally identified the higher-layer protocol that it carries, e.g., DNS, or RTP.

Step 3: UDP Message Structure

To check your understanding of UDP, sketch a figure of the UDP message structure as you observed. It should show the position of the IP header, UDP header, and UDP payload. Within the UDP header, show the position and size of each UDP field you can observe using Wireshark. Your figure can simply show the message as a long, thin rectangle.

Try not to look at the figure of a UDP segment in your text; check it afterwards to note and investigate any differences. To work out sizes, observe that when you click on a protocol block in the middle panel (the block itself) then Wireshark will highlight the bytes it corresponds to in the packet in the lower panel and display the length at the bottom of the window.

By looking at the details of the UDP messages in your trace, answer these questions:

1. *What does the Length field include? The UDP payload, UDP payload and UDP header, or UDP payload, UDP header, and lower layer headers?*
2. *How long in bits is the UDP checksum?*
3. *How long in bytes is the entire UDP header?*

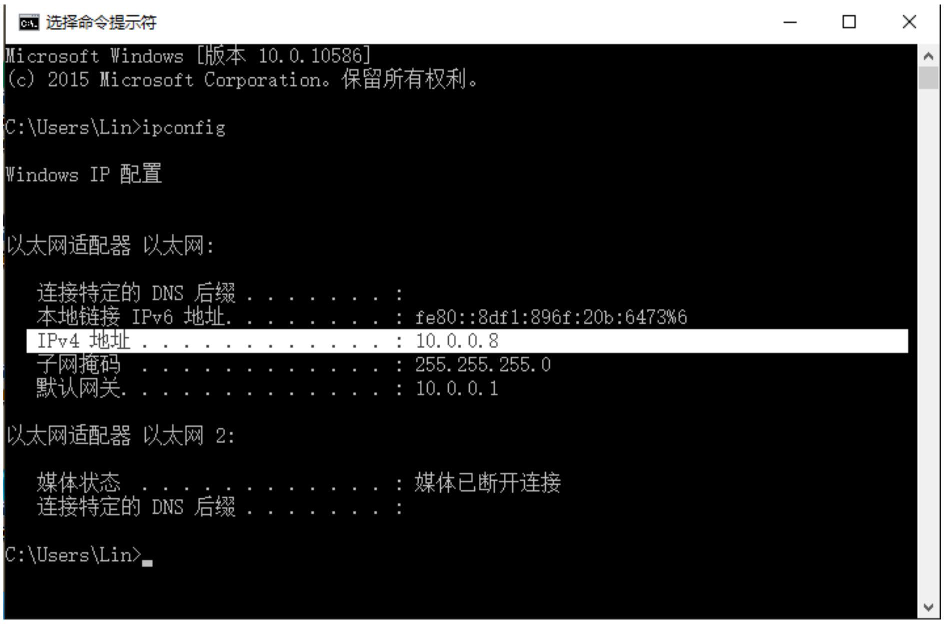
Turn-in: Hand in your drawing of a UDP message and the answers to the questions above.

Step 4: UDP Usage

To complete our understanding of UDP, we will look at how UDP is used in practice as a transport by applications. Beginning with IP, the next lower protocol layer, there are several issues we can consider. A first issue is how IP knows that the next higher protocol layer is UDP. The answer is that there is a Protocol field in the IP header that contains this information.

1. *Give the value of the IP Protocol field that identifies the upper layer protocol as UDP.*

A second issue is how UDP messages are typically addressed at the IP layer. You might be surprised to find UDP messages in your trace that neither come from your computer or are sent only to your computer. You can see this by sorting on the Source and Destination columns. The source and destinations will be domain names, if Network layer name resolution is turned, and otherwise IP addresses. (You can toggle this setting using the View menu and selecting Name resolution.) You can find out the IP address of your computer using the "`ipconfig`" command (Windows) or "`ifconfig`" command (Mac/Linux). Simply type this command into a terminal window and look for the IPv4 address of the main interface. We have given examples below.



```
Microsoft Windows [版本 10.0.10586]
(c) 2015 Microsoft Corporation。保留所有权利。

C:\Users\Lin>ipconfig

Windows IP 配置


以太网适配器 以太网:

   连接特定的 DNS 后缀 . . . . . . . :
   本地链接 IPv6 地址. . . . . . . . : fe80::8df1:896f:20b:6473%6
   IPv4 地址 . . . . . . . . . . . . : 10.0.0.8
   子网掩码  . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 10.0.0.1

以太网适配器 以太网 2:

   媒体状态  . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :

C:\Users\Lin>
```

Figure 3: Finding the computer's IP address (Windows)

The reason you may find UDP messages without your computer's IP address as either the source or destination IP address is that UDP is widely used as part of system protocols. These protocols often send messages to all local computers who are interested in them using broadcast and multicast addresses. In our traces, we find DNS (the domain name system), MDNS (DNS traffic that uses IP multicast), NTP (for time synchronization), NBNS (NetBIOS traffic), DHCP (for IP address assignment), SSDP (a service discovery protocol), STUN (a NAT traversal protocol), RTP (for carrying audio and video samples), and more. Your trace may have other protocols you have not heard about; it is OK, as there are a lot of protocols out there. You can look them up on the web for fun.

2. *Examine the UDP messages and give the destination IP addresses that are used when your computer is neither the source IP address nor the destination IP address.* (If you have only your computer as the source or destination IP address then you may use the supplied trace.)

Finally, let us look at the lengths of typical UDP messages. We know that UDP messages can be as large as roughly 64Kbytes. But as you browse you should see that most UDP messages are much shorter than this maximum, so that UDP messages fit in a single packet.

3. *What is the typical size of UDP messages in your trace?*

Turn-in: Hand in your answers to the questions above.

Explore on your own

We encourage you to keep exploring on your own, but there is not much more to UDP. Instead, you might examine the traffic of UDP-based applications to look at packet sizes and loss rates. Voice-over-IP and its companion protocols like RTP (Real-Time Protocol) are good candidates. Similarly, you might explore streaming and real-time applications to see which use UDP and which use TCP as a transport.

[END]

实验六 TCP

Objective

To see the details of TCP (Transmission Control Protocol). TCP is the main transport layer protocol used in the Internet. It is covered in §6.5 of your text. Review that section before doing this lab.

Requirements

Wireshark: This lab uses Wireshark to capture or examine a packet trace. A packet trace is a record of traffic at some location on the network, as if a snapshot was taken of all the bits that passed across a particular wire. The packet trace records a timestamp for each packet, along with the bits that make up the packet, from the low-layer headers to the higher-layer contents. Wireshark runs on most operating systems, including Windows, Mac and Linux. It provides a graphical UI that shows the sequence of packets and the meaning of the bits when interpreted as protocol headers and data. The packets are color-coded to convey their meaning, and Wireshark includes various ways to filter and analyze them to let you investigate different aspects of behavior. It is widely used to troubleshoot networks. You can download Wireshark from www.wireshark.org if it is not already installed on your computer. We highly recommend that you watch the short, 5 minute video "Introduction to Wireshark" that is on the site.

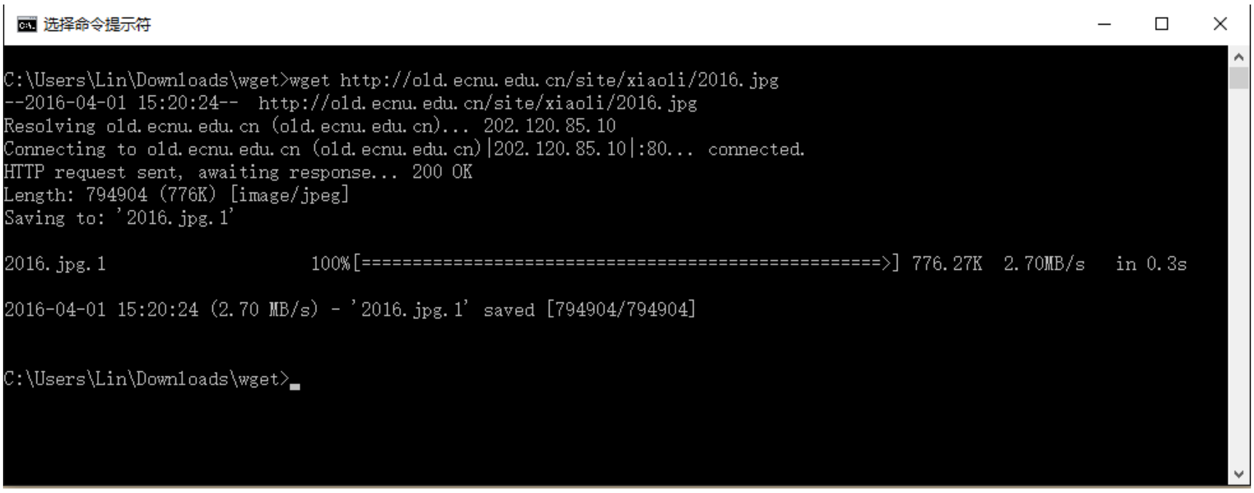
wget / curl: This lab uses `wget` (Linux and Windows) and `curl` (Mac) to fetch web resources. `wget` and `curl` are command-line programs that let you fetch a URL. Unlike a web browser, which fetches and executes entire pages, `wget` and `curl` give you control over exactly which URLs you fetch and when you fetch them. Under Linux, `wget` can be installed via your package manager. Under Windows, `wget` is available as a binary; look for download information on http://www.gnu.org/software/wget/. Under Mac, `curl` comes installed with the OS. Both have many options (try "`wget --help`" or "`curl --help`" to see) but a URL can be fetched simply with "`wget` *`URL`*" or "`curl` *`URL`*".

Browser: This lab uses a web browser to find or fetch pages as a workload. Any web browser will do.

Step 1: Capture a Trace

Proceed as follows to capture a trace of a single TCP connection that sends a moderate amount of data; alternatively, you may use a supplied trace. Many applications use TCP as a transport, including web browsers. So we will simply perform a web download to exercise a TCP connection. However, note that TCP is able to transfer data in both directions at the same time, but with a download content is only sent from the remote server the local computer (after the initial request).

10. *Find a URL of a single moderately-sized resource, and that you can download using HTTP (rather than HTTPS).* You may use your browser to search, perhaps looking for a picture (.jpg) or PDF document (.pdf). You want to ensure that it is a single resource and not a web page (e.g., a .html) with many inlined resources.
11. *Fetch the URL with* `wget` *or* `curl` *to check that you are able to retrieve at least 500 KB of content over at least several of network time seconds.* For example, use the command “`wget` http://old.ecnu.edu.cn/site/xiaoli/2016.jpg”. If the fetch does not work then try a different URL; keep in mind that you may be referring to a URL by a shortcut for which browsers must do work to find the intended content. Successful examples of fetching are shown in the figures below.

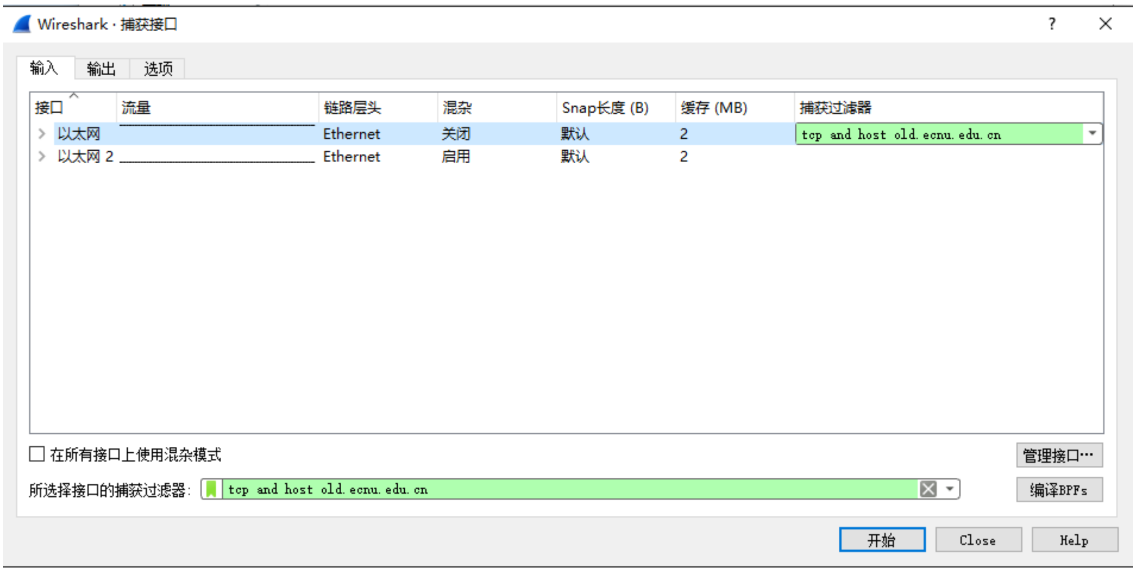


```
C:\Users\Lin\Downloads\wget>wget http://old.ecnu.edu.cn/site/xiaoli/2016.jpg
--2016-04-01 15:20:24--  http://old.ecnu.edu.cn/site/xiaoli/2016.jpg
Resolving old.ecnu.edu.cn (old.ecnu.edu.cn)... 202.120.85.10
Connecting to old.ecnu.edu.cn (old.ecnu.edu.cn)|202.120.85.10|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 794904 (776K) [image/jpeg]
Saving to: '2016.jpg.1'

2016.jpg.1              100%[================================================>] 776.27K  2.70MB/s   in 0.3s

2016-04-01 15:20:24 (2.70 MB/s) - '2016.jpg.1' saved [794904/794904]


C:\Users\Lin\Downloads\wget>
```

Figure 18: A successful fetch of a web resource with `wget` (Windows)

12. *Launch Wireshark and start a capture with a filter of* “`tcp and host xx.xx.xx`”, *where* `xx.xx.xx` *is the name of the remote server from which you will fetch content, e.g.,* “`old.ecnu.edu.cn`” *in the figure showing our example below.* The idea of the filter is to only capture TCP traffic between your computer and the server. Your capture window should be similar to the one pictured below, other than our highlighting. Select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful. Uncheck “capture packets in promiscuous mode”. This mode is useful to overhear packets sent to/from other

computers on broadcast networks. We only want to record packets sent to/from your computer. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive.



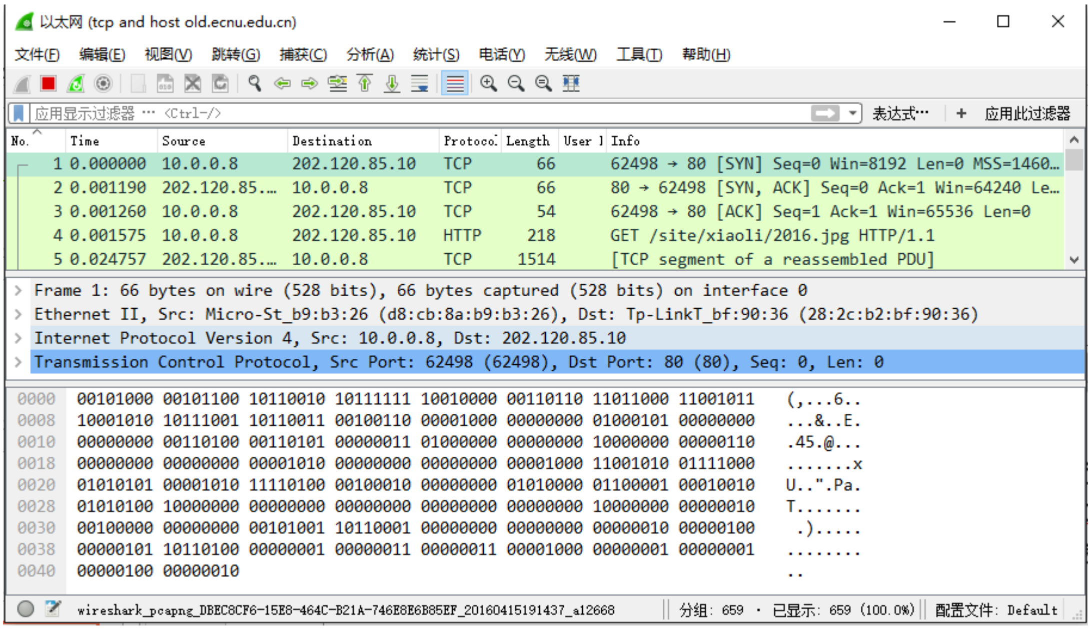
接口	流量	链路层头	混杂	Snap长度 (B)	缓存 (MB)	捕获过滤器
以太网		Ethernet	关闭	默认	2	tcp and host old.ecnu.edu.cn
以太网 2		Ethernet	启用	默认	2	

□ 在所有接口上使用混杂模式

所选择接口的捕获过滤器: tcp and host old.ecnu.edu.cn

Figure 19: Setting up the capture options

13. *After the capture is started, repeat the* `wget`*/*`curl` *command above.* This time, the packets will also be recorded by Wireshark.
14. *When the command is complete, return to Wireshark and use the menus or buttons to stop the trace.* You should now have a trace similar to that shown in the figure below. We have expanded the detail of the TCP header in our view, since it is our focus for this lab.



```
No.  Time      Source         Destination    Protocol Length User I Info
   1 0.000000  10.0.0.8       202.120.85.10  TCP        66          62498 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460…
   2 0.001190  202.120.85.…   10.0.0.8       TCP        66          80 → 62498 [SYN, ACK] Seq=0 Ack=1 Win=64240 Le…
   3 0.001260  10.0.0.8       202.120.85.10  TCP        54          62498 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
   4 0.001575  10.0.0.8       202.120.85.10  HTTP      218          GET /site/xiaoli/2016.jpg HTTP/1.1
   5 0.024757  202.120.85.…   10.0.0.8       TCP      1514          [TCP segment of a reassembled PDU]
```

```
> Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: Micro-St_b9:b3:26 (d8:cb:8a:b9:b3:26), Dst: Tp-LinkT_bf:90:36 (28:2c:b2:bf:90:36)
> Internet Protocol Version 4, Src: 10.0.0.8, Dst: 202.120.85.10
> Transmission Control Protocol, Src Port: 62498 (62498), Dst Port: 80 (80), Seq: 0, Len: 0
```

```
0000  00101000 00101100 10110010 10111111 10010000 00110110 11011000 11001011   (,...6..
0008  10001010 10111001 10110011 00100110 00001000 00000000 01000101 00000000   ...&..E.
0010  00000000 00110100 00110101 00000011 01000000 00000000 10000000 00000110   .45.@...
0018  00000000 00000000 00001010 00000000 00000000 00001000 11001010 01111000   .......x
0020  01010101 00001010 11110100 00100010 00000000 01010000 01100001 00010010   U..".Pa.
0028  01010100 10000000 00000000 00000000 00000000 00000000 10000000 00000010   T.......
0030  00100000 00000000 00101001 10110001 00000000 00000000 00000010 00000100    .).....
0038  00000101 10110100 00000001 00000011 00000011 00001000 00000001 00000001   ........
0040  00000100 00000010                                                         ..
```

wireshark_pcapng_DBEC8CF6-15E8-464C-B21A-746E8E6B85EF_20160415191437_a12668 || 分组: 659 · 已显示: 659 (100.0%) || 配置文件: Default

Figure 20: Trace of TCP traffic showing the details of the TCP header

Step 2: Inspect the Trace

Select a long packet anywhere in the middle of your trace whose protocol is listed as TCP. Expand the TCP protocol section in the middle panel (by using the "+" expander or icon). All packets except the initial HTTP GET and last packet of the HTTP response should be listed as TCP. Picking a long packet ensures that we are looking at a download packet from the server to your computer. Looking at the protocol layers, you should see an IP block before the TCP block. This is because the TCP segment is carried in an IP. We have shown the TCP block expanded in our figure.

You will see roughly the following fields:

- First comes the source port, then the destination port. This is the addressing that TCP adds beyond the IP address. The source port is likely to be 80 since the packet was sent by a web server and the standard web server port is 80.
- Then there is the sequence number field. It gives the position in the bytestream of the first payload byte.
- Next is the acknowledgement field. It tells the last received position in the reverse byte stream.
- The header length giving the length of the TCP header.
- The flags field has multiple flag bits to indicate the type of TCP segment. You can expand it and look at the possible flags.
- Next is a checksum, to detect transmission errors.
- There may be an Options field with various options. You can expand this field and explore if you would like, but we will look at the options in more detail later.
- Finally, there may be a TCP payload, carrying the bytes that are being transported.

As well as the above fields, there may be other informational lines that Wireshark provides to help you interpret the packet. We have covered only the fields that are carried across the network.

Step 3: TCP Segment Structure

To show your understanding of TCP, sketch a figure of the TCP segment you studied. It should show the position and size in bytes of the TCP header fields you can observe using Wireshark. Do not break down the Flags field, or any Options field, and if you find that some TCP fields share a byte then group them. As usual, your figure can simply show the frame as a long, thin rectangle. Try not to look at the figure of a TCP segment in your text; check it afterwards to note and investigate any differences.

To work out sizes, observe that when you click on a protocol block in the middle panel (the block itself, not the "+" expander) Wireshark will highlight the corresponding bytes in the packet in the lower panel, and display the length at the bottom of the window. You may also use the overall packet size shown in the Length column or Frame detail block. Note that this method will not tell you sub-byte positions.

Turn-in: Hand in your drawing of a TCP segment.

Step 4: TCP Connection Setup/Teardown

Three-Way Handshake

To see the "three way handshake" in action, look for a TCP segment with the SYN flag on, most likely at the beginning of your trace, and the packets that follow it. The SYN flag is noted in the Info column. You can also search for packets with the SYN flag on using the filter expression "`tcp.flags.syn==1`". A "SYN packet" is the start of the three-way handshake. In this case it will be sent from your computer to the remote server. The remote server should reply with a TCP segment with the SYN and ACK flags set, or a "SYN ACK packet". On receiving this segment, your computer will ACK it, consider the connection set up, and begin sending data, which in this case will be the HTTP request. Your exchange should follow this pattern, though it is possible that it differs slightly if a packet was lost and must be retransmitted.

Draw a time sequence diagram of the three-way handshake in your trace, up to and including the first data packet (the HTTP GET request) sent by your computer when the connection is established Put your computer on the left side and the remote server on the right side. As usual, time runs down the page, and lines across the page indicate segments. The result will be similar to diagrams such as Fig. 6-37.

Include the following features on your diagram:

- *The Sequence and ACK number, if present, on each segment.* The ACK number is only carried if the segment has the ACK flag set.
- *The time in milliseconds, starting at zero, each segment was sent or received at your computer.*
- *The round-trip time to the server estimated as the difference between the SYN and SYN-ACK segments.*

Connection Options

As well as setting up a connection, the TCP SYN packets negotiate parameters between the two ends using Options. Each end describes its capabilities, if any, to the other end by including the appropriate Options on its SYN. Often both ends must support the behavior for it to be used during data transfer.

Answer the following question:

1. *What TCP Options are carried on the SYN packets for your trace?*

Common Options include Maximum Segment Size (MSS) to tell the other side the largest segment that can be received, and Timestamps to include information on segments for estimating the round trip time. There are also Options such as NOP (No-operation) and End of Option list that serve to format the Options but do not advertise capabilities. You do not need to include these formatting options in your answer above. Options can also be carried on regular segments after the connection is set up when they play a role in data transfer. This depends on the Option. For example: the MSS option is not carried on each packet because it does not convey new information; timestamps may be included on each packet to keep a fresh estimate of the RTT; and options such as SACK (Selective Acknowledgments) are used only when data is received out of order. For fun, look at the options on data packets in your trace.

FIN/RST Teardown

Finally, the TCP connection is taken down after the download is complete. This is typically done with FIN (Finalize) segments. Each side sends a FIN to the other and acknowledges the FIN they receive; it is similar to the three-way handshake. Alternatively, the connection may be torn down abruptly when one end sends a RST (Reset). This packet does not need to be acknowledged by the other side.

Draw a picture of the teardown in your trace, starting from when the first FIN or RST is issued until the connection is complete. As before, show the sequence and ACK numbers on each segment. If you have FINs then use the time difference to estimate the round-trip time.

Turn-in: Hand in your drawings and the answers to the above questions.

Step 5: TCP Data Transfer

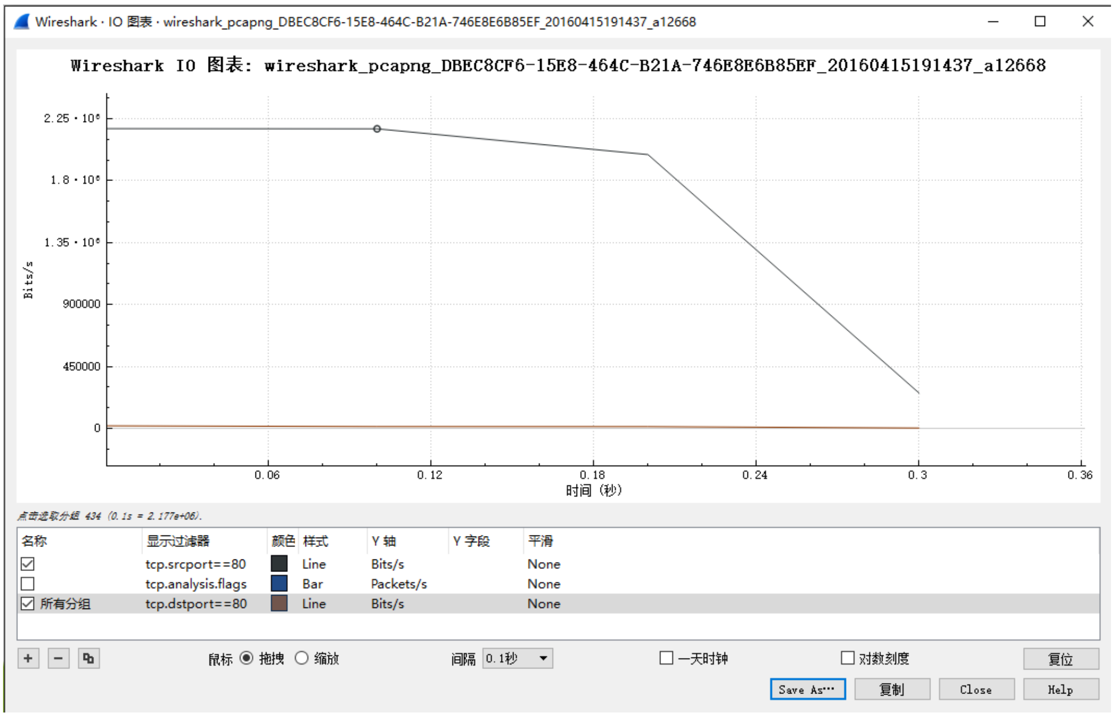
The middle portion of the TCP connection is the data transfer, or download, in our trace. This is the main event. To get an overall sense of it, we will first look at the download rate over time.

Under the Statistics(统计) menu select an "IO Graph（IO 图表）". By default, this graph shows the rate of packets over time. Tweak it to show the download rate with the changes given below. You might be tempted to use the "TCP Stream Graph" tools under the Statistics menu instead. However, these tools are not useful for our case because they assume the trace is taken near the computer sending the data; our trace is taken near the computer receiving the data.

- *On the x-axis, adjust the tick interval and pixels per tick.* The tick interval should be small enough to see into the behavior over the trace, and not so small that there is no averaging. 0.1 seconds is a good choice for a several second trace. The pixels per tick can be adjusted to make the graph wider or narrower to fill the window.
- *On the y-axis, change the unit to be Bits/s. The default is Packet/Tick.*
- *Add a filter expression to see only the download packets.* So far we are looking at all of the packets. Assuming the download is from the usual web server port of 80, you can filter for it with a filter of "`tcp.srcport==80`". Don't forget to click the Resrt(复位) button to cause it to redisplay.
- *To see the corresponding graph for the upload traffic, enter a second filter in the next box*. Again assuming the usual web server port, the filter is

"`tcp.dstport==80`". After you click the Resrt(复位) button, you should have two lines on the graph.

Our graph for this procedure is shown in the figure below. The download rate when the connection is running is approximately 2.6 Mbps. You can check your rate estimate with the information from `wget/curl`. The upload rate is a steady, small trickle of ACK traffic. Our download also proceeds fairly steadily until it is done. This is the ideal, but many downloads may display more variable behavior if, for example, the available bandwidth varies due to competing downloads, the download rate is set by the server rather than the network, or enough packets are lost to disrupt the transfer. You can click on the graph to be taken to the nearest point in the trace if there is a feature you would like to investigate.


Wireshark · IO 图表 · wireshark_pcapng_DBEC8CF6-15E8-464C-B21A-746E8E6B85EF_20160415191437_a12668
Wireshark IO 图表: wireshark_pcapng_DBEC8CF6-15E8-464C-B21A-746E8E6B85EF_20160415191437_a12668
Bits/s
2.25 · 10⁶
1.8 · 10⁶
1.35 · 10⁶
900000
450000
0
0.06
0.12
0.18
0.24
0.3
0.36
时间 (秒)
+
−
Save As…
复制
Close
Help

点击选取分组 434 (0.1s = 2.177e+06).

名称	显示过滤器	颜色	样式	Y 轴	Y 字段	平滑
☑	tcp.srcport==80	■	Line	Bits/s		None
☐	tcp.analysis.flags	■	Bar	Packets/s		None
☑ 所有分组	tcp.dstport==80	■	Line	Bits/s		None

鼠标 ● 拖拽 ○ 缩放 间隔 0.1秒 ☐ 一天时钟 ☐ 对数刻度 复位

Figure 21: TCP download rate over time via an IO graph

Answer the following questions to show your understanding of the data transfer:

1. *What is the rough data rate in the download direction in packets/second and bits/second once the TCP connection is running well?*
2. *What percentage of this download rate is content? Show your calculation*. To find out, look at a typical download packet; there should be many similar, large download packets. You can see how long it is, and how many bytes of TCP payload it contains.
3. *What is the rough data rate in the upload direction in packets/second and bits/second due to the ACK packets?*

Inspect the packets in the download in the middle of your trace for these features:

- You should see a pattern of TCP segments received carrying data and ACKs sent back to the server. Typically there will be one ACK every couple of packets. These ACKs are called Delayed ACKs. By delaying for a short while, the number of ACKs is halved.
- Since this is a download, the sequence number of received segments will increase; the ACK number of subsequently transmitted segments will increase correspondingly.
- Since this is a download, the sequence number of transmitted segments will not increase (after the initial get). Thus the ACK number on received segments will not increase either.
- Each segment carries Window information to tell the other end how much space remains in the buffer. The Window must be greater than zero, or the connection will be stalled by flow control.

Answer the following question:

4. *If the most recently received TCP segment from the server has a sequence number of X, then what ACK number does the next transmitted TCP segment carry?*

As well as regular TCP segments carrying data, you may see a variety of other situations. You can sort the trace on the Info column and browse the packets of type “`[TCP xxx ...`”. Depending on the download, you may see duplicate acks, out of order data, retransmissions, zero windows, window updates, and more. These segments are not generally distinguished by flags in the TCP header, like SYN or FIN segments. Instead, they are names for situations that may occur and be handled during transport.
Turn-in: Hand in your answers to the above questions.

Explore on your own

We encourage you to explore TCP on your own once you have completed this lab. Some ideas:

- Explore the congestion control and the classic AIMD behavior of TCP. To do this, you will likely want to capture a trace while you are sending (not receiving) a moderate amount of data on a TCP connection. You can then use the “TCP Stream Graph” tools as well as other analysis to observe how the congestion window changes over time.
- Explore the reliability mechanisms of TCP more deeply. Capture a trace of a connection that includes segment loss. See what triggers the retransmissions and when. Also look at the round-trip time estimator.
- Look at the use of options including SACK to work through the details. You should see information about ranges of received bytes during times of segment loss.

- TCP is the transport layer underlying the web. You can see how your browser makes use of TCP by setting up concurrent connections.

[END]

实验七 Socket Programming

Objective

To get the details of socket programming and simple client-server interaction. It is covered in §6.1 of your text. Review that section before doing this lab.

Requirements

You are going to build a **server** capable of receiving text messages from clients. The server should print these text messages on standard output, but should not print any other messages such as debug information. From the server perspective, a message corresponds to the data received from a particular client during a communication session with that client. The server should be listening for text messages to a port known to the clients. The server should be able to receive text messages from multiple clients. When multiple clients simultaneously try to send text messages to the server, the server should print them one at a time (in any order). Note that you don't have to implement an event-driven or multi-threaded server. Serving one client at a time is enough.

You should also implement a **client** to test the server. The client should receive the text message from standard input. The end of the message is marked by a control sequence which corresponds to hitting >ENTER< twice. This control sequence is also an instruction for the client to exit. This control sequence should not be transmitted. Also, if your client reads the text message from a file via pipes, and reaches EOF without seeing the control sequence, then the client should still transmit the message and exit.

This assignment can be completed in your favorite programming language. It should compile and run without errors by producing two binaries called server and client. The server should take as its first argument the port to listen to. The client should take as its first argument the name of the host that the server is running and the port that the server is listening. If the server cannot bind to the port that you specify, a message should be printed on standard error and the program should exit. You shouldn't assume that your server will be running on a particular IP address, or that clients will be coming from a predetermined IP address. Both the client and server should generate an appropriate error message and terminate when given invalid arguments.

You should test your code with long text messages (of size at least 20KB), not just short ones. You can use pipes to redirect the standard input of the client and standard output of the server. You should also test your code with multiple clients. (at least up to 5 simultaneous clients)

The deliverables are two files: "server.c" and "client.c" implementing the server and client as described above and an optional makefile for producing the executables.

NOTE: if your implementation of **client** and **server** is capable of duplex communication, you can get **5** extra points added to your final grade.

Example Implementation: simplex-talk

Here is an example implementation of a simple client/server program that uses the socket interface to send messages over a TCP connection. This application allows a user on one machine to type in and send text to a user on another machine. It is a simplified version of the Unix **talk** program, which is similar to the program at the core of an instant messaging application.

Client

The client side takes the name of the remote machine as an argument. It calls the Unix utility **gethostbyname** to translate this name into the remote host's IP address. The next step is to construct the address data structure (**sin**) expected by the socket interface. Notice that this data structure specifies that we'll be using the socket to connect to the Internet (**AF_INET**). Here we use TCP port 7701 as the server port. The final step in setting up the connection is to call **socket** and **connect**. Once the **connect** operation returns, the connection is established and the client program enters its main loop, which reads text from standard input and sends it over the socket.

```
/* Source: Peterson & Davie (2007), Computer Networks, a Systems
Approach,
 *         4th ed., Morgan Kaufmann, p. 34-35.
 * Included stdlib.h, string.h, and strings.h so it compiles on Linux.
 * Changed port from 5432 (postgresql) to 7701 (unassigned).
 * - JLND Feb 7 2009
 */

#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 7701
#define MAX_LINE 256
```

```
int
main(int argc, char * argv[])
{
  FILE *fp;
  struct hostent *hp;
  struct sockaddr_in sin;
  char *host;
  char buf[MAX_LINE];
  int s;
  int len;

  if (argc==2) {
    host = argv[1];
  }
  else {
    fprintf(stderr, "usage: simplex-talk host\n");
    exit(1);
  }

  /* translate host name into peer’s IP address */
  hp = gethostbyname(host);
  if (!hp) {
    fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
    exit(1);
  }

  /* build address data structure */
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
  sin.sin_port = htons(SERVER_PORT);

  /* active open */
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("simplex-talk: connect");
    close(s);
    exit(1);
  }

  /* main loop: get and send lines of text */
  while (fgets(buf, sizeof(buf), stdin)) {
    buf[MAX_LINE-1] = '\0';
    len = strlen(buf) + 1;
    send(s, buf, len, 0);
  }
}
```

Server

The server first constructs the address data structure by filling in its own port number (**SERVER_PORT**). By not specifying an IP address, the application program is willing to accept connections on any of the local host's IP addresses. Next, the server performs the preliminary steps involved in a passive open; it creates the socket, binds it to the local address, and sets the maximum number of pending connections to be allowed. Finally, the main loop waits for a remote host to try to connect, and when one does, it receives and prints out the characters that arrive on the connection.

```
/* Source: Peterson & Davie (2007), Computer Networks, a Systems
Approach,
 *          4th ed., Morgan Kaufmann, p. 35-36.
 * Included stdlib.h, string.h, and strings.h so it compiles on Linux.
 * Changed port from 5432 (postgresql) to 7701 (unassigned).
 * - JLND Feb 7 2009
 */

#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 7701
#define MAX_PENDING 5
#define MAX_LINE 256

int
main()
{
  struct sockaddr_in sin;
  char buf[MAX_LINE];
  int len;
  int s, new_s;

  /* build address data structure */
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = INADDR_ANY;
  sin.sin_port = htons(SERVER_PORT);

  /* setup passive open */
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
    perror("simplex-talk: bind");
    exit(1);
  }
  listen(s, MAX_PENDING);

  /* wait for connection, then receive and print text */
  while(1) {
    if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
      perror("simplex-talk: accept");
      exit(1);
    }
    while (len = recv(new_s, buf, sizeof(buf), 0))
      fputs(buf, stdout);
    close(new_s);
  }
}
```

[END]

附录 华东师范大学软件工程学院实验报告

实验课程：	年级：	实验成绩：
实验名称：	姓名：	
实验编号：	学号：	实验日期：
指导教师：	组号：	实验时间：

一、实验目的

本次实验所涉及并要求掌握的知识点。

二、实验内容与实验步骤

实验内容、原理分析及具体实验步骤。

三、实验环境

实验所使用的器件、仪器设备名称及规格。

四、实验过程与分析

详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。

根据具体实验，记录、整理相应数据表格、绘制曲线、波形图等。

五、实验结果总结

对实验结果进行分析，完成思考题目，总结实验的心得体会，并提出实验的改进意见。

六、附录