

ECE356 Wireshark Lab

Andrew Shim

19 September 2012

Part 1 - Getting Started

1.1

List up to 10 different protocols:

1. HTTP
2. TCP
3. GLBP
4. STP
5. LOOP
6. LLMNR
7. DHCPv6
8. ICMP
9. BROWSER
10. ARP

1.2

Looking at the red boxes in Figures 1 and 2 on the next page we see that the HTTP GET was sent at 20:01:13.271848 and the HTTP OK reply was received at 20:01:13.296157. Thus the time it took for the reply was:

$(0.296157 - 0.271858)$ seconds = **0.024309 seconds**

No.	Time	Source	Destination	Protocol	Length	Info
164	20:01:13.271848	10.181.19.70	128.119.245.12	HTTP	631	GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1

Frame 164: 631 bytes on wire (5048 bits), 631 bytes captured (5048 bits)

Ethernet II, Src: Clevo d1:aa:90 (00:90:f5:d1:aa:90), Det: Cisco 00:03:03 (00:07:b4:00:03:03)

Internet Protocol Version 4, Src: 10.181.19.70 (10.181.19.70), Det: 128.119.245.12 (128.119.245.12)

Transmission Control Protocol, Src Port: 51686 (51686), Det Port: http (80), Seq: 1, Ack: 1, Len: 565

Hypertext Transfer Protocol

```
GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1\r\n
Host: gaia.cs.umass.edu\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.19 (KHTML, like Gecko) Ubuntu/12.04 Chromium/18.0.1025.168 Chrome/18.0.1025.168 Safari/535.19\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Encoding: gzip,deflate,edch\r\n
Accept-Language: en-US,en;q=0.8\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3\r\n
If-None-Match: "8734b-51-cdff6240"\r\n
If-Modified-Since: Thu, 20 Sep 2012 00:00:01 GMT\r\n
\r\n
[Full request URI: http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html]
```

Figure 1: HTTP GET

No.	Time	Source	Destination	Protocol	Length	Info
166	20:01:13.296157	128.119.245.12	10.181.19.70	HTTP	446	HTTP/1.1 200 OK (text/html)

Frame 166: 446 bytes on wire (3568 bits), 446 bytes captured (3568 bits)

Ethernet II, Src: Cisco 21:3b:40 (00:15:c7:21:3b:40), Det: Clevo d1:aa:90 (00:90:f5:d1:aa:90)

Internet Protocol Version 4, Src: 128.119.245.12 (128.119.245.12), Det: 10.181.19.70 (10.181.19.70)

Transmission Control Protocol, Src Port: http (80), Det Port: 51686 (51686), Seq: 1, Ack: 566, Len: 380

Hypertext Transfer Protocol

```
HTTP/1.1 200 OK\r\n
Date: Thu, 20 Sep 2012 00:01:11 GMT\r\n
Server: Apache/2.2.3 (CentOS)\r\n
Last-Modified: Thu, 20 Sep 2012 00:01:01 GMT\r\n
ETag: "8734b-51-d192e940"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 81\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=UTF-8\r\n
\r\n
Line-based text data: text/html
```

Figure 2: HTTP OK

1.3

Looking again at Figures 1 and 2 we see that the Internet address of gaia.cs.umass.edu is 128.119.245.12 (green boxes) and the Internet address of my host is 10.181.19.70 (blue boxes).

1.4

HTTP GET message: See Figure 1.

HTTP OK message: See Figure 2.

Part 2 - HTTP

2.1

If I just enter "http" in the filter then one other protocol, SSDP, gets additionally listed. Deducing from the semantics of the filtering criteria, I can conclude that the "http && (!udp.port == 1900)" criteria filters out any packets sent through UDP to the destination port 1900. Also, the "http" criteria allows the SSDP packets to appear in the output because those packets do use some HTTP-related methods although the primary underlying protocol is UDP.

2.2

IP address of my computer: 10.181.19.70

IP address of the server: 152.3.67.47

2.3

The status code returned from the server to my browser was 200. There are many other HTTP status codes ranging from the 100s to the 500s. The codes that I have seen most often are:

400 - Bad Request

403 - Forbidden

404 - Not Found

500 - Internal Server Error

2.4

Time	10.181.19.70	74.125.137.99	152.3.67.47	
3.514	GET /complete/searc			HTTP: GET /complete/search?client=chrome&hl=en-US&q=%26%26*(!udp.port+%3D%3D+1900) HTTP/1.1
	(54662)> (80)		
3.560	HTTP/1.1 200 OK (t			HTTP: HTTP/1.1 200 OK (text/javascript)
	(54662)	<..... (80)		
4.603	GET /complete/searc			HTTP: GET /complete/search?client=chrome&hl=en-US&q=%26%26*(!udp.port+%3D%3D+1900) HTTP/1.1
	(54662)> (80)		
4.654	HTTP/1.1 200 OK (t			HTTP: HTTP/1.1 200 OK (text/javascript)
	(54662)	<..... (80)		
11.508	GET /~ronit/index.h			HTTP: GET /~ronit/index.html HTTP/1.1
	(34824)> (80)		
11.540	HTTP/1.1 200 OK (t			HTTP: HTTP/1.1 200 OK (text/html)
	(34824)	<..... (80)		

Figure 3: Flow Graph Output

Examining the figure above we can see that every GET request is accompanied by an OK response. In a more general view, the client sends HTTP requests to the server and the server replies to the client through HTTP responses. Depending on the status of the server or a variety of other conditions, the server will send a particular HTTP response to the client (e.g. if the client is not allowed to access the data requested in the GET the server would reply with a 403 Forbidden response).

2.5

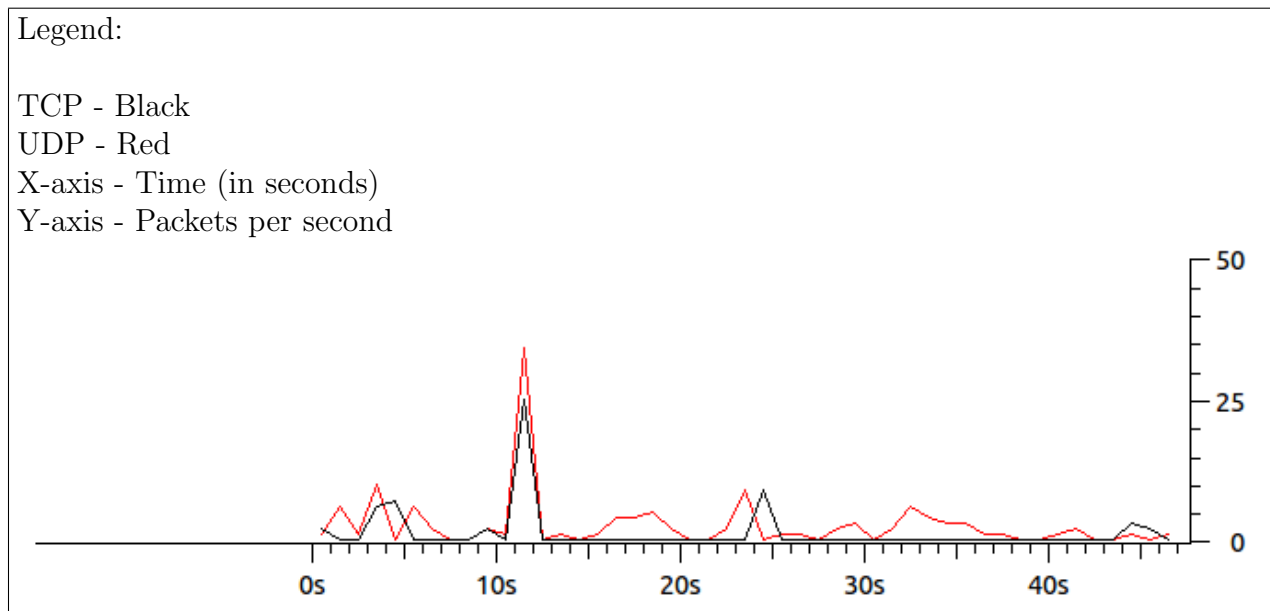


Figure 4: TCP vs. UDP I/O Graph

Looking at the I/O graph above we see that more UDP (red line) than TCP (black line) packets were sent. From taking a quick look at the Wireshark output, it looks as if most of the underlying communications used UDP. Basically, the communications that I (the user) do not necessarily need to worry about were primarily using UDP, whereas every communication that was a direct result of my actions (e.g. interaction with the browser) used TCP. Note the similarity of the UDP and TCP lines, where it appears as if the lines are overlayed. Assuming that this relationship is more than a coincidence, I would say that UDP packets are sent to pave the way for a TCP connection so that it can be properly established.

2.6

The HTTP message was sent over TCP. When you select a packet in Wireshark, Wireshark displays the details of the packet. The details section includes information about the frame, network connection, internet protocol, and which protocol it was sent through (e.g. TCP, UDP, etc.); from examining this section I was able to figure out that TCP was used.

Part 3: DNS

No. Info	Time	Source	Destination	Protocol	Length
118	17:31:34.383308	10.181.19.70	152.3.72.100	DNS	72
Standard query A www.ietf.org					
Frame 118: 72 bytes on wire (576 bits), 72 bytes captured (576 bits)					
Ethernet II, Src: Clevo_d1:aa:90 (00:90:f5:d1:aa:90), Dst: Cisco_00:03:03 (00:07:b4:00:03:03)					
Internet Protocol Version 4, Src: 10.181.19.70 (10.181.19.70), Dst: 152.3.72.100 (152.3.72.100)					
User Datagram Protocol, Src Port: 64898 (64898), Dst Port: domain (53)					
Source port: 64898 (64898)					
Destination port: domain (53)					
Length: 38					
Checksum: 0xafdc [validation disabled]					
Domain Name System (query)					

Figure 5: DNS Query Message

No. Info	Time	Source	Destination	Protocol	Length
119	17:31:34.565987	152.3.72.100	10.181.19.70	DNS	309
Standard query response A 64.170.98.30					
Frame 119: 309 bytes on wire (2472 bits), 309 bytes captured (2472 bits)					
Ethernet II, Src: Cisco_21:3b:40 (00:15:c7:21:3b:40), Dst: Clevo_d1:aa:90 (00:90:f5:d1:aa:90)					
Internet Protocol Version 4, Src: 152.3.72.100 (152.3.72.100), Dst: 10.181.19.70 (10.181.19.70)					
User Datagram Protocol, Src Port: domain (53), Dst Port: 64898 (64898)					
Source port: domain (53)					
Destination port: 64898 (64898)					
Length: 275					
Checksum: 0x97fc [validation disabled]					
Domain Name System (response)					

Figure 6: DNS Response Message

3.1

Looking at the red boxes in Figures 5 and 6 we can see that the DNS messages were sent over UDP.

3.2

Looking at the blue boxes in Figures 5 and 6 we can see that:

Destination port for DNS query = 53

Source port for DNS response = 53

3.3

No. Info	Time	Source	Destination	Protocol	Length
120	17:31:34.566473	10.181.19.70	64.170.98.30	TCP	74
45513 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=36349999 TSecr=0 WS=128					
Frame 120: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) Ethernet II, Src: Clevo_d1:aa:90 (00:90:f5:d1:aa:90), Dst: Cisco_00:03:03 (00:07:b4:00:03:03) Internet Protocol Version 4, Src: 10.181.19.70 (10.181.19.70), Dst: 64.170.98.30 (64.170.98.30) Transmission Control Protocol, Src Port: 45513 (45513), Dst Port: http (80), Seq: 0, Len: 0					

Figure 7: TCP SYN Packet Information

From Figure 7 we observe that the DNS response replied that the IP address of www.ietf.org is 64.170.98.30 (green box). Now looking back to Figure 6, we see that the TCP SYN packet was sent to that same IP address 64.170.98.30. So yes, the destination IP address of the TCP SYN packet corresponds to the IP address provided in the DNS response message.

3.4

My host did issue new DNS queries but, after looking at the HTML source of the web page, I do not believe that those queries were for the images because the images were provided by www.ietf.org. After the first DNS query for that name, my host cached the response IP address, so by the time it was retrieving the images it already knew which IP address it needed to get those images from. The new DNS queries had to do with the links on the web page that pointed to other servers for which my host did not know the IP addresses of (e.g. ioac.ietf.org, open-stand.org, rfc-editor.org).