Friday, Feb. 3
Winter 2017

CME 451 – Lab 3

Analyzing Network Packet Headers [1]

Pre-Lab: While the material provided in this lab is intended to be self-contained, the interested reader is urged to also browse through the online tutorial of Python programming by Python Software Foundation, available here: `https://www.python.org/doc/`.

Verification: Exercises marked with an **asterisk** (∗) are required to be verified and signed off by a lab instructor during your assigned lab time. When you complete a step that requires verification, simply demonstrate the step to a lab instructor. Turn in the completed verification sheet to your TA at the end of the lab.

Code Submission: Exercises marked with a **dagger** (†) can be completed after your assigned lab time. Submit your code and relevant outputs electronically in one zip file (named <NSID>-<Lab#>) through the BlackBoard no later than a week after your lab. Enhance the readability of your code by adding explanatory comments wherever appropriate. If you are unsure about what is expected, ask your TA.

1 Introduction

In this lab, you will use the Wireshark software to capture and examine a packet trace. Also, you will learn to analyze the information in the headers of IP packets and then parse them into human-readable format. In the process of constructing the packet analyzer, you will review the header format of IPv4 and how it accomplishes its goals.

In addition, you need to review how IPv4 datagrams are fragmented while passing through different networks with different MTUs. Also you will analyze the header format of TCP segments in order to review how TCP accomplishes its goals.

2 Wireshark

Wireshark (formerly Ethereal) is an open-source network packet analyzer. It captures packets passing through a network interface and displays the contents of the various protocol fields. This is useful for troubleshooting network problems, auditing traffic and gathering statistics, detecting security issues, and many other applications.

2.1 Capturing a Trace

Launch Wireshark and choose the **Interfaces...** option under the **Capture** menu. It brings up a list of network interfaces from which Wireshark can capture packet traces. A packet trace is a record of the network traffic observed at a specific location of a network. Select your preferred network adapter and hit the **Start** button to capture live network traffic on your network interface[2]. Open a web browser, pick a URL and fetch it, and observe how the packets corresponding to your web browsing activity are captured and displayed in Wireshark. Press the **Stop** button once you are done running live capture.

A packet trace is uploaded on the Blackboard, corresponding to a simple web fetch of the URL http://www.usask.ca. Download the trace and open it in Wireshark. The main Wireshark interface is shown in Fig. 1.

In Fig. 1, the packet listing panel displays a one line summary for each captured packet, the packet header details panel shows details about the packet selected in the packet listing panel, and the packet contents panel provides the contents of the captured frame, in both hexadecimal and ASCII formats.

[1] Lab handouts are available online on the Blackboard. Please report typos and send comments to francis.bui@usask.ca.
[2] Wireshark needs to be run with administrator privileges for live packet capturing on a network interface.

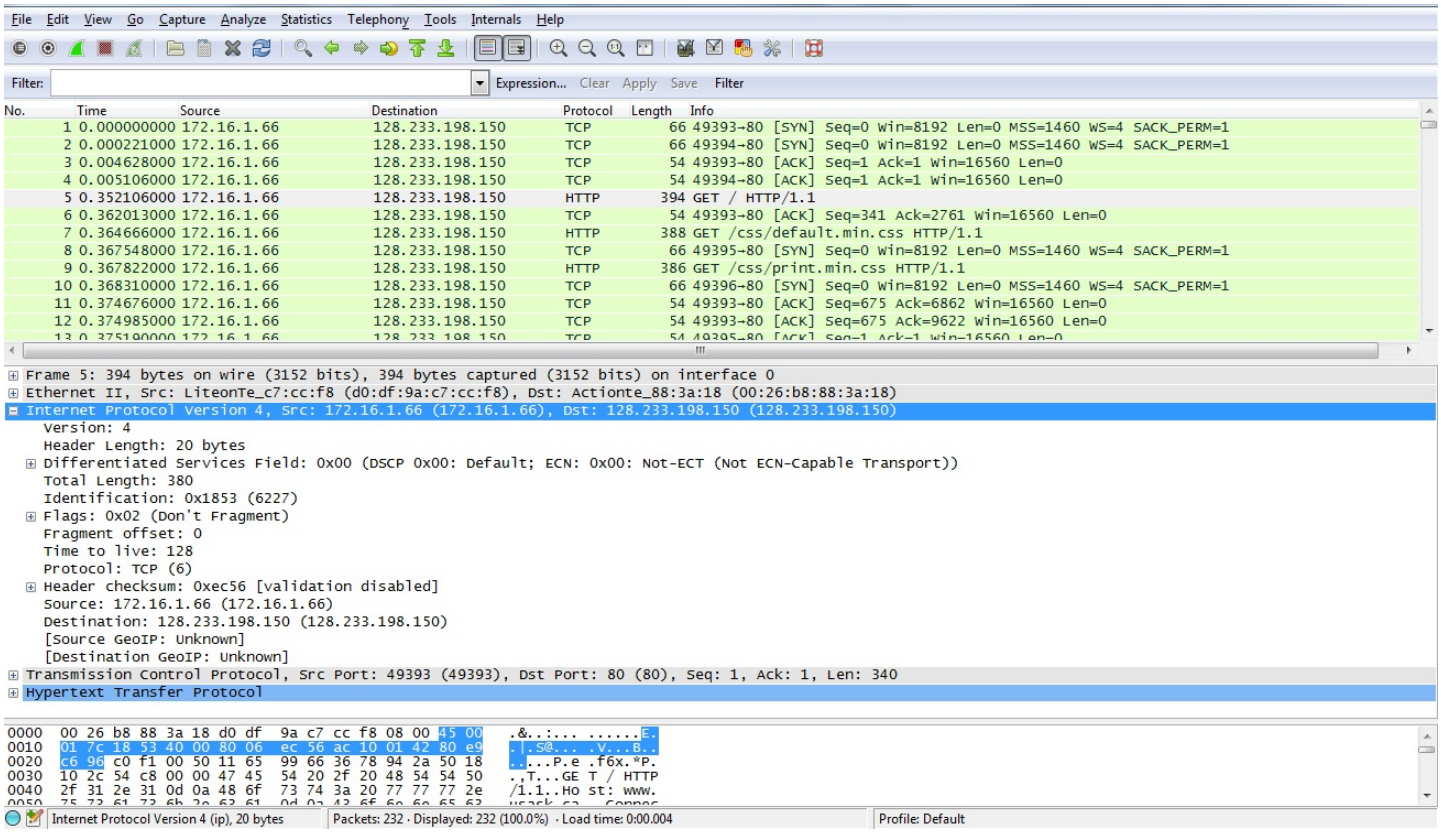

Figure 1: Wireshark Graphical User Interface

2.2 Filtering and Examining a Trace

Wireshark allows you to filter the traffic based on different criteria. Here, we want you to examine the HTTP traffic that occurs when fetching a web page. To this end, type HTTP in the filter toolbar and click on **Apply**. You can build more complex filters based on a variety of criteria using the **Expression...** option.

To examine the HTTP traffic, let us start with the first packet in the packet listing panel that corresponds to a HTTP request sent from your web browser to the server. Select a packet whose Info column says it is a GET.

In the packet header details panel, take a closer look at the packet structure which reflects the protocols that are in use. The protocol layers being used when fetching a web page are as follows. HTTP is the application layer web protocol used to fetch URLs. Like many Internet applications, it runs on top of the TCP/IP transport and network layer protocols. The link and physical layer protocols depend on your network, but are typically combined in the form of Ethernet if your network interface is wired (like for this trace), or 802.11 if your adaptor is wireless.

You can expand each protocol block listed in the packet header details panel to see further details. The first Wireshark block is Frame. This is not a protocol, but rather a record that describes overall information about the packet. The rest of the blocks from the bottom of the protocol stack upwards are Ethernet, IP, TCP, and HTTP.

Now, let us take a look at the packet that carries the HTTP response to the above request. In the packet listing panel, select the immediate next packet that has a "200 OK" in its Info field, indicating a successful fetch.

This packet has two extra blocks in the packet header details panel. The first extra block named "[... reassembled TCP segments ...]", lists the packets that are joined together to make the HTTP response. A web response is mist likely sent across the network as a series of packets that are put together by the client after they arrived. The second extra block named "Line-based text data ...", describes the contents of the web page fetched. For this packet, it is of the type text/html, but can easily be text/xml, image/jpeg, or other types.

Also look at the range of the Ethernet header and payload, as well as the IP header and payload for the two packets examined.

3 IP Packet

The format of IPv4 packet is shown in Fig. 4. Given this layered structure, you are asked in Exercise 3.1 to analyze an IP packet, extract various pieces of information from its header fields, and display the detailed information in an appropriate format.

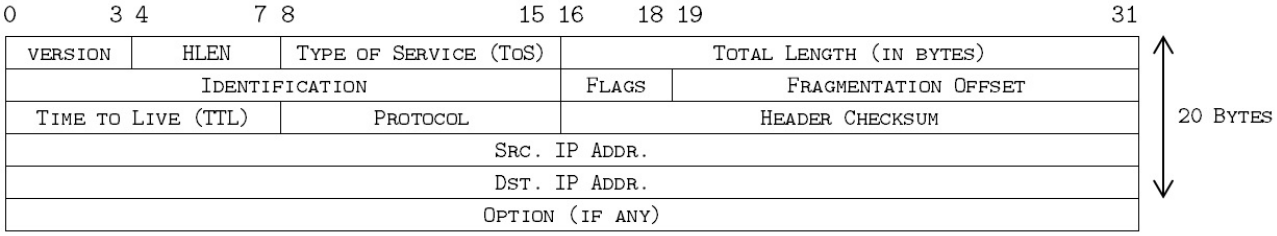

Figure 2: IPv4 Packet Format

Exercise 3.1 (*Analyzing IP Packets*) *

You are given the following hexadecimal data stream representing an IP packet.

45000028*3b*1*a*400080066*ac*40*ae*30108*cdc*47*b*42*c*76*a*0050498*a*3*cdaadbd*4*f*1*f*5010*fd*5*c*128*a*0000

Perform the following tasks:

(i) Find out the total length of the IP packet, the type of the packet (*i.e.*, the transport layer protocol), and the number of more routers each packet can travel to. Consult your lecture notes if necessary.

(ii) Write a Python script that gets the above data stream as the input, extracts the IP headers, parse the header information into a human-readable format, and save them as pairs of <field:value> in a Python dictionary structure.

(iii) Write another Python script that generates an HTML file and displays the dictionary items in the previous part in a tabular format (similar to Fig. 4).

Hint: A useful data type built into Python is the dictionary (see Python online documentation). Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys.

It is best to think of a dictionary as an unordered set of `key:value` pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}. Placing a comma-separated list of `key:value` pairs within the braces adds initial `key:value` pairs to the dictionary; this is also the way dictionaries are written on output.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a `key:value` pair with `del`. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.

The `keys()` method of a dictionary object returns a list of all the keys used in the dictionary, in arbitrary order (if you want it sorted, just apply the `sorted()` function to it). To check whether a single key is in the dictionary, use the `in` keyword.

Here is a small example using a dictionary:

```
>>> course = {'EE 845': 1, 'EE 840': 2}
>>> course['CME 451'] = 3
>>> course
{'EE 840': 2, 'CME 451': 3, 'EE 845': 1}
>>> course['CME 451']
3
>>> del course['EE 840']
>>> course
{'CME 451': 3, 'EE 845': 1}
>>> course.keys()
['CME 451', 'EE 845']
>>> 'CME 451' in course
True
```

4 Ethernet Frame

As discussed earlier, packet analyzer software, such as Wireshark, collect network traffic traces in the form of Ethernet frames. The internal structure of an Ethernet frame is specified in IEEE 802.3-2012. Fig. 3 illustrates a simplified format of an Ethernet frame.

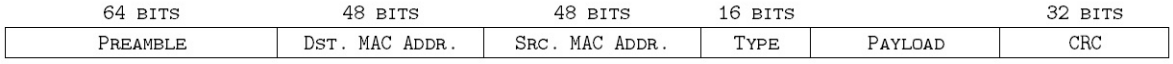

Figure 3: Ethernet Frame Format

If the `Type` field has the value of `0x0800`, then the payload of the Ethernet frame contains an IP packet.

Exercise 4.1 (*Analyzing Ethernet Frames*) †

You are given the following hexadecimal data stream representing a few Ethernet frames of a network traffic trace collected by Wireshark. (Frames do not have preamble, nor CRC checksum.)

00000*c*9*ff*001*d*0*df*9*ac*7*ccf*80800450000283*b*1*a*400080066*ac*40*ae*30108*cdc*47*b*42*c*76*a*0050498*a*3*cdaadbd*398*f*5010*fd*5*c*
281*a*000000000*c*9*ff*001*d*0*df*9*ac*7*ccf*80800450000283*b*1*b*400080066*ac*30*ae*30108*cdc*47*b*42*c*76*a*0050498*a*3*cdaadbd*44575
010*fd*5*c*1*d*52000000000*c*9*ff*001*d*0*df*9*ac*7*ccf*80800450000283*b*1*c*400080066*ac*20*ae*30108*cdc*47*b*42*c*76*a*0050498*a*3*cdaad*
*bd*4*f*1*f*5010*fd*5*c*128*a*0000

Perform the following tasks:

(i) Find out the number of Ethernet frames this traffic trace consists of.

(ii) Modify your python script form Exercise 3.1, part (iii) to parse the information in each frame into a human-readable format and then show the information in a tabular format in an HTML file.

5 Fragmentation

An IPv4 datagram may travel through different networks in its route. Each router decapsulates the datagram from the frame it receives, processes it, and then encapsulates it in another frame. When the datagram is encapsulated in a frame, its total size must be less than a maximum size (MTU) imposed by the hardware and software restrictions in the network through which the frame is sent. Therefore sometimes the source host or the routers in the path must divide datagrams into fragments in order to pass them through different networks. A datagram may be fragmented several times before it reaches its final destination.

Each fragment has its own header with most of the fields copied from the original datagram. In particular, you must change the values of three header fields for the fragments: flags, fragmentation offset, and total length.

(It goes without saying that, regardless of fragmentation, the checksum value also must be recalculated but is not considered in this lab.)

Exercise 5.1 (*Fragmentation and Header Format*) ∗

You are given the following IPv4 datagram.

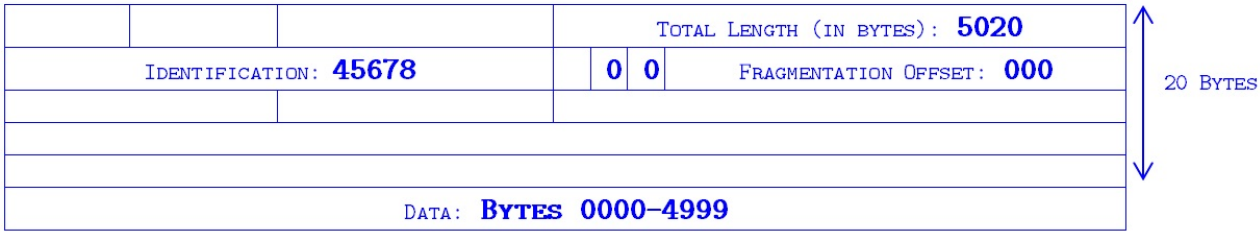

Figure 4: Original Datagram

(i) We want to pass this datagram through an Ethernet network. Divide the datagram into necessary fragments and change the values of the IP header fields shown in Fig. 4. Consult your lecture notes if necessary. Then write a Python script that generates an HTML file and displays the fragmented datagrams in a tabular format, similar to what is shown in the lecture slides (slide #74, Chapter 05 - TCP/IP Part 1).

(ii) The second fragmented datagram from the previous part needs to be sent through an X.25 network. Repeat the tasks in the previous step.

(iii) Design and implement a python script to defragment (*i.e.*, combine the fragments) in order to generate the complete original datagram. Your script should take an HTML file containing a number of datagrams in tabular format as the input and check whether they are fragments of a larger IP datagram before combing them. To simplify your design, you may assume that the number of fragments is known (*e.g.*, available as an input parameter). Demonstrate your implementation for recovering of the datagram, after being fragmented by parts (i) & (ii).

6 TCP Segment Header Processing

You are asked in Exercise 6.1 to analyze the header of a TCP segment, extract various pieces of information, and display the detailed information in an appropriate format.

Exercise 6.1 (*Analyzing TCP Header*) †

You are given the following hexadecimal data stream representing the header information of a TCP segment.

$f7500050ce0a0711208ae25c50181020da3f0000$

(i) Write a Python script that gets the above data stream as the input, extracts the TCP header field values, parse them into a human-readable format, and save them as pairs of <field:value> in a Python dictionary structure. Consult your lecture notes if necessary.

(ii) Write another Python script that generates an HTML file and displays the dictionary items in the previous part in a tabular format similar to Fig. 5.

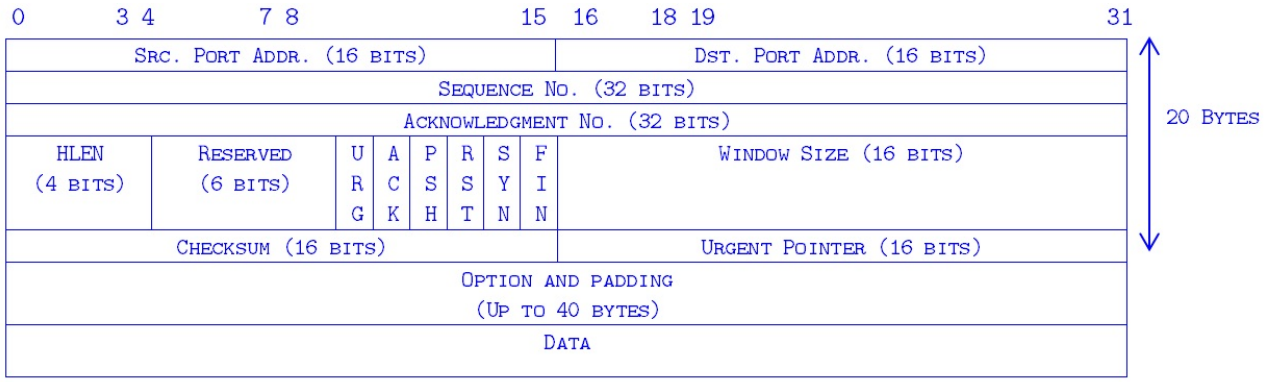

Figure 5: Format of a TCP segment

CME 451 Lab 3 - Instructor Verification Sheet

Turn this page to your grading TA.

Name:

Student No.:

Instructor Verification: Exercise 3.1, 5.1	
Verified by:	Date & Time: