Department of Electrical and Computer Engineering

Laboratory Manual for CENG460 Communications Networks

By

Ruonan Zhang, Emad Shihab, Zhe Yang, Xuan Wang, Lei Zheng, and Lin Cai

The knowledge acquired in the CENG460 course and the labs should NOT be misused under any circumstances. Please carefully read and follow the Policy on “Responsible Use for Information Technology Services” (available at “http://www.uvic.ca/6030”) for using Information technology services at the University of Victoria.

Please refer to the CENG 460 lab web page for supplementary lab information.

This lab manual has adopted several contents from the labs suggested in [1, 3, 4, 5].

Contents

1 Lab 1: Introduction to WireShark and Layered Protocol 2

1.1 Overview . . . 3

1.1.1 WireShark . . . 3

1.1.2 Networking Tools . . . 4

1.1.3 Layered Protocol . . . 5

1.2 Procedures . . . 5

1.2.1 Installation . . . 5

1.2.2 Getting familiar with WireShark . . . 5

1.2.3 Layered Protocol . . . 7

1.3 Discussion . . . 10

1.3.1 Running WireShark . . . 10

1.3.2 Networking Tools . . . 10

1.3.3 Layered Protocol . . . 11

2 Lab 2: Ethernet and IEEE 802.11 12

2.1 Objective . . . 12

2.2 Introduction . . . 12

2.2.1 Ethernet . . . 12

2.2.2 IEEE 802.11 . . . 13

2.3 Procedures . . . 14

2.3.1 Analyzing Ethernet frames . . . 14

2.3.2 Exploring IEEE 802.11 functions . . . 14

2.4 Discussion . . . 15

2.4.1 Analyzing Ethernet frames . . . 15

2.4.2 Exploring IEEE 802.11 functions . . . 15

3 Lab 3: ARP, IP, and ICMP **17**
3.1 Objective 17
3.2 Introduction 17
3.2.1 Address Resolution Protocol (ARP) 17
3.2.2 Internet Protocol (IP) 18
3.2.3 Internet Control Message Protocol (ICMP) 18
3.3 Procedures 19
3.3.1 Exploring ARP functions 19
3.3.2 Analyzing IP frames 19
3.3.3 Exploring ICMP functions 20
3.4 Discussion 21
3.4.1 Exploring ARP functions 21
3.4.2 Analyzing IP frames 22
3.4.3 Exploring ICMP functions 23

4 Lab 4: TCP **24**
4.1 Objective 24
4.2 Introduction 24
4.2.1 TCP Header Format 26
4.2.2 TCP Connection Setup 26
4.2.3 TCP Data Flow 27
4.2.4 TCP Connection Release 27
4.2.5 TCP Congestion Control (Optional) 28
4.2.6 TCP Flow Control (Optional) 29
4.2.7 Retransmission in TCP 29
4.3 Procedures 30
4.3.1 TCP Header Format 30
4.3.2 TCP Connection Setup 31
4.3.3 TCP Data Flow 31
4.3.4 TCP Connection Release 32
4.3.5 TCP Congestion Control 32
4.3.6 TCP Flow Control 32
4.3.7 Retransmission in TCP (Optional) 33
4.4 Discussion 33
4.4.1 TCP Header Format 33
4.4.2 TCP Connection Setup 33
4.4.3 TCP Data Flow 34
4.4.4 TCP Connection Release 36

4.4.5 TCP Congestion Control 36
4.4.6 TCP Flow Control 37
4.4.7 Retransmission in TCP (Optional) 37

Chapter 1

Lab 1: Introduction to WireShark and Layered Protocol

The labs for this course were designed to help students better understand the ideas learned in the classes through hands-on experiments.

A better way to understand network protocols is to observe how they actually work. A basic tool for observing the messages exchanged between executing protocol entities is the **packet sniffer**, which is an essential part of **network protocol analyzer**. WireShark is a free and open-source network protocol analyzer that runs on various operating systems including Linux, Unix, Mac, and Windows. We will give a brief overview of it in the following section.

This lab has three parts. The first part includes simple tasks that let you get familiar with the basic operations of WireShark. The second part will introduce some handy networking tools, which will be used in the following labs. The third part will focus on how protocols and layering are represented in packets by exploring the sniffed packet traces.

1.1 Overview

1.1.1 WireShark

WireShark (previously called Ethereal) is one of the most widely used network protocol analyzer. It passively sniffs packets that are sent from or received by a designated network interface, but never sends packets itself. It receives a *copy* of packets that are sent from or received by the applications and protocols executing on the end-system (e.g., your computer). WireShark also has a graphical front-end to display the packets that it sniffs.

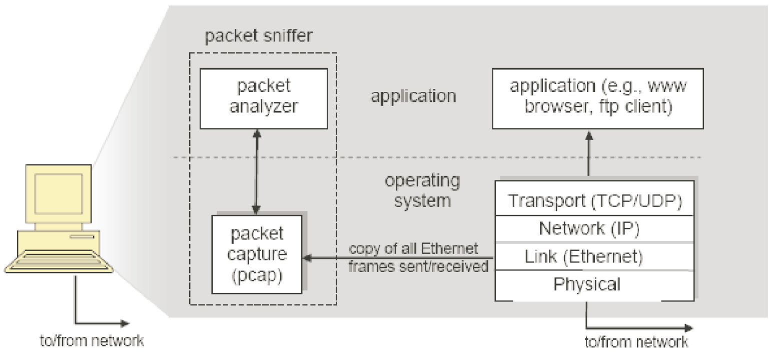


Figure 1.1: Network protocol analyzer structure

Fig. 1.1 [1] shows the structure of a network protocol analyzer. At the right of the figure shows the protocol stack and applications (such as a web browser or an FTP client) that normally run on your computer. The network protocol analyzer, shown within the dashed rectangle, has two parts, the packet capture and the packet analyzer. The packet capture library receives a copy of every link-layer frame that is sent from or received by a designated network interface. Recall that messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. In Fig. 1.1, the assumed physical media is an Ethernet, and so all upper layer protocols' headers are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent from or received by all protocols and applications executing in your computer.

The second component is the packet analyzer, which displays the contents of all fields within a link-layer frame. In order to do so, the packet analyzer must *understand* the structure of messages exchanged by the protocols. For example, we are interested in displaying the various fields in messages exchanged by the HTTP protocol in Fig. 1.1. The packet analyzer understands the format of Ethernet frames, and so it can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so it can extract the TCP segment within the IP datagram. It understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally, it understands the HTTP protocol and so, for example, knows that an HTTP message may contain the string of "GET", "POST", or "HEAD".

1.1.2 Networking Tools

ping

The *ping* program in the source host sends a packet to the target IP address; if the target is alive, the *ping* program in the target host responds by sending a packet back to the source host. Both of these *ping* packets carry ICMP messages. Try "ping --help" to find out its usage.

ifconfig

The *ifconfig* is a tool to configure a network interface, for instance, setting an interface's IP address and netmask, disabling or enabling a given interface. Try "ifconfig --help" to find out its usage.

netstat

The *netstat* is a tool that displays network connections, routing tables, and network interface statistics. It is used for finding problems in the network and to determine the amount of traffic on the network as a performance measurement. Try "netstat --help" to find its usage.

wget

wget is a command-line program that let you fetch a URL. Unlike a web browser, which fetches and executes the entire pages, wget give you the control over exactly which URLs you fetch and when you fetch them. wget has many options (try “wget --help” to see them) but a URL can be fetched simply with “wget URL”.

1.1.3 Layered Protocol

Two reference models are used to describe the network architecture, the OSI/ISO reference model and the TCP/IP reference model. The OSI/ISO model divides the network into seven layers and the TCP/IP model divides the network into four layers. No matter which model is used, the basic principle of the layered architecture is that each layer performs some services for the layer above it.

1.2 Procedures

1.2.1 Installation

WireShark is free to download at http://www.wireshark.org/. How to build and install WireShark onto machines with different operating systems can be referred to http://wiki.wireshark.org/BuildingAndInstalling.

1.2.2 Getting familiar with WireShark

A. Starting WireShark

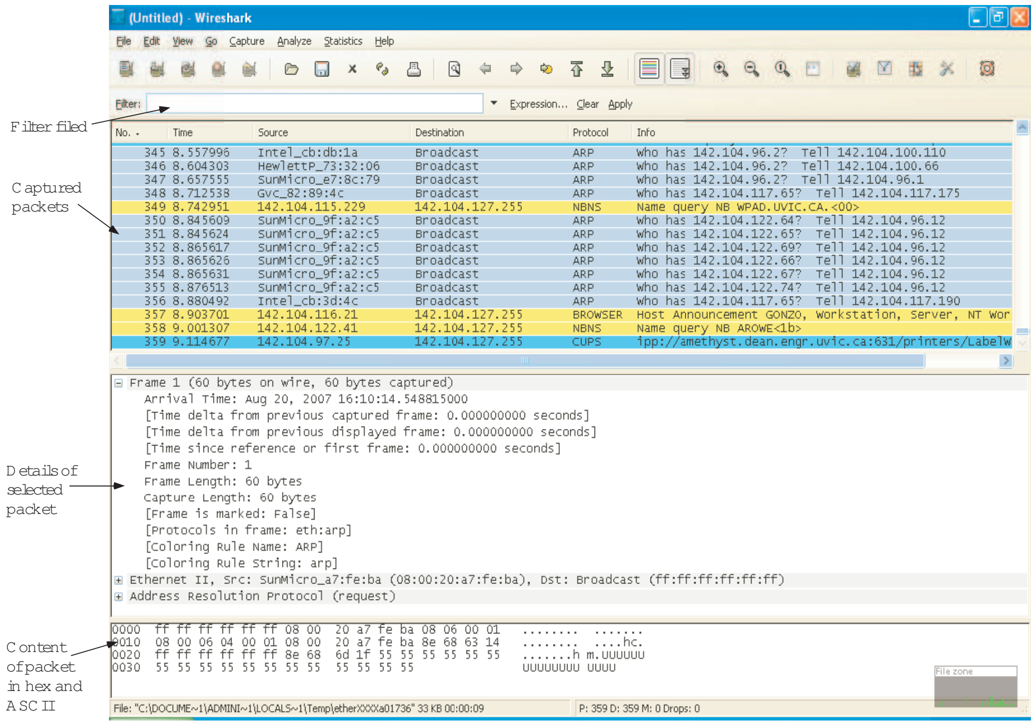
When you run WireShark, you will see the graphical user interface (GUI) as shown in Fig. 1.2. There are four main fields:

- **Filter field**: Used to filter out uninteresting packets with the entered specifications, so you can choose which packets should (not) be shown on the screen;

- **Captured packets**: Lists the packets captured by the selected interface;

- **Details of selected packet**: Lists information about the packet that is selected in the captured packets window;

- **Content of packet in hex/ASCII**: Displays the content of the captured packet, in hex and ASCII.


Filter filed
Captured packets
Details of selected packet
Content of packet in hex and ASCII
(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
File zone

```
No.  Time      Source              Destination      Protocol  Info
345 8.557996  Intel_cb:db:1a      Broadcast        ARP       Who has 142.104.96.2?  Tell 142.104.100.110
346 8.604303  HewlettP_73:32:06   Broadcast        ARP       Who has 142.104.96.2?  Tell 142.104.100.66
347 8.657555  SunMicro_e7:8c:79   Broadcast        ARP       Who has 142.104.96.2?  Tell 142.104.96.1
348 8.712538  Gvc_82:89:4c        Broadcast        ARP       Who has 142.104.117.65?  Tell 142.104.117.175
349 8.742951  142.104.115.229     142.104.127.255  NBNS      Name query NB WPAD.UVIC.CA.<00>
350 8.845609  SunMicro_9f:a2:c5   Broadcast        ARP       Who has 142.104.122.64?  Tell 142.104.96.12
351 8.845624  SunMicro_9f:a2:c5   Broadcast        ARP       Who has 142.104.122.65?  Tell 142.104.96.12
352 8.865617  SunMicro_9f:a2:c5   Broadcast        ARP       Who has 142.104.122.69?  Tell 142.104.96.12
353 8.865626  SunMicro_9f:a2:c5   Broadcast        ARP       Who has 142.104.122.66?  Tell 142.104.96.12
354 8.865631  SunMicro_9f:a2:c5   Broadcast        ARP       Who has 142.104.122.67?  Tell 142.104.96.12
355 8.876513  SunMicro_9f:a2:c5   Broadcast        ARP       Who has 142.104.122.74?  Tell 142.104.96.12
356 8.880492  Intel_cb:3d:4c      Broadcast        ARP       Who has 142.104.117.65?  Tell 142.104.117.190
357 8.903701  142.104.116.21      142.104.127.255  BROWSER   Host Announcement GONZO, Workstation, Server, NT Wor
358 9.001307  142.104.122.41      142.104.127.255  NBNS      Name query NB AROWE<1b>
359 9.114677  142.104.97.25       142.104.127.255  CUPS      ipp://amethyst.dean.engr.uvic.ca:631/printers/LabelW
```

```
⊟ Frame 1 (60 bytes on wire, 60 bytes captured)
    Arrival Time: Aug 20, 2007 16:10:14.548815000
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 60 bytes
    Capture Length: 60 bytes
    [Frame is marked: False]
    [Protocols in frame: eth:arp]
    [Coloring Rule Name: ARP]
    [Coloring Rule String: arp]
⊞ Ethernet II, Src: SunMicro_a7:fe:ba (08:00:20:a7:fe:ba), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Address Resolution Protocol (request)
```

```
0000  ff ff ff ff ff ff 08 00  20 a7 fe ba 08 06 00 01   ........ .......
0010  08 00 06 04 00 01 08 00  20 a7 fe ba 8e 68 63 14   ........ ....hc.
0020  ff ff ff ff ff ff 8e 68  6d 1f 55 55 55 55 55 55   .......h m.UUUUUU
0030  55 55 55 55 55 55 55 55  55 55 55 55              UUUUUUUU UUUU
```

```
File: "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\etherXXXXa01736" 33 KB 00:00:09    P: 359 D: 359 M: 0 Drops: 0
```

Figure 1.2: WireShark graphical user interface

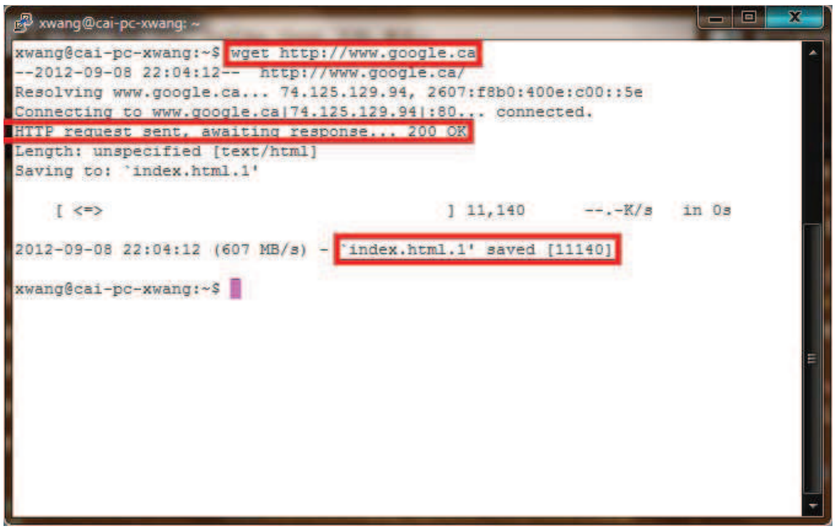
B. Capture Trace

Use the following procedure to capture the trace.

- Pick a URL and fetch it by *wget*. For example, **open a console**, type "wget http://www.google.ca", you will obtain the fetched resource written in a file. A successful example is shown in Fig. 1.3. The expected response is "200 OK".

- Close web browser. By minimizing browser activity you will stop your computer from fetching unnecessary web content, and avoid incidental traffic in the trace.

- Launch Wireshark. Choose the network interface that we would like to capture the packets on. To do this, select "Capture ⇒ Options" from the command menu. A window similar to the one shown in Fig. 1.4 should pop up. Select the interface you are using. Uncheck "Capture



```
xwang@cai-pc-xwang:~$ wget http://www.google.ca
--2012-09-08 22:04:12--  http://www.google.ca/
Resolving www.google.ca... 74.125.129.94, 2607:f8b0:400e:c00::5e
Connecting to www.google.ca|74.125.129.94|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: `index.html.1'

    [ <=>                                    ] 11,140      --.-K/s   in 0s

2012-09-08 22:04:12 (607 MB/s) - `index.html.1' saved [11140]

xwang@cai-pc-xwang:~$
```

Figure 1.3: Wget URL

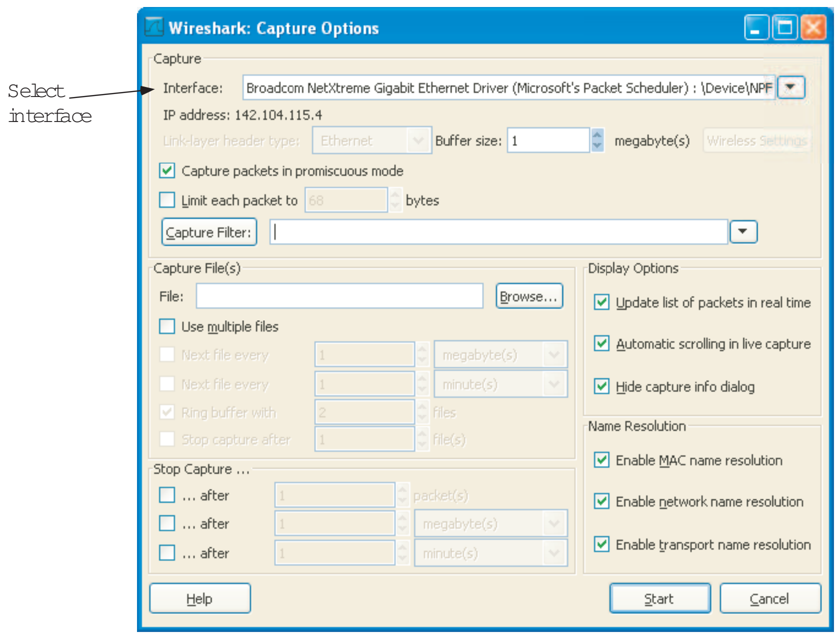
packets in promiscuous mode". This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets sent to/from your computer. Use capture filter "tcp port 80". This filter will record only standard web traffic and not other kinds of packets that your computer may send. Click "Start" to start the packet capture process.

- When the capture is started, repeat the web fetch using wget above. This time, the packets will be recorded by Wireshark as the content is transferred.

- After the fetch is successful, return to Wireshark and use the menus or buttons to stop the trace ("Capture ⇒ Stop"). If you have succeeded, the upper Wireshark window will show multiple packets. How many packets being captured will depend on the size of the web page, but there should be at least 8 packets in the trace. An example is shown in Fig. 1.5.

1.2.3 Layered Protocol

By inspecting the captured trace, or the provided trace (**lab1-wget-trace.pacp**) to understand the layered protocol.



Capture
Interface: Broadcom NetXtreme Gigabit Ethernet Driver (Microsoft's Packet Scheduler) : \Device\NPF
IP address: 142.104.115.4
Link-layer header type: Ethernet   Buffer size: 1 megabyte(s)   Wireless Settings
☑ Capture packets in promiscuous mode
☐ Limit each packet to 68 bytes

Capture File(s)
File:
☐ Use multiple files
☐ Next file every 1 megabyte(s)
☐ Next file every 1 minute(s)
☑ Ring buffer with 2 files
☐ Stop capture after 1 file(s)

Stop Capture ...
☐ ... after 1 packet(s)
☐ ... after 1 megabyte(s)
☐ ... after 1 minute(s)

Display Options
☑ Update list of packets in real time
☑ Automatic scrolling in live capture
☑ Hide capture info dialog

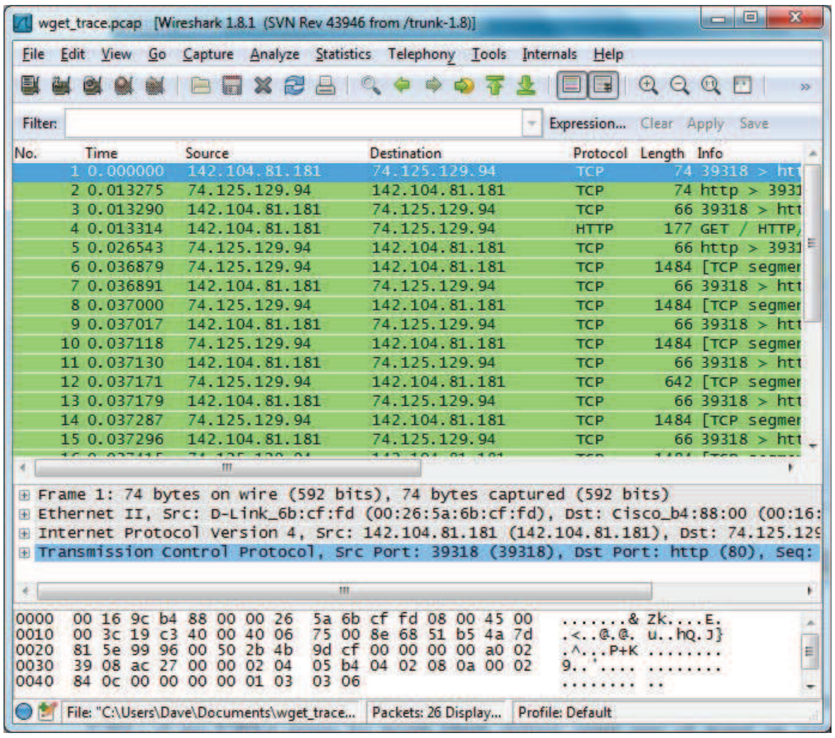
Name Resolution
☑ Enable MAC name resolution
☑ Enable network name resolution
☑ Enable transport name resolution

Figure 1.4: Capture options window

- Select an HTTP GET packet. This packet carries the HTTP request sent from your computer to the server.

- The protocol layers being used in web fetching are shown in Fig. 1.6. HTTP is the application layer web protocol used to fetch URLs. It runs on top of the TCP/IP transport and network layer protocols. The link layer protocol shown in the figure is Ethernet. It may be other protocol, depends on your network.

- Click on one HTTP packet, and turn to the middle panel with details of the packet. The first block is "Frame". This 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" (You may have taken trace in a computer with 802.11, but still you will see an Ethernet block. This is because Wireshark capture traffic in Ethernet format determined on the capture options. See Link-layer header type.). Then we can see IP, TCP, and HTTP. This is a bottom-up order, because as packets are passed down the protocol stack, the header of the lower layer protocol is added to the front of the information from


wget_trace.pcap [Wireshark 1.8.1 (SVN Rev 43946 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time      Source          Destination     Protocol Length Info
   1 0.000000  142.104.81.181  74.125.129.94   TCP        74 39318 > htt
   2 0.013275  74.125.129.94   142.104.81.181  TCP        74 http > 3931
   3 0.013290  142.104.81.181  74.125.129.94   TCP        66 39318 > htt
   4 0.013314  142.104.81.181  74.125.129.94   HTTP      177 GET / HTTP/
   5 0.026543  74.125.129.94   142.104.81.181  TCP        66 http > 3931
   6 0.036879  74.125.129.94   142.104.81.181  TCP      1484 [TCP segmer
   7 0.036891  142.104.81.181  74.125.129.94   TCP        66 39318 > htt
   8 0.037000  74.125.129.94   142.104.81.181  TCP      1484 [TCP segmer
   9 0.037017  142.104.81.181  74.125.129.94   TCP        66 39318 > htt
  10 0.037118  74.125.129.94   142.104.81.181  TCP      1484 [TCP segmer
  11 0.037130  142.104.81.181  74.125.129.94   TCP        66 39318 > htt
  12 0.037171  74.125.129.94   142.104.81.181  TCP       642 [TCP segmer
  13 0.037179  142.104.81.181  74.125.129.94   TCP        66 39318 > htt
  14 0.037287  74.125.129.94   142.104.81.181  TCP      1484 [TCP segmer
  15 0.037296  142.104.81.181  74.125.129.94   TCP        66 39318 > htt

⊞ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
⊞ Ethernet II, Src: D-Link_6b:cf:fd (00:26:5a:6b:cf:fd), Dst: Cisco_b4:88:00 (00:16:
⊞ Internet Protocol Version 4, Src: 142.104.81.181 (142.104.81.181), Dst: 74.125.12
⊞ Transmission Control Protocol, Src Port: 39318 (39318), Dst Port: http (80), Seq:

0000  00 16 9c b4 88 00 00 26  5a 6b cf fd 08 00 45 00   .......& Zk....E.
0010  00 3c 19 c3 40 00 40 06  75 00 8e 68 51 b5 4a 7d   .<..@.@. u..hQ.J}
0020  81 5e 99 96 00 50 2b 4b  9d cf 00 00 00 00 a0 02   .^...P+K ........
0030  39 08 ac 27 00 00 02 04  05 b4 04 02 08 0a 00 02   9..'.... ........
0040  84 0c 00 00 00 00 01 03  03 06                     ........ ..

File: "C:\Users\Dave\Documents\wget_trace...  Packets: 26 Display...  Profile: Default
```

Figure 1.5: Packet trace

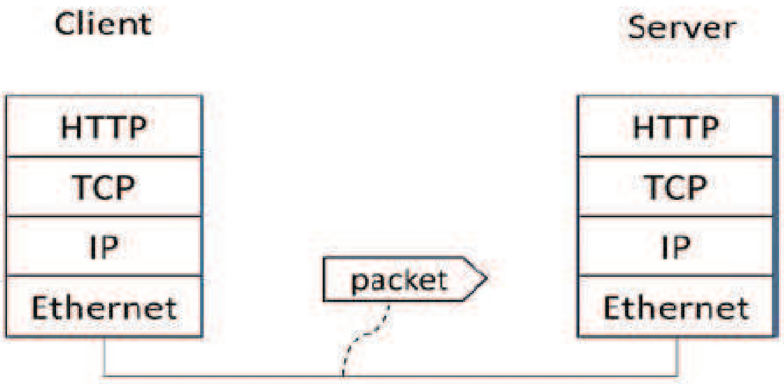


Figure 1.6: Protocol stack for a web fetch

the higher layer protocol. That is, the lower layer protocols come first in the packet.

- 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. In order to do this, the protocol use information in its header to determine the higher layer data unit encapsulated. Which field is used here?
- Draw a figure of an HTTP GET packet that shows the position and size in bytes of the TCP, IP and Ethernet protocol headers. On this drawing, show the range of header and payload of each layer.

1.3 Discussion

1.3.1 Running WireShark

1. Capture a trace without filter.
2. List at least 3 different protocols that appear in the protocol column in the *unfiltered* packet-listing window.
3. How long did it take from the HTTP GET message being sent to the HTTP OK reply being received?

1.3.2 Networking Tools

Explore the usage of "ifconfig", "ping", "netstat", and answer the following questions. (**Hint:** If you're not sure about how to use these commands, please check *Sec. 1.1.2 Networking Tools*".)

1. How many Ethernet interfaces are in your computer, how to determine it?
2. How to turn down/up an Ethernet interface?
3. Ping 10 packets to two websites. Compare the statistic results (packet loss, avg rtt).

1.3.3 Layered Protocol

1. Draw the structure of a HTTP GET packet.

2. In the provided trace (**lab1-wget-trace.pacp**), calculate the average overhead of **all** the packets **from the server to the client** (in percentage). (**Hint:** For one packet, the overhead is the size of all headers in one packet over the total size. The average overhead is the ratio between total size of headers and total size of the packets).

3. Which Ethernet header field tells the next higher layer protocol is IP? What value it used?

4. Which IP header field tells the next higher layer protocol is TCP? What value it used?

Chapter 2

Lab 2: Ethernet and IEEE 802.11

2.1 Objective

In this lab, we will investigate the link layer protocols, including the Ethernet and IEEE 802.11. The first part of this lab is mainly about the Ethernet frame format. The second part of the lab focuses on analyzing IEEE 802.11 frames.

2.2 Introduction

2.2.1 Ethernet

Ethernet stations communicate by sending each other data frames. As with other IEEE 802 LANs, each Ethernet station is given a single 48-bit MAC address, which is used to specify the destination and the source of each data frame. Network interface cards (NICs) or chips normally do not accept frames addressed to other Ethernet stations. Adapters are generally programmed with a globally unique MAC address, but this can be overridden, either to avoid an address change when an adapter is replaced, or to use locally administered addresses.

All generations of Ethernet (except very early experimental versions) share the same frame formats (and hence the same interface for higher layers), and can be readily (and in most cases, cheaply) interconnected.

Due to the ubiquity of Ethernet, the ever-decreasing hardware cost of it, and the reduced panel space needed by twisted pair Ethernet, most manufacturers now build the functionality of an Ethernet card directly into PC motherboards, eliminating the need for installation of a separate network card.

2.2.2 IEEE 802.11

In this part, we are going to explore the link layer, and management functions of 802.11. Generally speaking, there are three types 802.11 frames, the Data frame (Type 2), the Control frame (Type 1), and the Management frame (Type 0). For each type of frame, there are also different subtypes. Typically, Data frame is the longest, which can be up to 1500 bytes, while Management frames are much shorter, and Control frames are very short. As the Data and Control frames have been illustrated in the text book, here we introduce some important types of Management frames.

- **Beacon frame** Beacon frames are sent out periodically by an AP to advertise its existence and capabilities to nearby computers. Beacon is an IEEE 802.11 wireless LAN Management frame. In a Beacon frame, there are a series parameters, including the SSID name of the AP, the data rates it supports, and the channel on which it is operating.

- **Association** A computer has to associate with the AP after it learned an AP via a Beacon or otherwise and before it can send or receive data from the AP. Possibly, authentication process will be involved during the association. If the Association Request is successful received by AP, the AP will return an Association Response, and then the computer will acknowledge the association response. The Association Request and Response carry information that describes the capabilities of the AP and computer. Thus, both endpoints can know the other's abilities.

- **Probe Request/Response** In addition to find AP by waiting to learn about an AP from Beacons, a computer may also probe for specific APs. A Probe Request is sent by a computer to test whether an AP with a specific SSID is nearby. If the AP is nearby, it will reply with a Probe

Response. Like Beacon and Association frames, each of these frames carries information describing the capabilities of the computer and AP.

2.3 Procedures

2.3.1 Analyzing Ethernet frames

- Download and open the trace named "ethernet-trace-1".
- Find the HTTP GET message that was sent from the web browser to `gaia.cs.umass.edu` (should be packet No.10) and answer question (1)-(4) in section 2.4.1.
- Find the Ethernet frame containing the first byte of the HTTP response message and answer question (5)-(8) in section 2.4.1.

2.3.2 Exploring IEEE 802.11 functions

- Download and open the trace named "wlan-trace-1" [4]. Note that it may be difficult to gather your own trace using windows system. The main issue is that Windows system made 802.11 frames appear to come via a wired Ethernet. However, it is possible to use Mac or Linux to gather 802.11 frames directly, without this conversion.
- Select a Data packet. The packet detail can show four layers information: 1) Frame, which is a record added by Wireshark with information about the time and length of the frame; 2) Radiotap, which is also a record of captured physical layer parameters, such as the strength of the signal and the modulation; 3) IEEE 802.11, which is the bits of the 802.11 Data frame;4) Data, which is a record containing the frame payload data. Answer the related questions in section 2.4.2.
- Inspect different packets to see the values for different types of frames. You can use filter to see only one type frames by entering the expression wlan.fc.type==n into the Filter box above the list of frames in the top panel. For example, "n=2" is for data frames, "n=1" is for control frames, and "n=0" is for management frames. Answer the related questions in section 2.4.2.

- Inspect the packet transmission reliability. Use filter expressions to find the number of Data frames that are originals and retransmissions. For example, wlan.fc.type==2 && wlan.fc.retry==0 will find original Data frames. Answer the related questions in section 2.4.2.
- Inspect the Management frame. Use filter to help you find these frame, and answer the related questions in section 2.4.2.

2.4 Discussion

2.4.1 Analyzing Ethernet frames

For trace file "ethernet-trace-1", answer the following questions.

1. What is the 48-bit Ethernet address of the client computer?
2. What is the 48-bit destination address in the Ethernet frame? Is this the Ethernet address of `gaia.cs.umass.edu`? Which device has this as its Ethernet address?
3. Give the hexadecimal value for the two-byte Frame type field.
4. What is the value of the Ethernet source address? Is this the address of your computer, or of `gaia.cs.umass.edu` Which device has this as its Ethernet address?
5. What is the destination address in the Ethernet frame? Is this supposed to be the Ethernet address of the computer you are using?
6. Find the hexadecimal value for the two-byte Frame type field.

2.4.2 Exploring IEEE 802.11 functions

Answer the following questions based on the trace file "wlan-trace-1".

1. Which AP is the most active one (i.e., the one sends most Beacon messages)? what is its BSS ID?
2. How many Data frames are in the trace, how many subtypes, and what is the most common subtype of Data frame?

3. How many subtypes of Control frames are in the trace, what are they? and what is the most common subtype?

4. How many subtype of Management frames are in the trace, what are they and what is the most common subtype?

5. Please estimate the retransmission rate as the number of retransmissions (i.e., the total number of transmission - number of original frames) over the number of original transmissions. Show your calculation.

6. What are the Type and Subtype values for the Association Request/Association Response frames, the Probe Request/Probe Response frames?

Chapter 3

Lab 3: ARP, IP, and ICMP

3.1 Objective

In this lab, we will investigate the Address Resolution Protocol (ARP), the IP (Internet Protocol), and Internet Control Message Protocol (ICMP). The first part of this lab is mainly about the ARP protocol. We will study the operation of the protocol based on the fields that it sets in the Ethernet frame containing the ARP message. The second part of the lab focuses on analyzing IP frames, by observing and interpreting the fields in the IP frame. The last part of this lab focuses on the format and content of the ICMP messages.

3.2 Introduction

3.2.1 Address Resolution Protocol (ARP)

ARP is the standard method for finding a host's hardware address when only its network layer address is known. It can be used to resolve many different network-layer protocol addresses to hardware addresses. Due to the overwhelming prevalence of IPv4 and Ethernet, ARP is primarily used to translate IP addresses to Ethernet MAC addresses. ARP is used in the following four cases when two hosts communicate:

1. When two hosts are on the same network and one desires to send a packet to the other.

2. When two hosts are on different networks and must use a gateway/router to reach the other host.
3. When a router needs to forward a packet for one host through another router.
4. When a router needs to forward a packet from one host to the destination host on the same network.

The first case is used when two hosts are on the same physical network (that is, they can directly communicate without going through a router). The last three cases are the most used over the Internet as two computers on the Internet are typically separated by several hops.

3.2.2 Internet Protocol (IP)

Above the link layer, there is the network layer, which is responsible for relay the date between the transport layer and the link layer. The network protocol in network layer is called the Internet Protocol, or more commonly, the IP Protocol. The IP protocol performs two basic functions, the addressing (IP address) and the packet routing. Note that the internet layer is agnostic of application data structures as the transport layer, and it also does not distinguish between operation of the various transport layer protocols. Thus, IP protocol can carry data for a variety of different upper layer protocols by different protocol numbers, such as TCP, UDP and ICMP.

There are currently two versions of the IP protocol, IPv4 and IPv6. In this section we examine IPv4, which the most widespread version. With given trace files, we're going to learn about the details of IP frame.

3.2.3 Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) is one of the core protocols for network management in the Internet. It is mainly used by networked computers' operating systems to send error messages indicating, for instance, that a requested service is not available or that a host or router could not be reached. It has been used in network trouble-shooting and analyzing applications such as *ping* and *traceroute*.

ICMP uses the basic support of IP as if it were a higher level protocol; however, ICMP is actually an integral part of IP, and must be implemented by every IP module. ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route [RFC792].

In this part of the lab, we use two network tools. One is *ping*, which is used to test whether a particular host is reachable across an IP network, to self-test the network interface card of the computer, or to measure latency. The other one is *traceroute*, used to determine the route taken by packets across an IP network. We can understand the functions of ICMP by using these tools.

3.3 Procedures

3.3.1 Exploring ARP functions

- Download and open the trace named "ethernet-trace-1".
- This trace basically emulates retrieving a long document.
- The ARP protocol typically maintains a cache of IP-to-Ethernet address translation pairs on your computer.
- Find the ARP request message and answer questions (1)-(5) in section 3.4.1.
- Find the ARP reply that was sent in response to the ARP request and answer questions (6)-(10) in section 3.4.1.

3.3.2 Analyzing IP frames

- Using the same trace file as above.
- Select any packet with the HTTP GET message 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). And the 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.

- Select the the packet with HTTP GET message (the No.10 packet) and answer questions (1)-(2) in section 3.4.2.
- Observe all the packets and answer questions (3)-(4) in section 3.4.2.

3.3.3 Exploring ICMP functions

A. *Ping* The *ping* program in the source host sends a packet to the target IP address; if the target is alive, the *ping* program in the target host responds by sending a packet back to the source host. Both of these *ping* packets carry ICMP messages.

The following procedures describe how to capture the traces of ping messages.

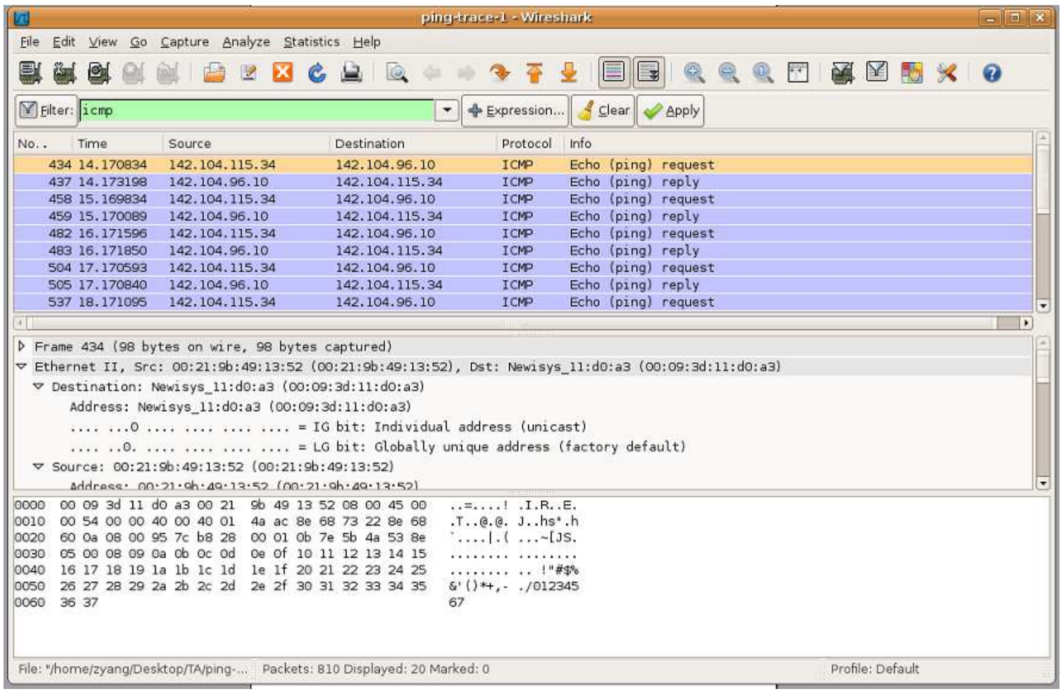
- Start up the WireShark and begin packet capture.
- Open a console, type the command *ping www.engr.uvic.ca -c 10*[1] in the command line. The argument "-c 10" indicates that ten ping messages should be sent.
- When the *ping* program terminates, stop the packet capture.

Download and open "ping-trace-1" in WireShark. Use the display filter to list the ICMP messages only, as shown in Figure 3.1 and answer questions (1)-(4) in section 3.4.3.

B. *Traceroute* The *traceroute* program is used to figure out the path a packet takes from the source to the destination. The following procedures describe how to capture the packets of traceroute messages.

- Start up the WireShark and begin packet capture.
- Open a console, type the command *traceroute www.engr.uvic.ca* in command line.

[1]The *ping* command here is different in Linux and Windows operating system. If you're working in Windows system, the command here should be *ping www.engr.uvic.ca -n 10*



```
No. .  Time       Source          Destination     Protocol  Info
  434 14.170834  142.104.115.34  142.104.96.10   ICMP      Echo (ping) request
  437 14.173198  142.104.96.10   142.104.115.34  ICMP      Echo (ping) reply
  458 15.169834  142.104.115.34  142.104.96.10   ICMP      Echo (ping) request
  459 15.170089  142.104.96.10   142.104.115.34  ICMP      Echo (ping) reply
  482 16.171596  142.104.115.34  142.104.96.10   ICMP      Echo (ping) request
  483 16.171850  142.104.96.10   142.104.115.34  ICMP      Echo (ping) reply
  504 17.170593  142.104.115.34  142.104.96.10   ICMP      Echo (ping) request
  505 17.170840  142.104.96.10   142.104.115.34  ICMP      Echo (ping) reply
  537 18.171095  142.104.115.34  142.104.96.10   ICMP      Echo (ping) request

▷ Frame 434 (98 bytes on wire, 98 bytes captured)
▽ Ethernet II, Src: 00:21:9b:49:13:52 (00:21:9b:49:13:52), Dst: Newisys_11:d0:a3 (00:09:3d:11:d0:a3)
  ▽ Destination: Newisys_11:d0:a3 (00:09:3d:11:d0:a3)
      Address: Newisys_11:d0:a3 (00:09:3d:11:d0:a3)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
  ▽ Source: 00:21:9b:49:13:52 (00:21:9b:49:13:52)
      Address: 00:21:9b:49:13:52 (00:21:9b:49:13:52)

0000  00 09 3d 11 d0 a3 00 21  9b 49 13 52 08 00 45 00   ..=....! .I.R..E.
0010  00 54 00 00 40 00 40 01  4a ac 8e 68 73 22 8e 68   .T..@.@. J..hs".h
0020  60 0a 08 00 95 7c b8 28  00 01 0b 7e 5b 4a 53 8e   `....|.( ...~[JS.
0030  05 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67

File: "/home/zyang/Desktop/TA/ping-...  Packets: 810 Displayed: 20 Marked: 0   Profile: Default
```

Figure 3.1: Capture of *ping* packet with ICMP display filter

- When the *traceroute* program terminates, stop the packet capture.

Download and open "tracert-trace-2" in WireShark, and set the display filter as *icmp*. Then answer the questions (5)-(7) in section 3.4.3 based on the trace.

3.4 Discussion

3.4.1 Exploring ARP functions

Answer the following questions based on the trace file "ethernet-trace-1".

1. What are the hexadecimal values for the source and destination addresses in the Ethernet frame containing the ARP request message?
2. Find the hexadecimal value for the two-byte Ethernet Frame type field.

3. Where the ARP opcode (operation code) field is located, i.e., how many bytes bits are there between of first bit of the opcode and the first bit of the ARP message?

4. What is the value of the opcode field within the ARP-payload part of the Ethernet frame, in which an ARP request is made?

5. Does the ARP message contain the IP address of the sender?

6. Where the ARP opcode field is located, i.e., how many bytes from the very beginning of the Ethernet frame does the ARP opcode field begin?

7. What is the value of the opcode field within the ARP-payload part of the Ethernet frame in which an ARP response is made?

8. What is the MAC address answered to the earlier ARP query?

9. What are the hexadecimal values for the source and destination addresses in the Ethernet frame containing the ARP reply message?

10. Why there is no ARP reply for the second ARP query (packet No.6)?

3.4.2 Analyzing IP frames

Answer following questions based on “ethernet-trace-1”.

1. Sketch a figure of **the packet you selected** to show the position and size in bytes of the IP header fields, as well as the values in hexadecimal. Your figure can simply show the frame as a long, thin rectangle.

2. What are the IP and MAC addresses of the source and destination?

3. How does the value of the Identification field change or stay the same for different packets? Is there any pattern if the value does change?

4. How can you tell from looking at a packet that it has not been fragmented?

3.4.3 Exploring ICMP functions

Answer following question based on “ping-trace-1” and “tracert-trace-2”.

1. What is the IP address of the source host (client)? What is the IP address of the destination host (server)?

2. Can you get the average RTT (Round Trip Time)? What’s that?

3. Examine one of the ping request packets. What are the ICMP type and code numbers? What other fields does this ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?

4. Examine the corresponding ping reply packet. What are the ICMP type and code numbers? What other fields does this ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?

5. What is the IP address of the source host (client)? What is the IP address of the destination host (server).

6. How many routers are between the source and the destination (*www.engr.uvic.ca*), for the trace file? Please draw a figure to show the sequences of these routers, i.e, source –> router_*first* –> ... –> router_*last* –> destination?

7. Can you get the average RTT times between the source host and each router? What are they?

Chapter 4

Lab 4: TCP

4.1 Objective

In this lab, we first get familiar with the format of TCP header, then study the TCP 3-Way Handshake and reliable data transfer, followed by the congestion control algorithm and retransmission scheme.

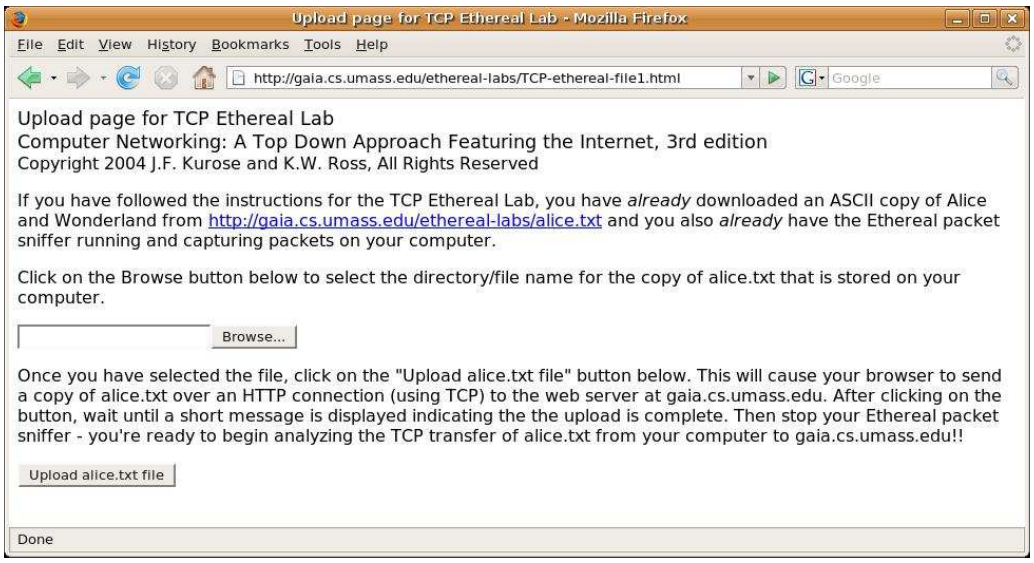
4.2 Introduction

TCP is the dominant transport layer protocol in the Internet. It provides a reliable, in-order stream of data between two end-points, even if they are connected by a network that may drop, re-order, or corrupt the packets. TCP provides the reliable data streaming service by detecting if packets are lost, delayed, or corrupted during transmission.

In this Lab and the following Lab, we investigate the behaviour of TCP in detail, by analysing a trace of the TCP segments sent and received in transferring a 300 KB file from a local computer (the client, IP address: 10.0.1.5) to a remote Web server (`http://gaia.cs.umass.edu/`, IP address: 128.119.245.12). The file, named alice.txt (which contains two copies of the text of *Alice in Wonderland*) is stored on the client computer and is uploaded to the server using the HTTP POST method. Here the POST method is used in order to transfer a large amount of data from a computer to another computer.

The procedure to transfer this file is as follows:

- Start up Web browser on the client computer and go to `http://gaia.cs.umass.edu/ethereal-labs/TCP-ethereal-file1.html`. The screen looks like Figure 4.1.


Upload page for TCP Ethereal Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://gaia.cs.umass.edu/ethereal-labs/TCP-ethereal-file1.html
Google
Browse...
Upload alice.txt file
Done

Upload page for TCP Ethereal Lab
Computer Networking: A Top Down Approach Featuring the Internet, 3rd edition
Copyright 2004 J.F. Kurose and K.W. Ross, All Rights Reserved

If you have followed the instructions for the TCP Ethereal Lab, you have *already* downloaded an ASCII copy of Alice and Wonderland from http://gaia.cs.umass.edu/ethereal-labs/alice.txt and you also *already* have the Ethereal packet sniffer running and capturing packets on your computer.

Click on the Browse button below to select the directory/file name for the copy of alice.txt that is stored on your computer.

Once you have selected the file, click on the "Upload alice.txt file" button below. This will cause your browser to send a copy of alice.txt over an HTTP connection (using TCP) to the web server at gaia.cs.umass.edu. After clicking on the button, wait until a short message is displayed indicating the the upload is complete. Then stop your Ethereal packet sniffer - you're ready to begin analyzing the TCP transfer of alice.txt from your computer to gaia.cs.umass.edu!!

Figure 4.1: Upload page

- Use the *Browse* button to enter the full path name of alice.txt on the client computer, and then press the *Upload alice.txt file* button to upload the file to the server `gaia.cs.umass.edu`.

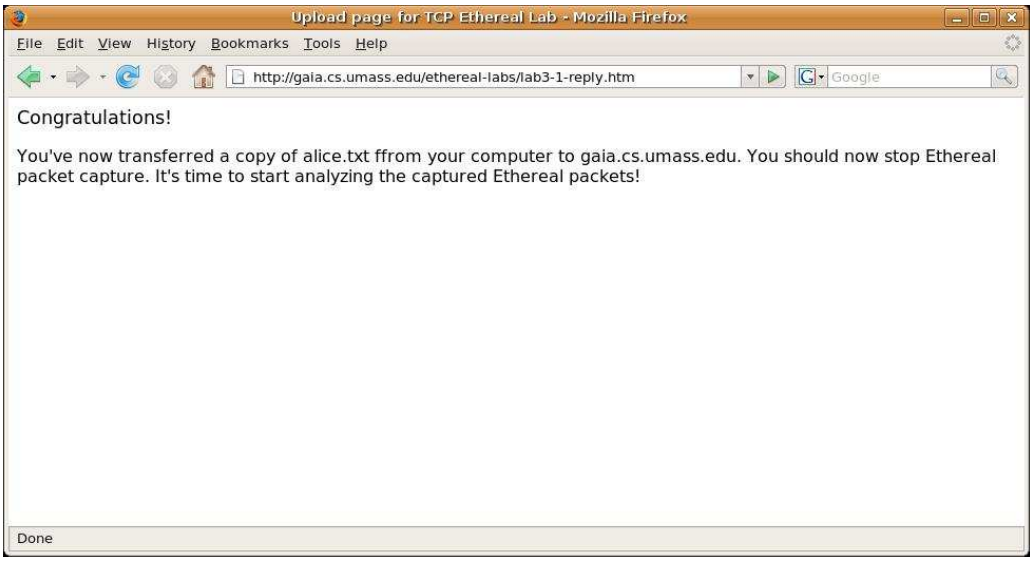
- Once the file has been uploaded, a new web page, which is a short congratulation, will be transferred from the Web server to the client and displayed in the web browser, as shown in Figure 4.2.

To transfer alice.txt and the congratulation page without error, a TCP connection between the client and the server is established. The TCP connection completes the four operations in this real-world application as follows:

- TCP connection setup.

- Transfer the HTTP POST command and the file alice.txt, from the client computer to the server `gaia.cs.umass.edu`.

- Transfer the congratulation page from the server to the client.


Upload page for TCP Ethereal Lab - Mozilla Firefox

File Edit View History Bookmarks Tools Help
http://gaia.cs.umass.edu/ethereal-labs/lab3-1-reply.htm
Google
Done

Congratulations!

You've now transferred a copy of alice.txt ffrom your computer to gaia.cs.umass.edu. You should now stop Ethereal packet capture. It's time to start analyzing the captured Ethereal packets!

Figure 4.2: Congratulation Page

- TCP connection release.

WireShark is run on the client computer to capture the trace of the TCP segments sent/received from/to the client computer while the file is being transferred. We saved the trace from the TCP stream in the file **tcp-trace-1.cap**. The trace tracked all of the above four actions of TCP. We use this trace to study the TCP behaviours in this lab.

4.2.1 TCP Header Format

Every TCP segment consists of a header followed by an optional data portion. The format of the header is defined in RFC 793, including Source Port (16 bits), Destination Port (16 bits), Sequence Number (32 bits), ACK (32 bits), ... In this lab, we will see the details of the TCP headers used in this application.

4.2.2 TCP Connection Setup

Before transferring data, a TCP connection is established between the two end systems, typically with three messages, called the three-way handshake:

SYN $\rightarrow$ SYNACK $\rightarrow$ ACK. The handshake is also used to negotiate certain properties of the connection, e.g., the Maximum Segment Size (MSS) that the client and server can accept, and the Selective Acknowledgement (SACK) option. In this lab, we will see the three-way handshake procedure in the trace **tcp-trace-1.cap**.

4.2.3 TCP Data Flow

Once the connection is established, the TCP sender partitions the message from the application into segments. The MSS is used to determine how to partition the single message so the underlying network can encapsulate each segment into a packet without further fragmentation. The sequence number and ACK number are used to detect packet loss, duplication, re-order in transmission and to deliver the segments correctly and in-order to the application in the destination host.

In this real-world application, after the connection was established, the client computer wrote about 300KB into the data stream using the HTTP POST command. From the application's perspective, this was sent as one unit, or one message. However, the underlying network cannot support packets large enough to hold all 300KB of data. We will see that TCP broke this single logical transmission into multiple segments according to MSS.

In the trace file **tcp-trace-1.cap**, the first three segments are used to establish the connection. Starting from the No.4 TCP segment, the client began to transfer the application layer message to the server. The 4th segment contains the HTTP POST command (we will dig into the packet content field and see this HTTP command). This segment is actually used to transfer this HTTP command. The text file is transferred by the following TCP segments. Here we regard both the HTTP POST command and the file (alice.txt) together as the whole message. Therefore, we consider the 4th TCP segment as the first segment in the TCP connection to transfer the message from the client to the server.

4.2.4 TCP Connection Release

The TCP connection is closed when the two end systems exchange TCP segments with FIN bit set and ACKed by the other side. The FIN bit literally means that no additional new data will be sent on that side of the connection.

The sequence of two FINs and their corresponding ACKs is the preferred way to gracefully terminate a TCP connection. However, TCP connections can also be terminated by setting the RESET bit. Although the RESET was designed to be used for unrecoverable errors, it is often used in practice for fast termination that avoids the formalities of the FIN-ACK exchanges.

In the trace file **tcp-trace-1.cap**, after the client acknowledged the data of the congratulation page, the server sent a FIN indicating that it would not be sending any additional data. The client acknowledged this FIN by sending back the ACK. Therefore the flow in the direction from the serve to the client is closed. The client computer could also terminate its flow to the server by sending the FIN segments. Alternatively, the client computer sent a RESET segment to the server to release the connection.

4.2.5 TCP Congestion Control (Optional)

In TCP, congestion control provides the ability to limit the sending rate in response to signs of network congestion. Congestion control helps the network to recover from congestion by shrinking sender's outgoing traffic and therefore avoids network congestion collapse, and at the same time tries to achieve throughput as high as possible.

Congestion control is realized by setting the size of congestion window, according to two strategies, the slow start and the congestion avoidance. During the slow start phase, the congestion window increases one MSS with each acknowledgement, and subsequently the window size is doubled every round-trip time (RTT). During congestion avoidance, each acknowledgement increases the congestion window by $MSS^2/congestion\ window\ size$ (if the receiver sends ACK for each received packet without delay), and subsequently the congestion window size is increased by one MSS every RTT. Slow start phase changes to congestion avoidance phase when congestion window exceeds the slow-start threshold.

We will use the TCP segment trace file, **tcp-trace-1.cap** to investigate TCP congestion control. In particular, we look at how the congestion window evolved since the beginning of transferring the HTTP POST command and the alice.txt file.

4.2.6 TCP Flow Control (Optional)

TCP also provides flow control or the ability to limit the sending rate to avoid a fast sender over-running a slow receiver. To provide a reliable service, a TCP receiver cannot deliver data that it received out of order to the waiting application. Therefore, the TCP receiver typically allocates a fixed amount of buffer space to store both out-of-order data and data waiting for the application to fetch. If the TCP receiver runs out of buffer space to hold the incoming data, then it has no choice but to drop the out-of-order data packet even if it is error-free.

The receiver advertises its available buffer in each acknowledgement. The receiver's advertised window field is used to inform the sender how much room is left for the incoming data. Then in the sliding-window based flow control, the sender chooses the minimum of the receiver window and the congestion window to be the size of the sliding window in order to make sure that the receiver will not run out of buffer space.

In this subsection, we will still use the TCP segment trace file, **tcp-trace-1.cap**, to exam TCP flow control. We will see that the receiver window took effect and throttled the sender even though the congestion window continued to grow.

4.2.7 Retransmission in TCP

We learned that TCP provides reliable data transmission over an unreliable network by relying on feedback from the receiver to detect loss and responding to packet loss with retransmissions. TCP uses two kinds of indications of packet losses, time-out and duplicated acknowledgement (which is regarded as an early indication of packet loss and causes the fast retransmission instead of waiting until timeout). The TCP sender must maintain a copy of the data it sent in case retransmission is needed, so it must store the data until the corresponding acknowledgement is received.

However, in the trace **tcp-trace-1.cap**, all the packets were received correctly the first time and thus there was no retransmission. In order to investigate the TCP retransmission scheme, we are going to analyse another trace of TCP connections, **tcp-trace-retransmission.cap** [3], in which retransmission does occur.

The trace, **tcp-trace-retransmission.cap**, was taken on a private network [3]. A desktop PC and a laptop were connected via a wireless router.

The laptop was connected via a wireless interface and specifically placed so as to interfere with a strong signal. The IP addresses of the desktop and the laptop are, respectively, 192.168.0.100 and 192.168.0.102. The desktop sent a file (about 40K bytes) to the laptop using TCP. The TCP port number for the desktop is 4480, and 5001 for the laptop. The experiment configuration is shown in Figure 4.3. WireShark was run on the sender, i.e., the desktop, while the file was being transferred to capture the TCP segments exchanged. The TCP connection trace was saved in file **tcp-trace-retransmission.cap**.

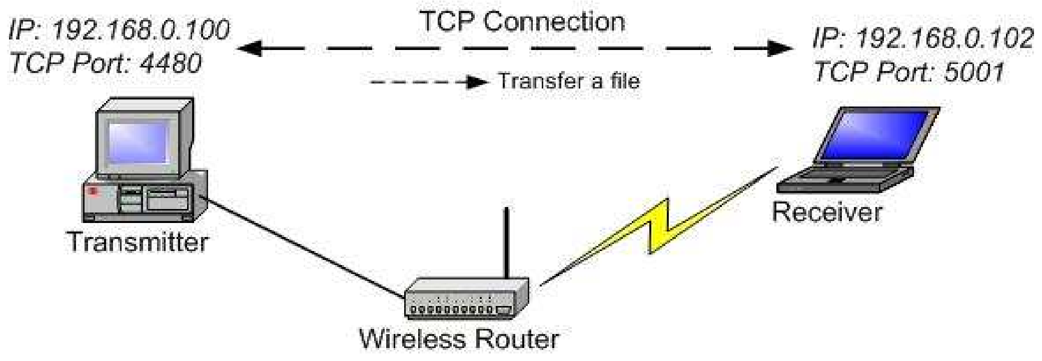


Figure 4.3: Network Configuration

In this lab, we will take a look at both the fast retransmission and the time out retransmission using this trace file.

4.3 Procedures

Note: You will answer a set of questions by exploring the trace file **tcp-trace-1.cap** and **tcp.analysis.retransmission.cap**. Whenever possible, when answering a question you should provide the information of the packet(s) within the trace that you used to answer the question asked. The information includes the Packet No., the name(s) and value(s) of the packet field(s) that you use to answer the questions.

4.3.1 TCP Header Format

- Download the traces folder from the lab website.

- Open the captured trace file named **tcp-trace-1.cap** with WireShark. Now what you should see is a series of TCP segments sent between the client and the server `gaia.cs.umass.edu`.
- Since this lab is about TCP rather than HTTP, change WireShark's *Packet List Pane* window so that it shows information about the TCP segments containing the HTTP messages. To do this, in WireShark, select *Analyze* ⇒ *Enabled Protocols*. Then uncheck the *HTTP box* and select *OK*.
- Select the first packet and explore the details of the TCP segment using the *packet details pane* and the *packet bytes pane*.
- Select the *Transmission Control Protocol* item in the *Packet Details Pane* then the content of the header is highlighted in the *Packet Bytes Pane*.
- Answer the related questions in section 4.4.1.

4.3.2 TCP Connection Setup

- Find the initial three-way handshake in the trace file. (Hint: You should see the SYN segment sent from the client to `gaia.cs.umass.edu`, and also the SYNACK segment being returned.)
- Answer the related questions in section 4.4.2.

4.3.3 TCP Data Flow

- Check the HTTP POST command. Select the 4th segment in the *Packet List Pane*. Select the *Data* item in the *Packet Details Pane* and the content of the data carried by this segment is highlighted in the *Packet Bytes Pane*. You should find a POST and other HTTP command information within its *Date* field.
- Set time reference. In order to make the following analysis easier, set time reference to the 4th packet. Choose the *Time Reference* items in the *Edit* menu, or from the pop-up menu of the *Packet List Pane*.

Note: Now the 4th packet becomes the starting point for all subsequent packets. The time values of all the following packets are calculated relative to this packet.

- Set the time display format as microseconds. Choose the *Time Display Format* in the *View* menu. Then select *Seconds Since Beginning of Capture* and *Microseconds.*
- Answer the related questions in section 4.4.3.

4.3.4 TCP Connection Release

- Find the segments used to release the connection between the client and the server.
- Answer the related questions in section 4.4.4.

4.3.5 TCP Congestion Control

- Download the HTTP traces folder from the lab website.
- Open the captured trace file named **tcp-trace-1.cap** with WireShark.
- Since this lab is about TCP rather than HTTP, change WireShark's *Packet List Pane* window so that it shows information about the TCP segments containing the HTTP messages. To do this, select *Analyze* $\Rightarrow$ *Enabled Protocols.* Then uncheck the *HTTP box* and select *OK.*
- Set time reference. In order to make the following analysis easier, set time reference to the 4th packet. Choose the *Time Reference* items in the *Edit* menu, or from the pop-up menu of the *Packet List Pane.*
- Answer the related questions in section 4.4.5.

4.3.6 TCP Flow Control

- Open the captured trace file named **tcp-trace-1.cap** with WireShark.
- Answer the related questions in section 4.4.6.

4.3.7 Retransmission in TCP (Optional)

- Open the captured trace file named **tcp-trace-retransmission.cap** with WireShark.
- List retransmissions. Search for retransmissions with the display filter *tcp.analysis.retransmission*. Applying this filter, you should see 9 retransmissions in the trace.
- Answer the related questions in section 4.4.7.

4.4 Discussion

4.4.1 TCP Header Format

1. Write down the TCP header content in hexadecimal format (in the *packet bytes pane*). Dissect the TCP header and indicate the value of each field in the header. Annotate the hexadecimal content to explain your answer.
2. What are TCP port numbers used by the client computer (source) and the server (destination) when transferring the file to `gaia.cs.umass.edu`? How did the client computer determine the port numbers when it wanted to set up a TCP connection to the server?
3. What is the maximum header length? Given the value of the Header Length field, how to calculate the length of the head in unit of bytes? Verify your answer using the first TCP segment in the trace file.
4. (Optional) How does TCP calculate the Checksum field? What is the pseudo-header format? Write down the pseudo-header of the flow from the client to the server in hexadecimal format. Verify the Checksum value in the first TCP segment in the trace file.

4.4.2 TCP Connection Setup

1. Which segments are the initial three-way handshake in the trace file? How do you determine this?

2. What is the actual initial sequence number in each direction (in hexadecimal format)? How did the client and the server determine these values?

   Note: WireShark displays the relative sequence number. You should select the *Sequence Number* field in the header, the actual value is highlighted in the *Packet Bytes Pane*.

3. What is the value of the acknowledgement number in the SYNACK segment? How did `gaia.cs.umass.edu` determine that value?

4. What are the values of the sequence number and the acknowledgement number in the third ACK segments in the three-way handshake? How did the client computer determine these values?

5. How did the client and the server announce the maximum TCP payload size that they were willing to accept? What are the values and why did they choose these values?

6. Is there data sent in the SYN, SYNACK, and ACK segment? How do you determine this?

4.4.3 TCP Data Flow

1. Beginning with the 4th segment, what are the sequence number, acknowledgement number, data length, and the time of the segment sent/received from/to the client computer of the 4th, 5th, 6th, ..., 15th segments in the TCP connection? Fill out Table 4.1 for the data flow from the client computer to the server. (**Note:** list both the actual value and relative value of the sequence number and acknowledgement number.)

2. What are the segments acknowledged by the packet 6, 9, 12, and 15, respectively? (**Hint:** acknowledgement number is the next byte expected, so it actually acknowledges the bytes before the acknowledgement number.)

3. Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the segments which have been acknowledged before the 15th segment?

Packet No.	Data Segments 10.0.1.5 --> 128.119.245.12				ACK Segments 128.119.245.12 --> 10.0.1.5		
	Seq. No./ Relative Seq. No.	Data Length	Time (s)		Ack. No./ Relative Ack. No.	Data Length	Time (s)
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							

Table 4.1: TCP segment exchange table (Please show the segment and its acknowledgement in the same row.)

4. (Optional) What is the Estimated RTT value after the receipt of each ACK? Assume that the value of the Estimated RTT is equal to the measured RTT for the first segment, and then is computed using the Estimated RTT equation for all subsequent segments. (**Hint:** Compare your calculation with the statistics analysis of TCP stream by WireShark. See Section 3.5).

5. In the trace file, how did the sequence number of the packets from the server to the client change? Why? (**Hint**: When transferring the alice.txt file, the server was only a receiver and did not send any data to the client.)

6. (Optional) At the end of the trace file, find the TCP segments used by the server to transfer the congratulation web page to the client computer? How do you determine this?

7. (Optional) Are there any retransmitted segments in the trace file? What do you check for (in the trace) in order to answer this question?

4.4.4 TCP Connection Release

1. Which packets were used to close the data flow from the server to the client? How do you determine this? (**Hint:** two segments are involved in the FIN-ACK sequence.)

2. Which packets were used to close the data flow from the client to the server? How do you determine this?

3. (Optional) In the FIN segment, what is the sequence number? In the corresponding ACK segment, what is the acknowledgement number? How did the client determine this number?

4.4.5 TCP Congestion Control

1. Exam the 4th to 15th TCP segments and take a reference to the Table in Question 1 of Section 4.4.3 in Lab 2. Can you find a pattern of the number of segments sent from the client and from the server `gaia.cs.umass.edu`? Why did the TCP data flow have such a pattern?

2. What is the initial size of congestion window? How do you determine this? What is the size of congestion window when the segment 5, 8, 11 and 14 were sent out?

3. In the lecture we have learned that the congestion window doubles its size in every RTT in the slow start phase. Beginning with the 4th packet, what is the size of the congestion window and which packet were inside the congestion window (i.e., these packets could be sent) during the first RTT? What is the size of the congestion window and which packet were inside the congestion window during the second RTT? How about the third RTT? Give the segment numbers.

4. When did the sender's congestion control change from the slow start phase to the congestion avoidance phase? Give the segment number and the time. How do you determine this?

5. What is the threshold between the slow start and congestion avoidance? (**Hint:** the size of congestion window when TCP change from slow start phase to congestion avoidance phase.)

4.4.6 TCP Flow Control

1. Exam the 179th segment in the trace file, why did the sender stop sending more segments? What is the size of receiver window advertised by the receiver at this moment? How do you determine this?

4.4.7 Retransmission in TCP (Optional)

1. Segment 12 is the first retransmission. What is it in the segment that identifies the segment as a retransmission? (Hint: the sequence number has been used by a previous packet.) Which segment was segment 12 retransmitted for?

2. Segment 12 is a fast retransmission, which should be triggered by triple-duplicated-acknowledgment. Find the three acknowledgments which triggered the fast retransmission of segment12. (Hint: in order to trigger a fast retransmission, the duplicated acknowledgments should acknowledge the same acknowledgment number, which is the sequence number of the fast retransmission.)

3. Is segment 44 a fast retransmission or timeout retransmission? How do you determine this? (Hint: whether the sequence number in the segment has been acknowledged for three times or not.)

Bibliography

[1] Ethereal Labs, http://www-net.cs.umass.edu/ethereal-labs

[2] Wikipedia.org, http://en.wikipedia.org/wiki/HTTP

[3] Jeanna Matthews, Computer Networking: Internet Protocols in Action, John Wiley & Sons, Inc., Dec. 2004.

[4] Andrew Tanenbaum and David Wetherall, Computer Networks 5/E, Prentice Hall, Oct. 2010

[5] James F. Kurose and Keith W. Ross. 2009. Computer Networking: A Top-Down Approach (5th ed.). Addison-Wesley Publishing Company, USA.