Department of Electrical and Computer Engineering

Laboratory Manual
for
CENG460 Communications Networks

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

Copyright 2017 University of Victoria. All rights reserved
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 and Discussions, Ethernet ......................................................... 14
   2.4 Procedures and Discussions, IEEE 802.11 .................................................. 14
   2.4.1 Discussion .............................................................................................. 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
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.3</td>
<td>Internet Control Message Protocol (ICMP)</td>
<td>18</td>
</tr>
<tr>
<td>3.3</td>
<td>Procedures and Discussions, ARP</td>
<td>19</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Exploring ARP Functions</td>
<td>19</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Discussions</td>
<td>19</td>
</tr>
<tr>
<td>3.4</td>
<td>Procedures and Discussions, IP</td>
<td>20</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Analyzing IP frames</td>
<td>20</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Discussions</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>Procedures and Discussions, ICMP</td>
<td>21</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Exploring ICMP Functions</td>
<td>21</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Discussions</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>Lab 4: TCP</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Objective</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>Introduction</td>
<td>24</td>
</tr>
<tr>
<td>4.2.1</td>
<td>TCP Header Format</td>
<td>26</td>
</tr>
<tr>
<td>4.2.2</td>
<td>TCP Connection Setup</td>
<td>26</td>
</tr>
<tr>
<td>4.2.3</td>
<td>TCP Data Flow</td>
<td>27</td>
</tr>
<tr>
<td>4.2.4</td>
<td>TCP Connection Release</td>
<td>27</td>
</tr>
<tr>
<td>4.2.5</td>
<td>TCP Congestion Control</td>
<td>28</td>
</tr>
<tr>
<td>4.2.6</td>
<td>TCP Flow Control</td>
<td>28</td>
</tr>
<tr>
<td>4.2.7</td>
<td>Retransmission in TCP</td>
<td>29</td>
</tr>
<tr>
<td>4.3</td>
<td>Procedures and Discussions</td>
<td>30</td>
</tr>
<tr>
<td>4.3.1</td>
<td>TCP Header Format</td>
<td>30</td>
</tr>
<tr>
<td>4.3.2</td>
<td>TCP Connection Setup</td>
<td>31</td>
</tr>
<tr>
<td>4.3.3</td>
<td>TCP Data Flow</td>
<td>32</td>
</tr>
<tr>
<td>4.3.4</td>
<td>TCP Connection Release</td>
<td>34</td>
</tr>
<tr>
<td>4.3.5</td>
<td>TCP Congestion Control</td>
<td>35</td>
</tr>
<tr>
<td>4.3.6</td>
<td>TCP Flow Control</td>
<td>36</td>
</tr>
<tr>
<td>4.3.7</td>
<td>Retransmission in TCP (Optional)</td>
<td>36</td>
</tr>
</tbody>
</table>
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 analyzers. 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.

![Network protocol analyzer structure](image)

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 on 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, so it 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**

*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**

*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* gives you the control on 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 as follows.

- **Filter field**: It is used to filter out uninterested packets with the entered specifications, so you can choose which packets should (not) be shown on the screen.

- **Captured packets**: It lists the packets captured by the selected interface.

- **Details of selected packet**: It lists the information about the packet that is selected in the captured packets window.

- **Content of packet in hex/ASCII**: It displays the content of the captured packet, in hex and ASCII.
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”, and 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(s). Closing the browser(s) can 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
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

Inspect the captured trace or the provided trace (lab1-wget-trace.pacp) to understand the layered protocol.
• 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 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 it has. 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 captures traffic in Ethernet format. See Link-layer header type.). Then we can see IP, TCP, and HTTP. This is in 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 the higher layer protocol. That is,
the header from the lower layer protocols comes earlier 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 uses 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, and how to determine it?

2. How to turn down/up an Ethernet interface?

3. Ping 10 packets to two websites. Compare the statistic results (i.e., the packet loss rate and average round-trip time).
1.3.3 Layered Protocol

1. Draw the structure of an 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 over the total packet size. The average overhead is the ratio of the sum of header sizes over the sum of packet sizes).

3. Which bytes in the Ethernet header field tell that the next higher layer protocol is IP? What is the hexadecimal value of this field?

4. Which bytes in the IP header field tell that the next higher layer protocol is TCP? What is the hexadecimal value of this field?
2.1 Objective

In this lab, we investigate the link layer protocols, including those for Ethernet and IEEE 802.11 networks. The first part of this lab is mainly about the Ethernet frames, and the second part 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) 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 the very early experimental versions) share the same frame formats (and hence the same interface for higher lay-
ers), and can be readily (and in most cases, cheaply) interconnected.

Due to the ubiquity of Ethernet and the ever-decreasing hardware cost of it, most manufacturers now build the functionality of an Ethernet card directly into PC motherboards, eliminating the need for installing 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 IEEE 802.11 standard. Generally speaking, there are three types of IEEE 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 frames are the largest, 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 textbook, 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 wireless stations (e.g., laptops, PCs, or handheld devices). Beacon is an IEEE 802.11 wireless LAN Management frame. In a Beacon frame, there are a series of parameters, including the SSID name of the AP, the data rates it supports, and the channel on which it is operating.

- **Association** A wireless station has to associate with the AP after it learns an AP via a Beacon frame, before it can send or receive data to or from the AP. Possibly, an authentication process will be involved during the association. If the Association Request is successfully received by the AP, the AP will return an Association Response, and then the station will acknowledge the association response. The Association Request and Response carry information that describes the capabilities of the station or the AP.

- **Probe Request/Response** In addition to finding an AP by waiting for Beacon frames, a station may also probe for specific APs. A Probe Request is sent by a station to test whether an AP with a specific SSID is nearby. If the AP is nearby, it will reply with a Probe Response.
Similar to the Beacon and Association frames, each of these frames carries information describing the capabilities of the station or the AP.

### 2.3 Procedures and Discussions, Ethernet

- 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 the following questions.

1. What is the 48-bit MAC address of the client computer?
2. What is the 48-bit destination MAC address in the Ethernet frame? Is this the MAC address of gaia.cs.umass.edu? Which device has this MAC address?
3. Give the hexadecimal value for the two-byte Frame type field.
4. What is the value of the source MAC address? Is this the address of your computer, or of gaia.cs.umass.edu? Which device has this MAC address?
- Find the Ethernet frame containing the first byte of the HTTP response message and answer the following questions.

5. What is the destination MAC address in the Ethernet frame? Is this the Ethernet address of the computer that you are using?
6. Find the hexadecimal value for the two-byte Frame type field.

### 2.4 Procedures and Discussions, IEEE 802.11

- 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.
- 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.

- 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.

- Inspect the packet transmission reliability. Use filter expressions to find the number of Data frames that are originals and retransmissions, respectively. For example, wlan.fc.type==2 && wlan.fc.retry==0 will find the original Data frames.

- Inspect the Management frame. Use filter to help you find these frame.

### 2.4.1 Discussion

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

7. Which AP is the most active one (i.e., the one sent most Beacon messages)? What is its BSS ID?

8. How many Data frames are in the trace, how many subtypes they have, and what is the most frequently appeared subtype of these Data frames?

9. How many subtypes of Control frames are in the trace, what are they, and what is the most frequently appeared subtype?

10. How many subtypes of Management frames are in the trace, what are they, and what is the most frequently appeared subtype?

11. Estimate the retransmission times, i.e., the number of retransmissions (i.e., the total number of transmissions - number of original frames) over the number of original transmissions. Show your calculation.
12. What are the Type and Subtype values for the Association Request/Association Response frames, and Probe Request/Probe Response frames, respectively?
Chapter 3

Lab 3: ARP, IP, and ICMP

3.1 Objective

In this lab, we investigate the Address Resolution Protocol (ARP), Internet Protocol (IP), and Internet Control Message Protocol (ICMP). The first part of this lab is mainly about the ARP protocol. We study the operation of the protocol based on the header fields in the Ethernet frames which contain the ARP message. The second part of the lab focuses on analyzing IP frames, by observing and interpreting the fields in the IP header. 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 the mapping between network-layer protocol addresses and the hardware addresses. Due to the popularity of IPv4 and Ethernet, ARP is most often used to translate IP addresses to Ethernet MAC addresses. ARP is used in the following four cases when two hosts communicate.

1. Two hosts are on the same network and one desires to send a packet to the other.
2. Two hosts are on different networks and one must use a gateway/router to reach the other host.

3. A router needs to forward a packet for one host through another router.

4. A router needs to forward a packet from one host to the destination host in 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 other three cases are the most widely used, as two computers in the Internet are typically separated by several hops.

### 3.2.2 Internet Protocol (IP)

The network layer is responsible for relaying packets over multiple hops from the source to the destination. The network layer protocol used in the Internet is called the Internet Protocol, or more commonly, the IP Protocol. The IP protocol performs two basic functions, addressing (with IP address) and routing. Note that the IP protocol does not distinguish the operations of various transport layer protocols and applications. Thus, it can carry data for a variety of upper layer protocols, such as TCP, UDP and ICMP.

Currently, there are two versions of IP protocols, IPv4 and IPv6. In this section, we examine IPv4, the most widely used version. With the given trace files, we learn the details of IP packets.

### 3.2.3 Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) is a core protocol for network management in the Internet. It is mainly used by networked devices’ 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 to deliver messages as if it were a higher level protocol; however, ICMP is actually an integral part of the network layer, and must be implemented by every IP module. ICMP messages are
sent in several situations: for example, when a datagram cannot reach its destination, the gateway does not have the buffering capacity to forward a datagram, or 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 or not 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 and Discussions, ARP

3.3.1 Exploring ARP Functions

- Download and open the trace named “ethernet-trace-1”.
- This trace was captured when a host retrieved a long document.
- The ARP protocol typically maintains a cache of IP-to-Ethernet address translation pairs.
- Find the ARP request message and answer questions 1-5 in Section 3.3.2.
- Find the ARP reply that was sent in response to the ARP request and answer questions 6-10 in Section 3.3.2.

3.3.2 Discussions

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 are there between the 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 are there between the first bit of the opcode and the first bit of the ARP message?

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 (in packet No. 6)?

3.4 Procedures and Discussions, IP

3.4.1 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 see the details of its structure (in the middle panel) and the bytes that make up the packet (in the bottom panel). Here, we focus on the IP header, and you may ignore the other higher and lower layer protocol headers.

- Select the packet with HTTP GET message (packet No.10) and answer questions 1-2 in Section 3.4.2.

- Observe all the packets and answer questions 3-4 in Section 3.4.2.
3.4.2 Discussions

Answer the 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 their hexadecimal values.

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

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 to judge whether a packet has been fragmented or not?

3.5 Procedures and Discussions, ICMP

3.5.1 Exploring ICMP Functions

**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 and type the command `ping www.engr.uvic.ca -c 10` 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 3.5.2.

---

1The 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`
Traceroute

The *traceroute* program is used to figure out the path that 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 and type the command `traceroute www.engr.uvic.ca` in command line.
- 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 5-8 in Section 3.5.2 based on the trace.
3.5.2 Discussions

Answer the following questions based on “ping-trace-1” and “tracert-trace-2”, respectively.

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

2. How long is the average Round-Trip Time (RTT)?

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 in 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 in the checksum, sequence number and identifier fields?

5. Examine the ICMP error packet, which could be found in the packets from tracert-trace-2. It has more fields than the ICMP echo packet. What are included in those fields? Find the TTL field, and explain what it is.

6. How many routers are between the source and the destination (www.engr.uvic.ca) from the trace file? Please draw a figure to show the sequences of these routers, i.e, source -> router_first -> ... -> router_last -> destination.

7. How long are the average RTTs between the source host and each router?
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 streaming service between two end-points, even if they reside in a network that may drop, re-order, or corrupt packets. TCP provides the reliable data streaming service by detecting if packets are lost, delayed, or corrupted during transmission.

In this lab, we investigate the behaviours of TCP in detail, by analyzing the 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/, with 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](http://gaia.cs.umass.edu/ethereal-labs/TCP-ethereal-file1.html)

**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.
WireShark is run on the client computer to capture the trace of the TCP segments sent/received to/from the client computer while the file is being transferred. The trace from the TCP stream is saved 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.

### 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), etc.

### 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 → SYN/ACK → 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 whether the Selective Acknowledgement (SACK) option is acceptable by both sides. 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 to avoid 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 message 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 server
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

In TCP, congestion control provides the ability to limit the sending rate
in response to signals 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 $\frac{MSS^2}{\text{congestion window size}}$ (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 ex-
ceeds the slow-start threshold.

We 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 from the beginning of transferring the HTTP POST command to
finishing the downloading of the alice.txt file.

4.2.6 TCP Flow Control

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.

We still use the TCP segment trace file, tcp-trace-1.cap, to exam TCP flow control. We observe how 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 analyze another trace, tcp-trace-retransmission.cap [3], in which retransmissions did 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 suffer from strong interferences. 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`.

![Network Configuration](image)

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

### 4.3 Procedures and Discussions

**Note:** Answer a set of questions by exploring the trace file `tcp-trace-1.cap` and `tcp.analysis.retransmission.cap`. When answering a question, you should provide the information of the packet(s) within the trace that you used to answer the question asked if possible. 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 questions below.

Discussions

1. Write down the TCP header content in hexadecimal format (in the packet bytes pane). Inspect 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 header in the 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.

4.3.2 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 SYN/ACK segment being returned.)

- Answer the questions below.
Discussions

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

2. What is the actual initial sequence number in each direction (in hexadecimal format)?

   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 SYN/ACK 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 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, SYN/ACK, and ACK segment?

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 Data 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 the time of 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 questions below.

**Discussions**

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 the relative value of the sequence number and acknowledgement number.)*

<table>
<thead>
<tr>
<th>Packet No.</th>
<th>Data Segments 10.0.1.5 --&gt; 128.119.245.12</th>
<th>ACK Segments 128.119.245.12 --&gt; 10.0.1.5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seq. No./Relative Seq. No.</td>
<td>Data Length</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

2. What are the segments acknowledged by packet 6, 9, 12, and 15, respectively? *(Hint: acknowledgement number is the next byte expected, so it actually acknowledges the byte before the acknowledgement number.)*
3. Given the difference between the time each TCP segment was sent and the time its acknowledgement was received, what is the RTT value for each of the segments which have been acknowledged before the 15th segment?

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.)

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?

4.3.4 TCP Connection Release

- Find the segments used to release the connection between the client and the server.

- Answer the questions below.

Discussions

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?

4.3.5 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 ⇒ 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 questions below.

Discussions

1. Exam the 4th to 15th TCP segments and take a reference to the Table in Question 1 of Section 4.3.3. 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 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?

4.3.6 TCP Flow Control

- Open the captured trace file named tcp-trace-1.cap with WireShark.
- Answer the questions below.

Discussions, TCP Flow Control

1. Examine 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?

4.3.7 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 questions below.

Discussions

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: Check whether the sequence number in the segment has been acknowledged for three times or not.)
Bibliography


