Review: The TCP/IP network does not know how to route using names, such as www.yahoo.com. It only knows how to route using IP addresses. Therefore, a common name (CNAME), such as www.yahoo.com must be translated to an IP address, like 216.109.117.106 before your computer can request a web page.

Your computer uses a DNS request to lookup a CNAME and it gets back a set of IP addresses (a primary address and one or more backup addresses) that can be used to contact the server.

Hint: See below appendix describing "Hits Versus Page Views".

Lab1– Basic Wireshark network analysis

Open "Wireshark", then use the "File" menu and the "Open" command to open the file "Lab1.pcap". You should see 176 packets listed.

a) In the first few packets, the client machine is looking up the common name (cname) of a web site to find its IP address. What is the cname of this web site? Give two IP addresses for this web site.
b) How many packets/frames does it take to receive the web page (the answer to the first http get request only)?
c) Does this web site use gzip to compress its data for sending? Does it write cookies? In order to answer these questions, look under the payload for the reassembled packet that represents the web page. This will be the last packet from question b above. Look to see if it has "Content-Encoding" set to gzip, and to see if it has a "Set-Cookie" to write a cookie.
d) What is happening in packets 26 and 27? Does every component of a web page have to come from the same server? See the Hint to the left.
e) In packet 37 we see another DNS query, this time for us.i1.yimg.com. Why does the client need to ask for this IP address? Didn't we just get this address in packet 26? (This is a trick question; carefully compare the two common names in packet 26 and 37.)
f) In packet 42 we see a HTTP "Get" statement, and in packet 48 a new HTTP "Get" statement. Why didn't the system need another DNS request before the second get statement? Click on packet 42 and look in the middle window. Expand the line titled "Hypertext Transfer Protocol" and read the "Host:" line. Compare that line to the "Host:" line for packet 48.
g) Examine packet 139. It is one segment of a PDU that is reassembled with several other segments in packet 160. Look at packets 141, 142, and 143. Are these three packets also part of packet 160? What happens if a set of packets that are supposed to be reassembled do not arrive in a continuous stream or do not arrive in the proper order?
h) Return to examine frames 141 and 142. Both of these are graphics (GIF files) from the same source IP address. How does the client know which graphic to match up to each get statement? Hint: Click on each and look in the middle window for the heading line that

starts with “Transmission Control Protocol”. What difference do you see in the heading lines for the two files? Return to the original “Get” statements. Can you see the same difference in the “Get” statements?

Appendix: Hits Versus Page Views

This topic is appropriate for this guide because it helps explains the plethora of packets that add together to display a single web page. However, it is also interesting to consider the implications for the number of ‘hits’ a web site gets.

Let’s analyze what it takes to get a million hits on a web page. First, assume an average page has 150 images. In comparison, this would be 10% smaller than CNN’s front page. Now, assume each visitor sees three pages on the web site. It will take less than 2,300 visitors to get one million hits on this hypothetical web site.

150 Hits/Page
X 3 Pages/Visitor
X 2,300 Visitors
= 1,035,000 hits

It may take more effort than you realize to deliver a web page to your computer. The first step is to get the raw HTML code for the page. Getting this code takes several sets of packets – the details will be left to an exercise to be completed later, but suffice it to say that retrieval includes setup and control packets as well as query and response packets. Furthermore, in most cases the response will be a multi-packet data burst that must be reassembled into a complete http response.

However, once the page is delivered to the application, the system has only completed the first step required to display the web page. Let’s consider a simplified web page in HTML, as shown in the box below.

```
<HTML>
<Body>
Look at this pretty Christmas tree.<br>
<img src=tree.jpg>
</Body>
</HTML>
```

Figure : Simplified Web Page

This web page will display a short sentence (Look at this pretty Christmas tree.), followed by a line break, and then a picture of a tree. Notice that the picture of the tree is not part of the HTML page that is delivered. All that gets delivered with the page is a placeholder that tells the browser to get the picture called tree.jpg and to put it into a specific spot on the page.

So, once the browser deciphers the web page, it knows it must make another request of the web server. Now the browser asks for the picture tree.jpg. As a result, displaying this page takes two hits on the browser. One hit (or request) was for the original web page, and the second hit was for the picture to be embedded into the web page. Each additional picture or external page element is another hit on the web page.

How many pictures are on a single page? 10? 20? A recent analysis of the CNN front page indicated over one hundred and fifty separate files were required to display the page. A lot of

these files are graphic files. This includes tiny graphic arrows, almost invisible lines, menu choices, and advertisements. In addition, javascript files, stylesheets, and iFrames can all be external links, and thus can be additional sources of hits.

Especially in the case of advertisements, these hits may not come from the original web site. Therefore, at the packet level there may be many packets from many different sources that have to be considered as part of the same web page.

Increasingly, developers are making dynamic web pages. This means that some portion of the web page may be continuously updated through interaction between the user and the server. This dynamic process requires ongoing hits on the server, even after the web page is initially ‘complete’.

Since each of these hits results in a new request from the server, the number of packets required to assemble a web page is larger than many people realize.