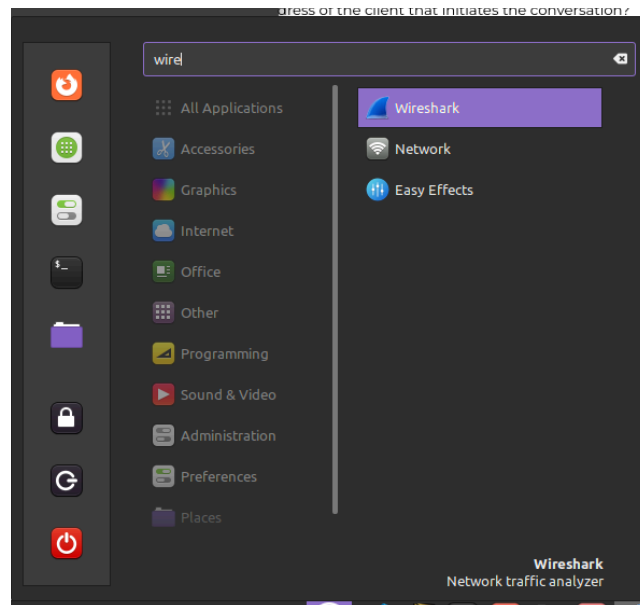


Installing WireShark

- Install with `sudo apt install wireshark` #Ubuntu/Debian
- Open the app and continue



Exercise One

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	131.247.95.216	131.247.92.200	DNS	74	Standard query 0xefc3 A www.google.com
2	0.000405	131.247.92.200	131.247.95.216	DNS	142	Standard query response 0xefc3 A www.google.com CNAME www.l.google.com A 64.233.161.99 A 64.233.161.104 A 64.233.161.147
3	0.001925	131.247.95.216	64.233.161.99	TCP	62	1143 → 80 [SYN] Seq=0 Win=16384 Len=0 MSS=1460 SACK_PERM=1
4	0.027528	64.233.161.99	131.247.95.216	TCP	60	80 → 1143 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
5	0.027635	131.247.95.216	64.233.161.99	TCP	54	1143 → 80 [ACK] Seq=1 Ack=1 Win=17520 Len=0
6	0.027997	131.247.95.216	64.233.161.99	HTTP	546	GET / HTTP/1.1
7	0.053913	64.233.161.99	131.247.95.216	TCP	60	80 → 1143 [ACK] Seq=1 Ack=493 Win=7698 Len=0
8	0.054815	64.233.161.99	131.247.95.216	TCP	60	[TCP Window Update] 80 → 1143 [ACK] Seq=1 Ack=493 Win=6432 Len=0
9	0.073552	64.233.161.99	131.247.95.216	TCP	1484	80 → 1143 [ACK] Seq=1 Ack=493 Win=6432 Len=1430 [TCP segment of a reassembled PDU]
10	0.073623	64.233.161.99	131.247.95.216	HTTP	275	HTTP/1.1 200 OK (text/html)
11	0.073677	131.247.95.216	64.233.161.99	TCP	54	1143 → 80 [ACK] Seq=493 Ack=1652 Win=17520 Len=0
12	0.129189	131.247.95.216	64.233.161.99	HTTP	522	GET /intl/en/images/logo.gif HTTP/1.1
13	0.166185	64.233.161.99	131.247.95.216	TCP	1484	80 → 1143 [ACK] Seq=1652 Ack=961 Win=7504 Len=1430 [TCP segment of a reassembled PDU]
14	0.166293	64.233.161.99	131.247.95.216	TCP	1484	80 → 1143 [ACK] Seq=3082 Ack=961 Win=7504 Len=1430 [TCP segment of a reassembled PDU]
15	0.166353	131.247.95.216	64.233.161.99	TCP	54	1143 → 80 [ACK] Seq=961 Ack=4512 Win=17520 Len=0
16	0.166397	64.233.161.99	131.247.95.216	TCP	1484	80 → 1143 [ACK] Seq=4512 Ack=961 Win=7504 Len=1430 [TCP segment of a reassembled PDU]
17	0.193399	64.233.161.99	131.247.95.216	TCP	1484	80 → 1143 [ACK] Seq=5942 Ack=961 Win=7504 Len=1430 [TCP segment of a reassembled PDU]
18	0.193515	64.233.161.99	131.247.95.216	TCP	1484	80 → 1143 [ACK] Seq=7372 Ack=961 Win=7504 Len=1430 [TCP segment of a reassembled PDU]
19	0.193548	131.247.95.216	64.233.161.99	TCP	54	1143 → 80 [ACK] Seq=961 Ack=7372 Win=17520 Len=0
20	0.193598	64.233.161.99	131.247.95.216	TCP	1484	80 → 1143 [ACK] Seq=8802 Ack=961 Win=7504 Len=1430 [TCP segment of a reassembled PDU]
21	0.193626	131.247.95.216	64.233.161.99	TCP	54	1143 → 80 [ACK] Seq=961 Ack=10232 Win=17520 Len=0
22	0.220099	64.233.161.99	131.247.95.216	HTTP	238	HTTP/1.1 200 OK (GIF89a)
23	0.260899	131.247.95.216	64.233.161.99	HTTP	477	GET /favicon.ico HTTP/1.1
24	0.294356	64.233.161.99	131.247.95.216	TCP	1484	80 → 1143 [ACK] Seq=10416 Ack=1384 Win=8576 Len=1430 [TCP segment of a reassembled PDU]
25	0.294500	64.233.161.99	131.247.95.216	HTTP	239	HTTP/1.1 200 OK (image/x-icon)
26	0.294600	131.247.95.216	64.233.161.99	TCP	54	1143 → 80 [ACK] Seq=1384 Ack=12031 Win=17520 Len=0

a) What is the IP address of the client that initiates the conversation?

- 131.247.95.216

b) Use the first two packets to identify the server that is going to be contacted.

List the common name, and three IP addresses that can be used for the server.

- Packet #1
 - Standard Query looking for the www.google.com IP

- Packet #2
 - Standard query response
 - CNAME: www.l.google.com
 - IP addresses:
 - **64.233.161.99**
 - **64.233.161.104**
 - **64.233.161.147**

c) What is happening in frames 3, 4, and 5?

- In frames 3, 4, and 5, the TCP three-way handshake is taking place to establish a connection between the client and the server.

d) What is happening in frames 6 and 7?

- These frames show the start of the HTTP communication, where the client requests the homepage from the server, and the server acknowledges that request.

e) Ignore frame eight. However, for your information, frame eight is used to manage flow control.

- Frame 8 is related to flow control, which manages data exchange between the client and server to prevent overwhelming either side. It updates the "TCP window" size, indicating how much data the receiver can handle. While not crucial for the main request/response cycle, it plays an important role in ensuring reliable data transmission.

f) What is happening in frames nine and ten? How are these two frames related?

- In frame 9, the server acknowledges receipt of data from the client, confirming it received all information up to a specific sequence number. In **frame 10**, the server responds to the client's HTTP GET request with an HTTP 200 OK status, indicating that the requested web page is available and includes its HTML content. These frames are connected, as frame 9 confirms the receipt of the client's request, while frame 10 provides the server's response to that request.

g) What happens in packet 11?

- In packet 11, the client sends an acknowledgment (ACK) back to the server, confirming that it has successfully received the HTTP response sent in packet 10. This acknowledgment indicates that the client is ready for the next segment of data or to continue the conversation, ensuring reliable communication between the client and server. The sequence number and acknowledgment number in this packet reflect the ongoing TCP session's state.

h) After the initial set of packets is received, the client sends out a new request in packet 12. This occurs automatically without any action by the user. Why does this occur? See the first "hint" to the left.

- In packet 12, the client sends a new HTTP GET request to the server, specifically requesting the image file located at `/intl/en/images/logo.gif`. This indicates that the client is interested in retrieving an additional resource from the server after successfully loading the initial webpage. The packet contains the necessary headers to inform the server about the client's capabilities and preferences for the request. The sequence number and acknowledgment number are updated to reflect the current state of the TCP session.

i) What is occurring in packets 13 through 22?

- I'm gonna list what's occurring:
 - **Packets 13 and 14:** The server responds to the client's request for the image file (`logo.gif`). It sends TCP segments that are parts of the image data back to the client. These packets include acknowledgment numbers that indicate the data received by the client, confirming the receipt of the initial request.

- **Packet 15:** The client sends an acknowledgment (ACK) back to the server, confirming that it has received the segments of the image data sent in the previous packets.
- **Packets 16 and 17:** The server continues to send additional segments of the image data to the client, updating the acknowledgment numbers to indicate the segments being sent.
- **Packet 18:** The server sends another segment of image data, continuing to update the acknowledgment numbers.
- **Packet 19:** The client sends another ACK back to the server, confirming the receipt of the latest segment of image data.
- **Packet 20:** The server sends another segment of data to the client, which is still part of the image file.
- **Packet 21:** The client sends another ACK confirming the receipt of the image data segment.
- **Packet 22:** The server sends an HTTP response indicating that it has successfully delivered the requested image data.
- Overall, this exchange represents the transfer of the image data from the server to the client, with the client sending acknowledgment packets to confirm receipt of each segment. The flow control and reliability mechanisms of TCP ensure that all data is sent and received correctly.

j) Explain what happens in packets 23 through 26. See the second “hint” to the left.

- I'm gonna list what's occurring:
 - **Packet 23:** The client sends a new HTTP request for the `favicon.ico` file to the server. The `favicon.ico` is a small icon that represents the website, usually displayed in the browser's address bar or tab.
 - **Packet 24:** The server responds to the client's request for the `favicon.ico` file. It sends a TCP segment containing data related to the requested icon, along with the acknowledgment of the previous data received from the client (indicated by the acknowledgment number).

- **Packet 25:** The server sends an HTTP response back to the client, indicating that it has successfully delivered the requested `favicon.ico` file. This response includes the appropriate headers and the data for the favicon.
- **Packet 26:** The client sends an acknowledgment (ACK) back to the server, confirming the receipt of the `favicon.ico` data sent in the previous packet.
- This exchange illustrates the client initiating a request for an additional resource, specifically the `favicon.ico` file, while the server replies with the requested data. The packets highlight the continuous communication between the client and the server, ensuring reliable data transfer through TCP acknowledgment processes.

k) In one sentence describe what the user was doing (Reading email? Accessing a web page? FTP? Other?).

- The user was accessing a web page, specifically loading resources from the Google website.

Exercise two

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.

- The cname (canonical name) of the website being looked up is www.yahoo.akadns.net.
 - 216.109.117.106
 - 216.109.117.109

b) How many packets/frames does it take to receive the web page (the answer to the first http get request only)?

- Record 6: The client sends an HTTP GET request to Yahoo's server for the webpage.
- Records 7 to 21: The server sends TCP segments of the webpage content (reassembled as part of the HTTP response).
- Record 22: The client receives the full HTTP response, with the HTTP/1.1 200 OK message, marking the successful transmission of the requested content.

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.

- Yes, the website use it

```
- Transmission Control Protocol, Src Port: 80, Dst Port: 1221,
+ [11 Reassembled TCP Segments (15661 bytes): #7(1460), #8(1460)
- Hypertext Transfer Protocol
+ HTTP/1.1 200 OK\r\n
  Date: Mon, 08 May 2006 19:59:42 GMT\r\n
  P3P: policyref="http://p3p.yahoo.com/w3c/p3p.xml", CP="CA0
  Cache-Control: private\r\n
  Vary: User-Agent\r\n
  Set-Cookie: FPB=ol1uquj7e125v8pe; expires=Thu, 01 Jun 2006
  Set-Cookie: D=_ylh=X3oDMTFmdXFnazJsBF9TAzI3MTYxNDkEc6lkAzE
  Connection: close\r\n
  Transfer-Encoding: chunked\r\n
  Content-Type: text/html\r\n
  Content-Encoding: gzip\r\n
  \r\n
[HTTP response 1/1]
[Time since request: 0.102872000 seconds]
```

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.

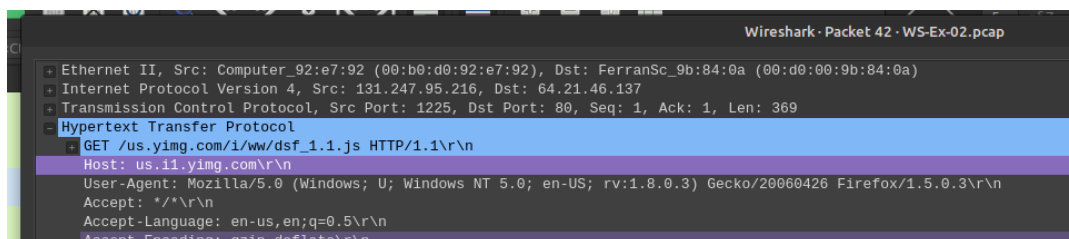
- Packets 26 and 27 are resolving the DNS for `us.js2.yimg.com`, which is a separate domain serving part of the content for the main web page. This demonstrates that web pages often rely on multiple servers for different components.

e) In packet 37 we see another DNS query, this time for `us.il.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.)

- Packet 26 resolves the IP for `us.js2.yimg.com` (likely for JavaScript content).
- Packet 37 resolves the IP for `us.il.yimg.com` (likely for other content like images).
- This is a common pattern in modern websites where different types of resources (scripts, images, stylesheets, etc.) are served from different subdomains, sometimes even via CDNs, to optimize load times. Therefore, separate DNS queries are needed for each distinct subdomain.

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.

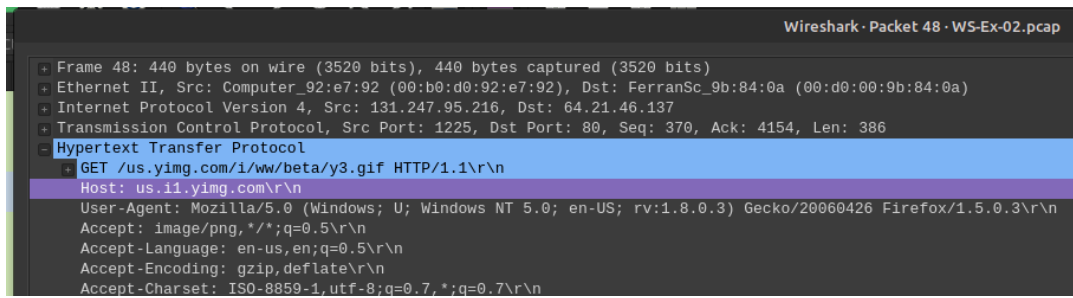
- Packet 42



The image shows a Wireshark packet capture window titled "Wireshark · Packet 42 · WS-Ex-02.pcap". The packet list on the left shows several protocols expanded: Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. The packet details pane on the right shows the structure of the HTTP request. The "Host:" field is highlighted in blue and contains the value "us.il.yimg.com\r\n". Other fields visible include "GET /us.yimg.com/i/ww/dsf_1.1.js HTTP/1.1\r\n", "User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.8.0.3) Gecko/20060426 Firefox/1.5.0.3\r\n", "Accept: */*\r\n", "Accept-Language: en-us,en;q=0.5\r\n", and "Accept-Encoding: gzip,deflate\r\n".

```
Wireshark · Packet 42 · WS-Ex-02.pcap
Ethernet II, Src: Computer_92:e7:92 (00:b0:d0:92:e7:92), Dst: FerranSc_9b:84:0a (00:d0:00:9b:84:0a)
Internet Protocol Version 4, Src: 131.247.95.216, Dst: 64.21.46.137
Transmission Control Protocol, Src Port: 1225, Dst Port: 80, Seq: 1, Ack: 1, Len: 369
Hypertext Transfer Protocol
  GET /us.yimg.com/i/ww/dsf_1.1.js HTTP/1.1\r\n
  Host: us.il.yimg.com\r\n
  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.8.0.3) Gecko/20060426 Firefox/1.5.0.3\r\n
  Accept: */*\r\n
  Accept-Language: en-us,en;q=0.5\r\n
  Accept-Encoding: gzip,deflate\r\n
```

- Packet 48

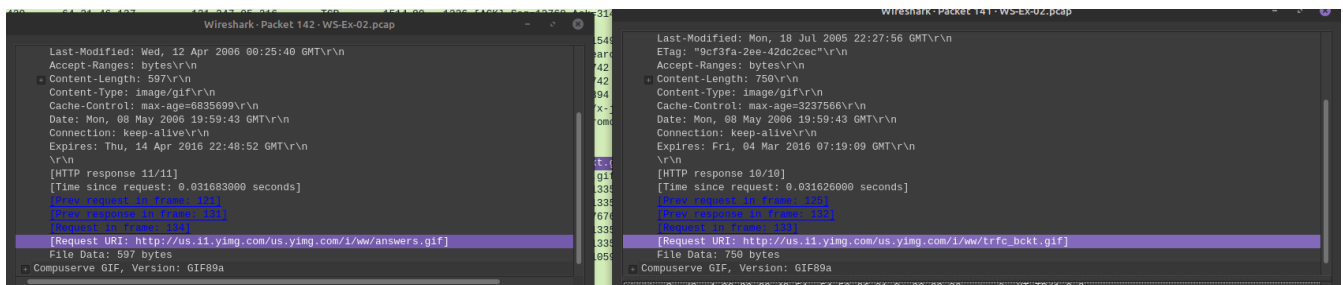


- Both packet 42 and packet 48 are HTTP GET requests to the **same host** (us.yimg.com), so no new DNS resolution is required.
- The system uses the cached IP address from the previous DNS lookup to continue fetching resources from us.yimg.com.

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?

- Packets 141, 142, and 143 are part of the reassembled PDU in packet 160, as they continue the sequence of data segments that began in packet 139. TCP handles such reassembly by collecting and ordering segments from multiple packets. If packets arrive out of order, TCP buffers them and waits for the missing segments to reassemble the complete message. If packets are lost, TCP requests retransmission from the sender, ensuring reliable data transfer, though this can introduce some delays as it waits for all necessary segments to arrive.

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?



- The client uses a combination of sequence numbers, destination ports, and content lengths, along with the specific URIs from the GET requests, to correctly identify and match each graphic response to its corresponding request. This ensures the proper display of images, even when multiple files are being requested in quick succession.

Exercise Three

a) Compare the destination port in the TCP packet in frame 3 with the destination port in the TCP packet in frame 12. What difference do you see? What does this tell you about the difference in the two requests?

- Frame 3: The destination port is 80, which is the standard port for HTTP traffic. This indicates that the request being made in this frame (from IP address 192.168.1.3 to 68.142.226.44) is for a regular HTTP connection.
- Frame 12: The destination port is 443, which is the standard port for HTTPS traffic. This indicates that the request being made in this frame (from IP address 192.168.1.3 to 131.247.100.94) is for a secure connection using SSL/TLS.

The following table compares the two requests for web pages. For example, row i) shows that frames 1-2 and frames 8-9 represent the DNS lookups for each of the web requests.

Row	www.yahoo.com frames	my.usf.com frames	Brief Explanation of Activity
i)	1-2	8-9	DNS Request to find IP address for common name & DNS Response
ii)	3-5	10-12	Three-way handshake
iii)	--	13-20	
iv)	6	21	"Get" request for web page
v)	7	22	First packet from web server with web page content.

b) Explain what is happening in row "iii" above. Why are there no frames listed for yahoo in row "iii"?

- In row "iii," frames 13-20 illustrate the SSL/TLS handshake process for my.usf.edu, where the client initiates a secure connection with a Client Hello message, followed by the server's Server Hello and certificate transmission, leading to the Client Key Exchange and the start of secure data transmission. There are no corresponding frames for www.yahoo.com because the initial request (frames 1-2) was made over HTTP, which does not require SSL/TLS, indicating a transition to a secure HTTPS connection with a different site. This reflects a common scenario where users move from a standard HTTP request to a secure HTTPS connection with another domain, generating no SSL/TLS frames for the first site.

c) Look at the “Info” column on frame 6. It says: “GET / HTTP / 1.1. What is the corresponding Info field for the my.usf.com web request (frame 21)? Why doesn't it read the same as in frame 6?

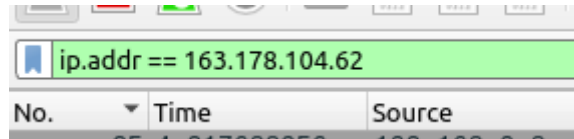
- The corresponding Info field for the my.usf.edu web request in frame 21 indicates "Application Data." This differs from the "GET / HTTP/1.1" in frame 6 because frame 21 is part of the secure SSL/TLS communication, where the HTTP request has already been sent and acknowledged. In a secure connection, the data is encrypted, so Wireshark displays it as "Application Data," reflecting that the actual HTTP request and responses are encapsulated within the secure SSL/TLS protocol. Therefore, while frame 6 shows a clear HTTP GET request, frame 21 does not display this information due to the encryption and the subsequent data exchange occurring after the handshake process.

Exercise Four

- For this exercise we need to run Wireshark as administrator, in my case:

```
mario@Mario-MonsterEXT-Linux:~$ sudo wireshark
[sudo] password for mario:
** (wireshark:14001) 07:35:55.640980 [GUI WARNING] -- QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

- I have visited "os.ecci.ucr.ac.cr/ci0121" and apply the filter:



- **Initial TCP Termination**
 - Frame 25: The client initiates TCP connection termination with a FIN, ACK packet.
 - Frame 27: The server responds with a RST packet, abruptly closing the connection.
- **New TCP Connections (SYN-SYN/ACK Handshake)**
 - Frames 29-32: Two new TCP connections start almost simultaneously. The client sends two SYN packets (frames 29 and 30) to initiate connections on ports 51344 and 51358. The server replies with SYN, ACK packets in frames 31 and 35.
 - Frames 32 and 36: The client acknowledges the server's response (ACK packets).
- **TLS Handshake**
 - Frame 34 and 38: The client initiates the TLS handshake with a Client Hello message over both connections.
 - Frame 40 and 45: The server responds with Server Hello, followed by Change Cipher Spec and Encrypted Handshake Messages.
- **Encrypted Data Transfer**
 - Frame 43 and 49: Application data is exchanged after the handshake, encrypted using TLS.
 - Frames 44-51: The connection is maintained through acknowledgments (ACK) and additional application data is sent.

- **Connection Closure**

- Frames 107-114: After some time, the server sends an Encrypted Alert (likely a close notify), and then sends a FIN, ACK to initiate the termination of the connection. The client responds with an ACK, completing the graceful closure of the session.