**Computer Communications and Networks**



**Lab 07**

**Introduction to Wireshark**

**HTTP and DNS on Wireshark**

Lab Manual 07

|  |
| --- |
| Objectives  * Introduction to Wireshark * Running Wireshark * Explore Wireshark Filters * DNS * HTTP on Wireshark * Tracing DNS with Wireshark * Lab Graded Task |

# Reference Material

**Introduction to Wireshark**

Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color-coding and other features that let you dig deep into network traffic and inspect individual packets.

**Running Wireshark**

When you run the Wireshark program, the Wireshark graphical user interface shown in Figure 1 will displayed. Initially, no data will be displayed in the various windows.

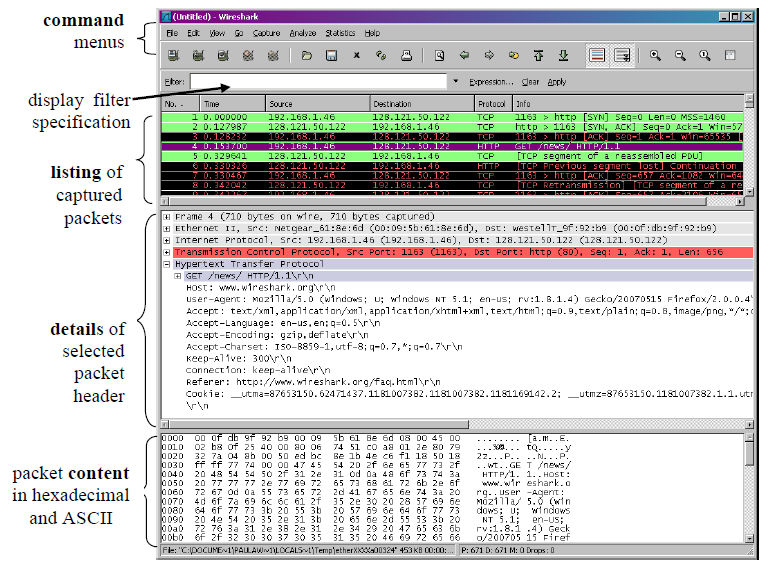


Figure: Wireshark Graphical User Interface

**Wireshark Filters**

The simplest filter allows you to check for the existence of a protocol or field. If you want to see all packets which contain the IP protocol, the filter would be "ip" (without the quotation marks).

1. **Comparison operators**

Fields can also be compared against values. The comparison operators can be expressed either through English-like abbreviations or through C-like symbols:

eq, == Equal

ne, != Not Equal

gt, > Greater Than

lt, < Less Than

ge, >= Greater than or Equal to

le, <= Less than or Equal to

**Example**

ip.addr == 10.0.0.1 [Sets a filter for any packet with 10.0.0.1, as either the source or dest]

tcp.time\_delta > .250 [sets a filter to display all tcp packets that have a delta time of greater than 250mSec in the context of their stream

1. **Search and match operators**

Additional operators exist expressed only in English, not C-like syntax:

contains Does the protocol, field contain a value

matches, ~ Does the protocol or text string match the given case-insensitive Perl-compatible regular expression

The "contains" operator allows a filter to search for a sequence of characters, expressed as a string (quoted or unquoted), or bytes, expressed as a byte array, or for a single character, expressed as a C-style character constant.

**Example**

To search for a given HTTP URL in a capture, the following filter can be used:

http contains <https://www.wireshark.org>

The "contains" operator cannot be used on atomic fields, such as numbers or IP addresses.

The "matches" or "~" operator allows a filter to apply to a specified Perl-compatible regular expression (PCRE). The "matches" operator is only implemented for protocols and for protocol fields with a text string representation. Matches are case-insensitive by default. For example, to search for a given WAP WSP User-Agent, you can write:

wsp.user\_agent matches "cldc"

This would match "cldc", "CLDC", "cLdC" or any other combination of upper and lower case letters.

1. **The membership operator**

A field may be checked for matches against a set of values simply with the membership operator. For instance, you may find traffic on common HTTP/HTTPS ports with the following filter:

tcp.port in {80 443 8080}

as opposed to the more verbose:

tcp.port == 80 or tcp.port == 443 or tcp.port == 8080

**Example**

To find HTTP requests using the HEAD or GET methods:

http.request.method in {"HEAD" "GET"}

The set of values can also contain ranges:

tcp.port in {443 4430..4434}

ip.addr in {10.0.0.5 .. 10.0.0.9 192.168.1.1..192.168.1.9}

frame.time\_delta in {10 .. 10.5}

1. **Logical expressions**

Tests can be combined using logical expressions. These too are expressible in C-like syntax or with English-like abbreviations:

and, && Logical AND

or, || Logical OR

not, ! Logical NOT

**Example**

ip.addr==10.0.0.1  && ip.addr==10.0.0.2 [sets a conversation filter between the two defined IP addresses]

!(arp or icmp or stp) [masks out arp, icmp, stp, or whatever other protocols may be background noise.

Expressions can be grouped by parentheses as well. The following are all valid display filter expressions:

tcp.port == 80 and ip.src == 192.168.2.1

not llc

http and frame[100-199] contains "wireshark"

(ipx.src.net == 0xbad && ipx.src.node == 0.0.0.0.0.1) || ip

**DNS: Domain Name System**

The domain name system (DNS) is a naming database in which internet domain names are located and translated into internet protocol (IP) addresses. The domain name system maps the name people use to locate a website to the IP address that a computer uses to locate a website.

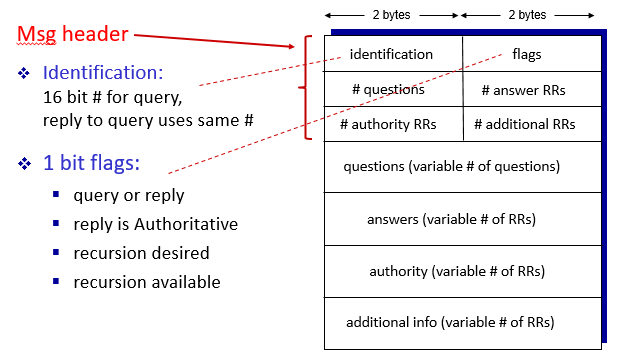
**DNS Records**

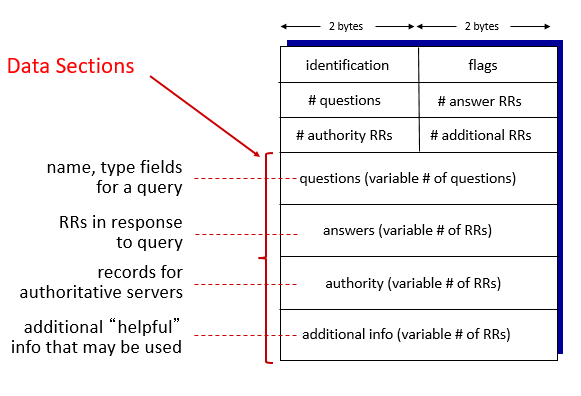
* DNS servers store resource records (RR)
* Each DNS Reply carries one/more RRs

RR format: (name, value, type, ttl)

**DNS protocol, messages**

* Query & Reply messages, both with same message format





**Taking Wireshark for a Test Run:**

1. Open Wireshark, select interface from list of interfaces (Ethernet in your case). Change your Interface to the appropriate one from the list provided. Then Press the capture Start button to start capturing the packets at run time.
2. While Wireshark is running, enter the URL:

<http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html> and have that page displayed in your browser.

1. Now enter another URL <http://gaia.cs.umass.edu/favicon.ico> and you will see that this page is not found on the server.
2. In order to display both the pages, your browser will contact the HTTP server at gaia.cs.umass.edu and exchange HTTP messages with the server in order to download this page. The Ethernet frames containing these HTTP messages will be captured by Wireshark.
3. After your browser has displayed both the web pages, stop Wireshark packet capture by selecting stop in the Wireshark capture window. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP message exchanges with the gaia.cs.umass.edu web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well.
4. Type in “http” (without the quotes, and in lower case – all protocol names are in lower case in Wireshark) into the display filter specification window at the top of the main Wireshark window. Then select Apply (to the right of where you entered “http”). This will cause only HTTP message to be displayed in the packet-listing window.
5. Select the first http message shown in the packet-listing window. This should be the HTTP GET message that was sent from your computer to the gaia.cs.umass.edu HTTP server. When you select the HTTP GET message, the Ethernet frame, IP datagram, TCP segment, and HTTP message header information will be displayed in the packet-header window3. By clicking plus and- minus boxes to the left side of the packet details window, minimize the amount of Frame, Ethernet, Internet Protocol, and Transmission Control Protocol information displayed. Maximize the amount information displayed about the HTTP protocol. Your Wireshark display should now look roughly as shown in Figure 5. (Note in particular, the minimized amount of protocol information for all protocols except HTTP, and the maximized amount of protocol information for HTTP in the packet-header window).
6. Now try to find out the packet which contains the second request you sent to the browser and also analyze the packet which your browser received as a result of second GET Request.

# Lab Tasks

1. **Explore the packets you captured from test run and answer the following questions**

**[10 Marks]**

1. List up to 4 different protocols that appear in the protocol column in the unfiltered packet-listing window.

**TCP, UDP, HTTP, DNS, ARP, IMAP, TLSV1.2, ...... (Any 3 possible protocols will be accepted.)**

1. What is the response time against HTTP GET Request?

page1image13056

1. Was the second Get Request successful? How can you tell it from the corresponding response packet?

**Solution:  
gaia.cs.umass.edu: 128.119.245.12 my computer: xxx.xxx.xxx.xxx**

By looking at the information in the HTTP GET and Response Messages for both the HTTP Requests, answer the following questions

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?
2. What languages (if any) does your browser indicate that it can accept to the server?
3. What is the IP address of the gaia.cs.umass.edu server and your computer?
4. What is the MAC address of the server and your computer?
5. What is sending and receiving Port Number? What does Port No. 80 represents?
6. When was the HTML file, that you are retrieving, last modified at the server?
7. How many bytes of total packet content are being returned to your browser?

**Note: Make a Word file and post the screen shots of all the answers in it. Apart from the answers explore different settings of wireshark, analyze all the layers of the HTTP Packets and try to understand how layering system works in Computer Networks. Implement different filters in your data to view different grouping of packets. Make yourself familiar with the software as we will be using it in the next labs frequently.**

1. **Tracing DNS with Wireshark [10 Mark]**

First, capture the DNS packets that are generated by ordinary Web surfing activity.

* Use ipconfig to empty the DNS cache in your host.
* Open your browser and empty your browser cache. (With Internet Explorer, go to Tools menu and select Internet Options; then in the General tab select Delete Files.)
* Open Wireshark and enter “ip.addr == your\_IP\_address” into the filter, where you obtain your\_IP\_address (the IP address for the computer on which you are running Wireshark) with ipconfig. This filter removes all packets that neither originate nor are destined to your host.
* Start packet capture in Wireshark.
* With your browser, visit the Web page: <http://www.ietf.org>
* Stop packet capture.

Answer the following questions:

1. Locate the DNS query and response messages. Are they sent over UDP or TCP?
2. What is the destination port for the DNS query message? What is the source port of DNS response message?
3. To what IP address is the DNS query message sent? Use ipconfig to determine the IP address of your local DNS server. Are these two IP addresses the same?
4. Examine the DNS query message. What “Type” of DNS query is it? Does the query message contain any “answers”?
5. Examine the DNS response message. How many “answers” are provided? What does each of these answers contain?
6. Consider the subsequent TCP SYN packet sent by your host. Does the destination IP address of the SYN packet correspond to any of the IP addresses provided in the DNS response message?
7. This web page contains images. Before retrieving each image, does your host issue new DNS queries?