



Lab 1 INF-256: Wireshark

Macarena Hidalgo 201473608-8

David Medel 201573548-4

1. Referente a los mensajes realizados por las aplicaciones: ¿Qué tipos de protocolo espera ver? ¿Cuáles encontró? Justifique sus expectativas y las diferencias que encuentre

Se esperan ver los protocolos TCP, UDP y HTTP, debido a que dentro de la definición de los sockets solo ocupamos protocolos TCP y UDP, además de esperar HTTP debido a la conexión efectuada con la web.

Los protocolos que pudimos ver en el intercambio de paquetes de una consulta hacia la URL: "www.youtube.com", fueron exactamente algunos de los que esperábamos ver (TCP y HTTP), tal y como muestra la imagen.

| | | | | | | |
|----|-----------|--------------|---------------|---------|-----|--|
| 9 | 9.820804 | 192.168.0.10 | 190.54.120.23 | DNS | 73 | Standard query 0xa7ad A www.google.cl |
| 11 | 9.837070 | 192.168.0.10 | 64.233.190.94 | TCP | 78 | 52418 → 80 [SYN] Seq=0 Win=65535 Len=0 |
| 13 | 9.850932 | 192.168.0.10 | 64.233.190.94 | TCP | 66 | 52418 → 80 [ACK] Seq=1 Ack=1 Win=13184 |
| 14 | 9.850999 | 192.168.0.10 | 64.233.190.94 | HTTP | 84 | GET / HTTP/1.1 |
| 17 | 9.901933 | 192.168.0.10 | 64.233.190.94 | TCP | 66 | 52418 → 80 [FIN, ACK] Seq=19 Ack=1419 |
| 20 | 9.902801 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=19 Win=0 Len=0 |
| 21 | 9.902835 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=19 Win=0 Len=0 |
| 23 | 9.903096 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=19 Win=0 Len=0 |
| 26 | 9.903367 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=19 Win=0 Len=0 |
| 27 | 9.903367 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=19 Win=0 Len=0 |
| 31 | 9.904900 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=19 Win=0 Len=0 |
| 32 | 9.904901 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=19 Win=0 Len=0 |
| 33 | 9.904920 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=19 Win=0 Len=0 |
| 35 | 9.905204 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=19 Win=0 Len=0 |
| 37 | 9.915703 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=20 Win=0 Len=0 |
| 39 | 9.916430 | 192.168.0.10 | 64.233.190.94 | TCP | 54 | 52418 → 80 [RST] Seq=20 Win=0 Len=0 |
| 40 | 10.000202 | 192.168.0.10 | 18.229.250.79 | TLSv1.2 | 122 | Application Data |

Figura 1: Ruta de intercambio de paquetes para www.youtube.com visualizado en Wireshark

La razón por la cual no observamos el protocolo UDP, es porque el socket que definimos para trabajar con dicho protocolo solo se encarga de hacer el envío de paquetes desde nuestro servidor a cliente, por lo cual no forma parte del proceso de comunicación entre el servidor del servicio web y nuestro servidor (que en este proceso actuaría como el cliente).

Podemos observar que existen dos protocolos que no se esperaban ver los cuales son: protocolo DNS y protocolo TLSv1.2

El protocolo DNS, es un protocolo que sirve para recordar los nombres de dominio en lugar de las IP, haciendo que los usuarios accedan más fácilmente a las webs, en otras palabras es el encargado de convertir las secuencias numéricas a nombres inteligibles que se asocian a entidades, marcas, personas o a los servicios que brindan.

Por otro lado aparece el protocolo TLSv1.2 (Transport Layer Security), este protocolo es un protocolo criptográfico que garantiza las comunicaciones en Internet, es decir permite y garantiza el



intercambio de datos en un entorno seguro y privado entre dos entes, mediante aplicaciones como HTTP, POP3, IMAP, SSH, SMTP o NNTP. En otras palabras básicamente lo que hace o permite este protocolo es encriptar la información compartida.

2. Las interacciones vía TCP entre el cliente y el servidor, ¿deben ocupar los mismos puertos a lo largo del tiempo? ¿Coincide con lo visto en Wireshark? Fundamente.

Debemos recordar que TCP es un protocolo orientado en la conexión, el cual genera un handshake para determinar o fijar las comunicaciones de principio a fin, dado esto solo cuando la conexión sea determinada, los datos del usuario pueden ser mandados de modo bidireccional por la conexión.

Asumiendo que nuestro servidor actuara como cliente y que el servidor del servicio web actuara como servidor, podemos notar que lo estipulado en el párrafo anterior si coincide, lo que queda en evidencia en la columna de información de la imagen adjunta, donde se genera una comunicación entre el puerto 80 y el puerto 64891.

Además de lo anterior es importante señalar que el puerto TCP 64891 garantiza la entrega de paquetes de datos en el mismo orden en que fueron mandados.

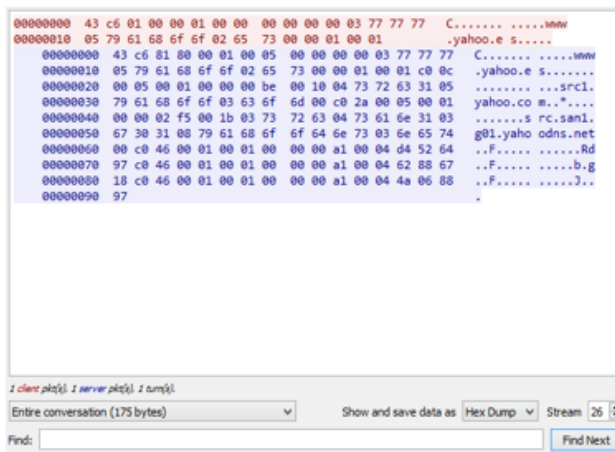
| tcp.stream eq 10 | | | | | | |
|------------------|-----------|---------------|---------------|----------|--------|--|
| No. | Time | Source | Destination | Protocol | Length | Info |
| 1496 | 14.077013 | 192.168.0.14 | 98.136.103.24 | TCP | 66 | 64891 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 |
| 1524 | 14.283998 | 98.136.103.24 | 192.168.0.14 | TCP | 66 | 80 → 64891 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128 |
| 1525 | 14.284072 | 192.168.0.14 | 98.136.103.24 | TCP | 54 | 64891 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 |
| 1526 | 14.284121 | 192.168.0.14 | 98.136.103.24 | HTTP | 72 | GET / HTTP/1.1 |
| 1552 | 14.495594 | 98.136.103.24 | 192.168.0.14 | TCP | 56 | 80 → 64891 [ACK] Seq=1 Ack=19 Win=29312 Len=0 |
| 1556 | 14.497871 | 98.136.103.24 | 192.168.0.14 | TCP | 1514 | 80 → 64891 [ACK] Seq=1 Ack=19 Win=29312 Len=1460 [TCP segment of a reassembled PDU] |
| 1557 | 14.497871 | 98.136.103.24 | 192.168.0.14 | TCP | 1514 | 80 → 64891 [ACK] Seq=1461 Ack=19 Win=29312 Len=1460 [TCP segment of a reassembled PDU] |
| 1558 | 14.497872 | 98.136.103.24 | 192.168.0.14 | TCP | 1514 | 80 → 64891 [ACK] Seq=2921 Ack=19 Win=29312 Len=1460 [TCP segment of a reassembled PDU] |
| 1559 | 14.497873 | 98.136.103.24 | 192.168.0.14 | HTTP | 558 | HTTP/1.1 200 OK (text/html) |
| 1560 | 14.497974 | 192.168.0.14 | 98.136.103.24 | TCP | 54 | 64891 → 80 [ACK] Seq=19 Ack=4885 Win=65536 Len=0 |
| 1561 | 14.498033 | 192.168.0.14 | 98.136.103.24 | TCP | 54 | 64891 → 80 [RST, ACK] Seq=19 Ack=4885 Win=0 Len=0 |

Figura 2: Ruta de intercambio de paquetes para www.yahoo.es visualizado en Wireshark

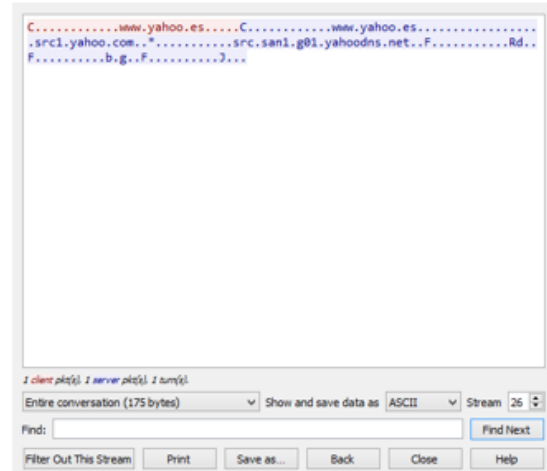


3. Los contenidos de los mensajes enviados entre las aplicaciones, ¿son legibles?

No, ya que la comunicación que se genera entre servidor y web es siempre codificada, por lo cual no es legible.



(1) Mensaje codificado en hexadecimal

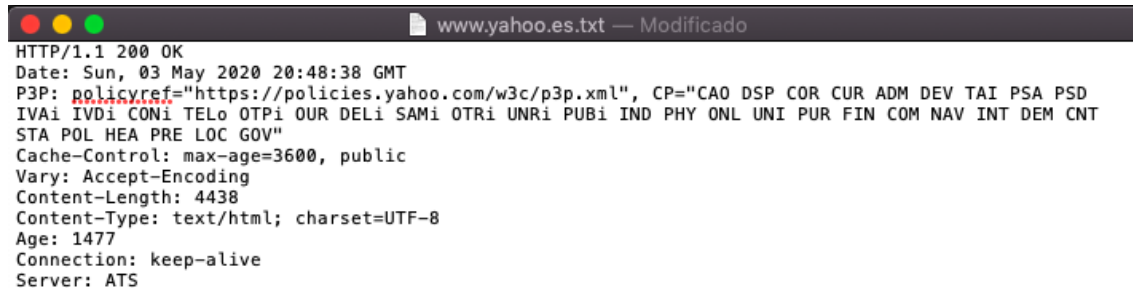


(2) Mensaje codificado en ASCII

Figura 3: Diferente codificación para el mismo mensaje visualizado en Wireshark

4. Encuentre la respuesta a la consulta HTTP recibida por el servidor, ¿el header es igual al almacenado por el cliente, o existe alguna diferencia importante? Explique.

A continuación se presenta la respuesta almacenada en el documento de texto creado por el cliente de la consulta http a la web **www.yahoo.es**.



```
HTTP/1.1 200 OK
Date: Sun, 03 May 2020 20:48:38 GMT
P3P: policyref="https://policies.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PSD
IVAI IVDI CONI TELI OTPI OUR DELI SAMI OTRI UNRI PUBI IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT
STA POL HEA PRE LOC GOV"
Cache-Control: max-age=3600, public
Vary: Accept-Encoding
Content-Length: 4438
Content-Type: text/html; charset=UTF-8
Age: 1477
Connection: keep-alive
Server: ATS
```

Figura 4: Documento de texto con respuesta obtenida por el cliente

Por otro lado se accede a la respuesta de la consulta HTTP obtenida por el servidor mediante Wireshark resultando lo siguiente.



```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      [HTTP/1.1 200 OK\r\n]
      [Severity level: Chat]
      [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
    Date: Sun, 03 May 2020 20:13:41 GMT\r\n
    P3P: policyref="https://policies.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PSD IVAI IVDI CONI
    Cache-Control: max-age=3600, public\r\n
    Vary: Accept-Encoding\r\n
    Content-Length: 4438\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    Age: 986\r\n
    Connection: keep-alive\r\n
    Server: ATS\r\n
```

Figura 5: Respuesta recibida por el servidor visible en wireshark

Al observar las figuras 4 y 5 se puede ver que los headers son idénticos, con la particularidad que el header del servidor entregado por Wireshark da la posibilidad de explorar *Expert Info*, donde se detalla información como por ejemplo, la *severity level*, que corresponde principalmente a una clasificación nivel de gravedad para permitir que los usuarios novatos y expertos encuentren problemas de red más rápido que escanear manualmente a través de la lista de paquetes. Para mayor información respecto a esto se puede observar el siguiente [enlace](#).

Por otro lado al momento de ejecutar y hacer pruebas en la creación de los archivos de servidor y cliente, podemos observar que la respuesta de la web hacia el servidor siempre viene codificada, la cual se la enviamos al cliente y decodificamos para que el usuario pueda visualizar su contenido.



5. Bonus

Nuestra interacción cliente-servidor permite la conexión de múltiple clientes consultando al mismo tiempo al servidor, esto pues el servidor siempre permanece escuchando en el mismo puerto.

A continuación se detalle un ejemplo de la conexión entre dos clientes diferentes con el servidor, especificando en la figura 8 las consultas http que realizó el servidor con lo solicitado por los diversos clientes.

```
Servidor TCP escuchando en puerto 65010
```

Figura 6: Servidor esperando en el puerto 65010

```
MacBook-Air-de-David:lab1_redes dmedel$ python3 cliente.py
Ingrese la URL: www.google.cl
Ingrese la URL: www.gmail.com
Ingrese la URL: www.bancoestado.cl
Ingrese la URL: terminate
MacBook-Air-de-David:lab1_redes dmedel$

MacBook-Air-de-David:lab1_redes dmedel$ python3 cliente.py
Ingrese la URL: www.facebook.com
Ingrese la URL: www.twitter.com
Ingrese la URL: terminate
MacBook-Air-de-David:lab1_redes dmedel$
```

Figura 7: Consultas realizadas por el cliente 1 y 2.

| http | | | | | | |
|------|-----------|----------------|----------------|----------|--------|--------------------------------------|
| No. | Time | Source | Destination | Protocol | Length | Info |
| 41 | 13.783937 | 192.168.0.10 | 172.217.192.94 | HTTP | 84 | GET / HTTP/1.1 |
| 302 | 26.418910 | 192.168.0.10 | 172.217.192.18 | HTTP | 84 | GET / HTTP/1.1 |
| 331 | 26.516868 | 192.168.0.10 | 157.240.204.35 | HTTP | 84 | GET / HTTP/1.1 |
| 333 | 26.723064 | 157.240.204.35 | 192.168.0.10 | HTTP | 392 | HTTP/1.1 301 Moved Permanently |
| 499 | 36.508470 | 192.168.0.10 | 23.74.92.87 | HTTP | 84 | GET / HTTP/1.1 |
| 501 | 36.689430 | 23.74.92.87 | 192.168.0.10 | HTTP | 485 | HTTP/1.0 400 Bad Request (text/html) |
| 514 | 36.892544 | 192.168.0.10 | 104.244.42.129 | HTTP | 84 | GET / HTTP/1.1 |
| 516 | 37.066174 | 104.244.42.129 | 192.168.0.10 | HTTP | 216 | HTTP/1.1 404 Not Found |

172.217.192.94 : google.cl
157.240.204.35: facebook.com
23.74.92.87 : bancoestado.cl
104.244.42.129: twitter.com

Figura 8: Visualización consultas HTTP realizadas por el servidor en Wireshark