Universidad Técnica Federico Santa María
Campus Santiago San Joaquín
Departamento de Informática
1° semestre 2020

# Laboratorio 1

Redes de Computadores
Profesora: Erika Rosas Olivos
Joaquín Castillo y María Paz Morales

3 de mayo de 2020

## 1. Wireshark

Las preguntas sobre *Wireshark* se contestan en orden a continuación:

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 espera encontrar sólo tres tipos de protocolo: el protocolo HTTP de la capa de aplicación y los protocolos de TCP y UDP de la capa de transporte.

   En el análisis de las aplicaciones a través de WireShark se encontraron efectivamente los tres tipos de protocolos esperados. Sin embargo, también se observaron los protocolos usados en la Capa de Red y en la Capa de Enlace para los mensajes enviados entre aplicaciones, y el protocolo usado en la Capa Física para la petición HTTP.

   En la Figura 1 se muestra la transferencia de mensajes entre el cliente y el servidor, donde se puede ver que el flujo de mensajes entre las aplicaciones efectivamente utiliza protocolos TCP y UDP.

| | | | | | |
|---|---|---|---|---|---|
| 12 | 10.298894 | 127.0.0.1 | 127.0.0.1 | TCP | 108 57701 → 50366 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SA… |
| 13 | 10.298928 | 127.0.0.1 | 127.0.0.1 | TCP | 108 50366 → 57701 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=6549… |
| 14 | 10.298947 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57701 → 50366 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 44 | 20.246732 | 127.0.0.1 | 127.0.0.1 | TCP | 98 57701 → 50366 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=14 |
| 45 | 20.246756 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57701 [ACK] Seq=1 Ack=15 Win=2619648 Len=0 |
| 46 | 20.306376 | 127.0.0.1 | 127.0.0.1 | TCP | 89 50366 → 57701 [PSH, ACK] Seq=1 Ack=15 Win=2619648 Len=5 |
| 47 | 20.306416 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57701 → 50366 [ACK] Seq=15 Ack=6 Win=2619648 Len=0 |
| 48 | 20.307016 | 127.0.0.1 | 127.0.0.1 | UDP | 62 64484 → 50367 Len=2 |
| 49 | 20.307125 | 127.0.0.1 | 127.0.0.1 | UDP | 775 50367 → 64484 Len=715 |
| 78 | 39.085353 | 127.0.0.1 | 127.0.0.1 | TCP | 93 57701 → 50366 [PSH, ACK] Seq=15 Ack=6 Win=2619648 Len=9 |
| 79 | 39.085387 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57701 [ACK] Seq=6 Ack=24 Win=2619648 Len=0 |
| 80 | 39.085715 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57701 → 50366 [FIN, ACK] Seq=24 Ack=6 Win=2619648 Len=0 |
| 81 | 39.085734 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57701 [ACK] Seq=6 Ack=25 Win=2619648 Len=0 |
| 82 | 39.086937 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57701 [FIN, ACK] Seq=6 Ack=25 Win=2619648 Len=0 |
| 83 | 39.086983 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57701 → 50366 [ACK] Seq=25 Ack=7 Win=2619648 Len=0 |
| 101 | 44.349358 | 127.0.0.1 | 127.0.0.1 | TCP | 108 57704 → 50366 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SA… |
| 102 | 44.349409 | 127.0.0.1 | 127.0.0.1 | TCP | 108 50366 → 57704 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=6549… |
| 103 | 44.349432 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57704 → 50366 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 135 | 54.485241 | 127.0.0.1 | 127.0.0.1 | TCP | 97 57704 → 50366 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=13 |
| 136 | 54.485279 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57704 [ACK] Seq=1 Ack=14 Win=2619648 Len=0 |
| 137 | 54.602551 | 127.0.0.1 | 127.0.0.1 | TCP | 89 50366 → 57704 [PSH, ACK] Seq=1 Ack=14 Win=2619648 Len=5 |
| 138 | 54.602581 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57704 → 50366 [ACK] Seq=14 Ack=6 Win=2619648 Len=0 |
| 139 | 54.603113 | 127.0.0.1 | 127.0.0.1 | UDP | 62 61421 → 50367 Len=2 |
| 140 | 54.603212 | 127.0.0.1 | 127.0.0.1 | UDP | 473 50367 → 61421 Len=413 |
| 146 | 59.141313 | 127.0.0.1 | 127.0.0.1 | TCP | 93 57704 → 50366 [PSH, ACK] Seq=14 Ack=6 Win=2619648 Len=9 |
| 147 | 59.141339 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57704 [ACK] Seq=6 Ack=23 Win=2619648 Len=0 |
| 148 | 59.141757 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57704 → 50366 [FIN, ACK] Seq=23 Ack=6 Win=2619648 Len=0 |
| 149 | 59.141785 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57704 [ACK] Seq=6 Ack=24 Win=2619648 Len=0 |

Figura 1: Mensajes entre aplicaiones Cliente-Servidor.

En la Figuras 2 y 3 a continuación, se muestran datos de dos frames enviados entre aplicaiones, uno con protocolo TCP y el otro con protocolo UDP. En la sección *Protocols in frame* se pueden ver los protocolos utilizados en todas las capas involucradas de la transferencia de mensajes.

```
Frame 44: 98 bytes on wire (784 bits), 58 bytes captured (464 bits) on interface \Device\NPF_Loopback, id 0
  Interface id: 0 (\Device\NPF_Loopback)
  Encapsulation type: NULL/Loopback (15)
  Arrival Time: May  1, 2020 22:43:35.644726000 Hora est. Sudamérica Pacífico
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1588387415.644726000 seconds
  [Time delta from previous captured frame: 0.001891000 seconds]
  [Time delta from previous displayed frame: 9.947785000 seconds]
  [Time since reference or first frame: 20.246732000 seconds]
  Frame Number: 44
  Frame Length: 98 bytes (784 bits)
  Capture Length: 58 bytes (464 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: null:ip:tcp:data]
  [Coloring Rule Name: TCP]
  [Coloring Rule String: tcp]
```

Figura 2: Información de un frame mediante protocolo TCP.

```
Frame 49: 775 bytes on wire (6200 bits), 747 bytes captured (5976 bits) on interface \Device\NPF_Loopback, id 0
  Interface id: 0 (\Device\NPF_Loopback)
  Encapsulation type: NULL/Loopback (15)
  Arrival Time: May  1, 2020 22:43:35.705119000 Hora est. Sudamérica Pacífico
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1588387415.705119000 seconds
  [Time delta from previous captured frame: 0.000109000 seconds]
  [Time delta from previous displayed frame: 0.000109000 seconds]
  [Time since reference or first frame: 20.307125000 seconds]
  Frame Number: 49
  Frame Length: 775 bytes (6200 bits)
  Capture Length: 747 bytes (5976 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: null:ip:udp:data]
  [Coloring Rule Name: UDP]
  [Coloring Rule String: udp]
```

Figura 3: Información de un frame mediante protocolo UDP.

Se puede observar que ambos protocolos de transporte utilizan *Internet Protocol* (IP) en su Capa de Red, especificamente en su versión 4. Además, la Capa de Enlace no presenta protocolo, ya que esta conexión se hace directamente en la red local, es decir, que ambas aplicaciones están en el mismo *host*.

En la Figura 4 a continuación, se presentan los datos del frame correspondiente a una de las consultas HTTP y los protocolos de las distintas capas. En la Capa de Enlace se utiliza el protocolo Ethernet II, el protocolo IPv4 en su Capa de Red, protocolo TCP en su Capa de Transporte y protocolo HTTP en su capa de aplicación.

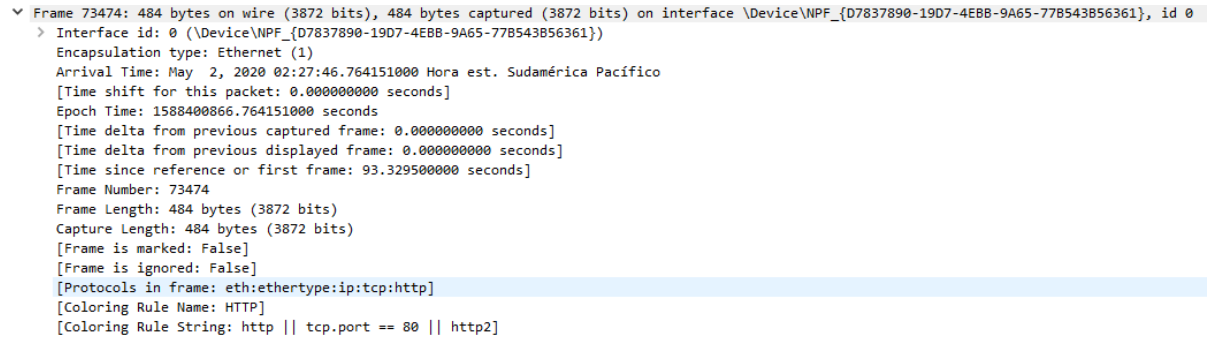



Figura 4: Información de un frame de consulta HTTP.

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.

   Para una conexión de cliente con servidor, sin que el cliente termine la comunicación con "*terminate*", las interacciones vía TCP entre ellos sí ocupan los mismos puertos a lo largo del tiempo. Esto coincide con lo visto en *Wireshark*, ya que desde el primer *handshake* hasta que el cliente corta la comunicación siempre se envían mensajes y se reciben por los mismos puertos asignados al momento que crear sus *sockets* correspondientes. Sin embargo, el puerto del cliente es diferente al puerto del servidor: el puerto del servidor es definido por nosotros al momento de crear el *socket* en el programa. En cambio, el puerto del cliente es diferente y es asignado dependiendo de los disponibles en el momento, y entre clientes varía. Entonces, el puerto del servidor y el puerto del cliente son diferentes, pero se mantiene cada uno durante toda la comunicación vía TCP.

   En la figura 5 a continuación se aprecia que en toda las interacciones TCP entre el cliente y el servidor, el cliente envía y recibe en el puerto 57976, y el servidor en el puerto 50366, independiente de que se hayan realizado muchas consultas de *headers*.

| | | | | | |
|---|---|---|---|---|---|
| 1284… | 3191.981703 | 127.0.0.1 | 127.0.0.1 | TCP | 108 57976 → 50366 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SA… |
| 1284… | 3191.981755 | 127.0.0.1 | 127.0.0.1 | TCP | 108 50366 → 57976 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=6549… |
| 1284… | 3191.981807 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57976 → 50366 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 1284… | 3199.500803 | 127.0.0.1 | 127.0.0.1 | TCP | 97 57976 → 50366 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=13 |
| 1284… | 3199.500839 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57976 [ACK] Seq=1 Ack=14 Win=2619648 Len=0 |
| 1284… | 3199.564714 | 127.0.0.1 | 127.0.0.1 | TCP | 89 50366 → 57976 [PSH, ACK] Seq=1 Ack=14 Win=2619648 Len=5 |
| 1284… | 3199.564748 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57976 → 50366 [ACK] Seq=14 Ack=6 Win=2619648 Len=0 |
| 1284… | 3199.565261 | 127.0.0.1 | 127.0.0.1 | UDP | 62 59398 → 50367 Len=2 |
| 1284… | 3199.565370 | 127.0.0.1 | 127.0.0.1 | UDP | 773 50367 → 59398 Len=713 |
| 1285… | 3234.188467 | 127.0.0.1 | 127.0.0.1 | TCP | 97 57976 → 50366 [PSH, ACK] Seq=14 Ack=6 Win=2619648 Len=13 |
| 1286… | 3234.188490 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57976 [ACK] Seq=6 Ack=27 Win=2619648 Len=0 |
| 1286… | 3234.304430 | 127.0.0.1 | 127.0.0.1 | TCP | 89 50366 → 57976 [PSH, ACK] Seq=6 Ack=27 Win=2619648 Len=5 |
| 1286… | 3234.304463 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57976 → 50366 [ACK] Seq=27 Ack=11 Win=2619648 Len=0 |
| 1286… | 3234.304959 | 127.0.0.1 | 127.0.0.1 | UDP | 62 64959 → 50367 Len=2 |
| 1286… | 3234.305067 | 127.0.0.1 | 127.0.0.1 | UDP | 473 50367 → 64959 Len=413 |
| 1286… | 3244.789615 | 127.0.0.1 | 127.0.0.1 | TCP | 97 57976 → 50366 [PSH, ACK] Seq=27 Ack=11 Win=2619648 Len=13 |
| 1286… | 3244.789653 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57976 [ACK] Seq=11 Ack=40 Win=2619648 Len=0 |
| 1286… | 3244.789935 | 127.0.0.1 | 127.0.0.1 | TCP | 89 50366 → 57976 [PSH, ACK] Seq=11 Ack=40 Win=2619648 Len=5 |
| 1286… | 3244.789947 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57976 → 50366 [ACK] Seq=40 Ack=16 Win=2619648 Len=0 |
| 1286… | 3244.790352 | 127.0.0.1 | 127.0.0.1 | UDP | 62 64960 → 50367 Len=2 |
| 1286… | 3244.790434 | 127.0.0.1 | 127.0.0.1 | UDP | 773 50367 → 64960 Len=713 |
| 1286… | 3271.692956 | 127.0.0.1 | 127.0.0.1 | TCP | 93 57976 → 50366 [PSH, ACK] Seq=40 Ack=16 Win=2619648 Len=9 |
| 1286… | 3271.692972 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57976 [ACK] Seq=16 Ack=49 Win=2619648 Len=0 |
| 1286… | 3271.692995 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57976 → 50366 [FIN, ACK] Seq=49 Ack=16 Win=2619648 Len=0 |
| 1286… | 3271.693003 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57976 [ACK] Seq=16 Ack=50 Win=2619648 Len=0 |
| 1287… | 3271.693401 | 127.0.0.1 | 127.0.0.1 | TCP | 84 50366 → 57976 [FIN, ACK] Seq=16 Ack=50 Win=2619648 Len=0 |
| 1287… | 3271.693417 | 127.0.0.1 | 127.0.0.1 | TCP | 84 57976 → 50366 [ACK] Seq=50 Ack=17 Win=2619648 Len=0 |

Figura 5: Interacciones vía TCP entre un cliente y un servidor vistas en *Wireshark*.

Cabe destacar que entre medio se ocupan otros puertos, pero esos se corresponden con las interacciones UDP para responder con el *header* solicitado, que están dentro de la comunicación TCP. Por lo que no se consideran como interacciones vía TCP sino UDP.

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

   Los contenidos de los mensajes enviados entre las aplicaciones no son logibles, ya que están codificados. Se codifican al momento de ser enviados vía cualquiera de los dos protocolos (TCP y UDP), y son decodificados al momento de ser recibidos y utilizados según corresponda.

   Se puede ver en *Wireshark* para cada mensaje enviado, cómo luce codificado. En la figura 6 a continuación se muestra un ejemplo de un mensaje enviado entre un cliente y un servidor; a la izquierda se ve cómo es el mensaje real enviado (un *header* enviado desde el servidor al cliente), que está codificado y no es legible. A la derecha sale cómo se vería decodificado.

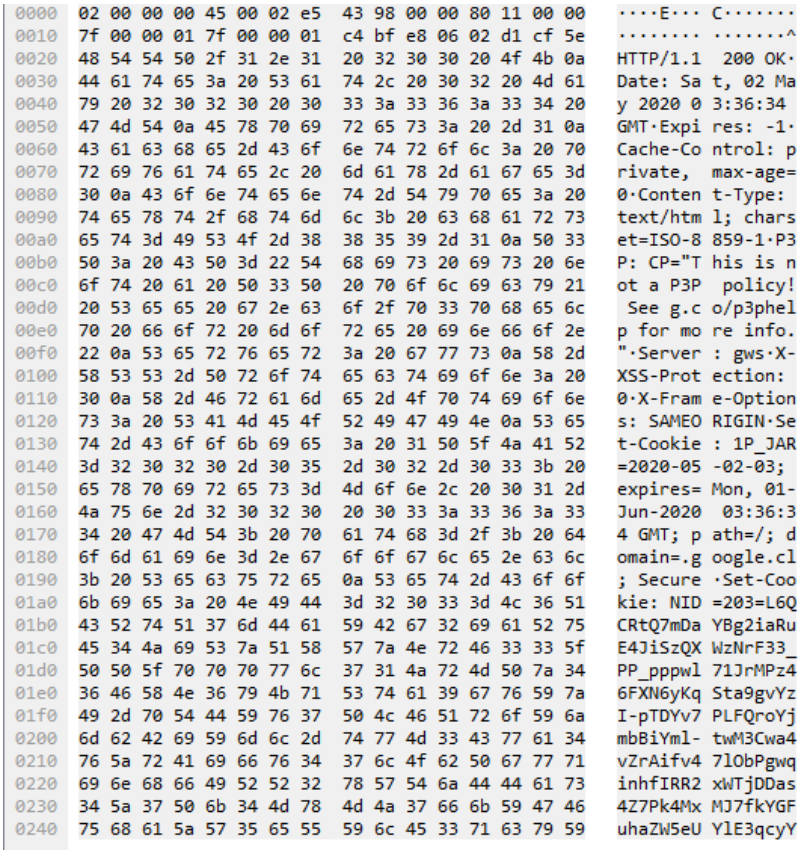



Figura 6: Mensaje codificado enviado entre un cliente y un servidor visto en *Wirechark*.

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.

   El *header* recibido directamente por el servidor desde la consulta HTTP y el almacenado por el cliente (que se puede ver en el archivo "URL.txt" correspondiente a la consulta) no presentan diferencias muy significativas. Hay una leve pérdida de información al traspasar el *header* desde el servidor al cliente, probeblemente por temas de conversión de formatos, pero no es una pérdida importante.

   A continuación se muestran las figuras 7 y 8 correspondientes a la misma consulta de un cliente a un servidor; en la figura 7 se ve lo obtenido por el servidor directamente de la consulta HTTP, y en la figura 8 lo almacenado por el cliente en el archivo *www.twitch.tv*.txt. Se puede observar que la única diferencia en la información de ambos archivos es la sección *Expert Info* que contiene información más detallada sobre la secuencia HTTP. Sin embargo, toda la información relevante del *header* es recibida y almacenada por el cliente.

```
Hypertext Transfer Protocol
  HTTP/1.1 301 Moved Permanently\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 301 Moved Permanently\r\n]
        [HTTP/1.1 301 Moved Permanently\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Response Version: HTTP/1.1
      Status Code: 301
      [Status Code Description: Moved Permanently]
      Response Phrase: Moved Permanently
  Server: Varnish\r\n
  Retry-After: 0\r\n
  Location: https://www.twitch.tv/\r\n
  Content-Length: 0\r\n
      [Content length: 0]
  Accept-Ranges: bytes\r\n
  Date: Sat, 02 May 2020 06:27:46 GMT\r\n
  Via: 1.1 varnish\r\n
  Connection: close\r\n
  X-Backend: 12jz6zqSzygLMoGmOwFUBI--F_go_twitch_tv\r\n
  Set-Cookie: twitch.lohp.countryCode=CL; domain=.twitch.tv; expires=Tue, 30 Apr 2030 06:27:46 GMT;\r\n
  X-Served-By: cache-scl19426-SCL\r\n
  X-Cache: HIT\r\n
  X-Cache-Hits: 0\r\n
```

Figura 7: *Header* recibido por un servidor a partir de una consulta HTTP visto en *Wirechark*.

```
HTTP/1.1 301 Moved Permanently
Server: Varnish
Retry-After: 0
Location: https://www.twitch.tv/
Content-Length: 0
Accept-Ranges: bytes
Date: Sat, 02 May 2020 06:27:46 GMT
Via: 1.1 varnish
Connection: close
X-Backend: 12jz6zqSzygLMoGmOwFUBI--F_go_twitch_tv
Set-Cookie: twitch.lohp.countryCode=CL; domain=.twitch.tv; expires=Tue, 30 Apr 2030 06:27:46 GMT;
X-Served-By: cache-scl19426-SCL
X-Cache: HIT
X-Cache-Hits: 0
```

Figura 8: *Header* recibido por un cliente desde un servidor, escrito por el cliente en un archivo de texto.

## 2. Desafío

Modifique el servidor para que atienda a más de un cliente al mismo tiempo. Explique diferencias importantes entre las interacciones de un cliente y otro atendidos al mismo tiempo.

Para implementar un servidor que atienda a múltiples clientes a la vez se utilizó la librería *threading* de *Python*, que permite implementar hebras. Lo que se hizo fue que el servidor estuviese corriendo y por cada cliente que se

conectara a él, generaba una nueva hebra para manejar todas las consultas correspondientes. También se hizo uso de un *lock*, de la misma librería, para, al momento de que un cliente terminara de consultar (pusiese *terminate*), se actualizara el archivo *cache.txt*. Al ingresar al archivo y escribirlo nuevamente, se protegió con un *lock* para que no ocurriese el error de que se intentara acceder a él al mismo tiempo desde dos hebras distintas.

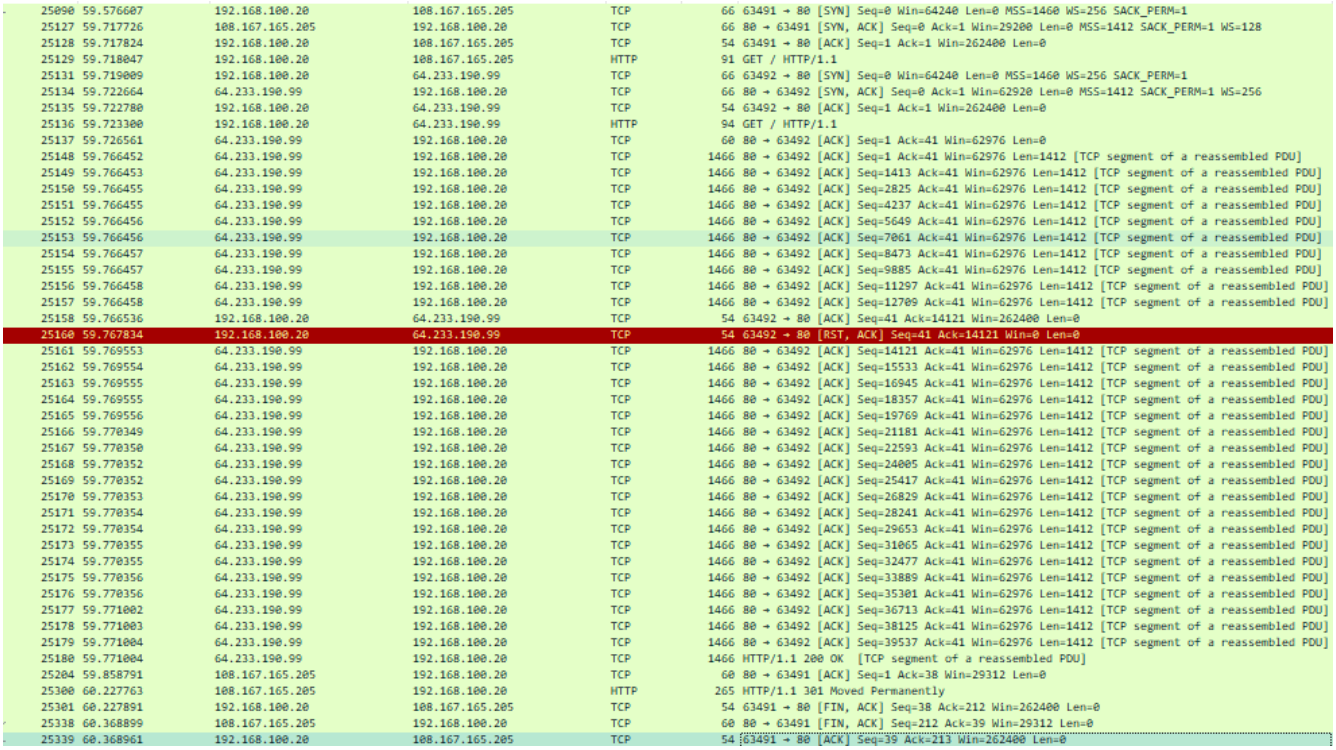
La diferencia más imporante que se presenta cuando un servidor atiende a muchos clientes al mismo tiempo es que el orden en que se realizan y responden las consultas puede ir variando; si un cliente 1 envía una consulta al mismo tiempo (o un tiempo muy cercano) a un cliente 2 que envía otra consulta, puede que el servidor primero haga la petición del cliente 2 antes que la del cliente 1, incluso cuando el cliente 1 fue el primero en enviar una consulta. Un ejemplo de esto se puede ver en la figura 9 a continuación, sacada de *Wireshark*, en que dos clientes son atendidos por el mismo servidor al mismo tiempo. En el ejemplo, el cliente 1 tiene asignado el puerto 62876, el cliente 2 el puerto 62880 y el servidor el puerto 50366. Se puede ver claramente que el cliente 2 realizó la consulta antes que el cliente 1, pero que el servidor les va respondiendo de forma intermitente a cada uno.

| | | | | | | |
|---|---|---|---|---|---|---|
| 94 | 51.817712 | 127.0.0.1 | 127.0.0.1 | TCP | 108 | 62876 → 50366 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SA… |
| 95 | 51.817747 | 127.0.0.1 | 127.0.0.1 | TCP | 108 | 50366 → 62876 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=6549… |
| 96 | 51.817767 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 62876 → 50366 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 159 | 80.663291 | 127.0.0.1 | 127.0.0.1 | TCP | 108 | 62880 → 50366 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SA… |
| 160 | 80.663621 | 127.0.0.1 | 127.0.0.1 | TCP | 108 | 50366 → 62880 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=6549… |
| 161 | 80.663955 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 62880 → 50366 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 277 | 116.378240 | 127.0.0.1 | 127.0.0.1 | TCP | 95 | 62880 → 50366 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=11 |
| 278 | 116.378275 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 50366 → 62880 [ACK] Seq=1 Ack=12 Win=2619648 Len=0 |
| 279 | 116.562792 | 127.0.0.1 | 127.0.0.1 | TCP | 98 | 62876 → 50366 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=14 |
| 280 | 116.562822 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 50366 → 62876 [ACK] Seq=1 Ack=15 Win=2619648 Len=0 |
| 281 | 116.615782 | 127.0.0.1 | 127.0.0.1 | TCP | 89 | 50366 → 62876 [PSH, ACK] Seq=1 Ack=15 Win=2619648 Len=5 |
| 282 | 116.615828 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 62876 → 50366 [ACK] Seq=15 Ack=6 Win=2619648 Len=0 |
| 283 | 116.616331 | 127.0.0.1 | 127.0.0.1 | UDP | 62 | 56347 → 50367 Len=2 |
| 284 | 116.616428 | 127.0.0.1 | 127.0.0.1 | UDP | 775 | 50367 → 56347 Len=715 |
| 287 | 117.020726 | 127.0.0.1 | 127.0.0.1 | TCP | 89 | 50366 → 62880 [PSH, ACK] Seq=1 Ack=12 Win=2619648 Len=5 |
| 288 | 117.020740 | 127.0.0.1 | 127.0.0.1 | TCP | 84 | 62880 → 50366 [ACK] Seq=12 Ack=6 Win=2619648 Len=0 |
| 289 | 117.021354 | 127.0.0.1 | 127.0.0.1 | UDP | 62 | 56348 → 50367 Len=2 |
| 290 | 117.021469 | 127.0.0.1 | 127.0.0.1 | UDP | 260 | 50367 → 56348 Len=200 |

Figura 9: Consultas realizadas por múltiples clientes conectados a un mismo servidor, vistas en *Wireshark*.

En la figura 9 también se puede apreciar que cada cliente tiene su propio puerto asigando, que es diferente del de los otros clientes, y que todos los clientes se conectan al mismo puerto del servidor. La dinámica por cliente es la misma; primero se genera una conexión vía TCP, y dentro de esta, para que el servidor entregue su respuesta, se genera una conexión vía UDP. Pero, como varios clientes están haciendo consultas al mismo tiempo, el servidor las hace todas de forma paralela.

Cabe destacar también que la diferencia que ocurre al nivel de cliente-servidor en conexión con múltiples clientes, ocurre también cuando el servidor realiza las consultas HTTP en el puerto 80; cuando el servidor realiza una consulta HTTP y luego otra con una diferencia de tiempos muy pequeña entre ellas, también puede que el orden de respuestas recibidas sea diferente. Eso se puede ver en la figura 10 a continuación, en el que el servidor primero hace una petición HTTP desde el puerto 63491, luego hace una segunda petición desde el puerto 63492, y primero se le responde la segunda petición.



| | | | | | | |
|---|---|---|---|---|---|---|
| 25090 | 59.576607 | 192.168.100.20 | 108.167.165.205 | TCP | 66 | 63491 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1 |
| 25127 | 59.717726 | 108.167.165.205 | 192.168.100.20 | TCP | 66 | 80 → 63491 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1412 SACK_PERM=1 WS=128 |
| 25128 | 59.717824 | 192.168.100.20 | 108.167.165.205 | TCP | 54 | 63491 → 80 [ACK] Seq=1 Ack=1 Win=262400 Len=0 |
| 25129 | 59.718047 | 192.168.100.20 | 108.167.165.205 | HTTP | 91 | GET / HTTP/1.1 |
| 25131 | 59.719009 | 192.168.100.20 | 64.233.190.99 | TCP | 66 | 63492 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1 |
| 25134 | 59.722664 | 64.233.190.99 | 192.168.100.20 | TCP | 66 | 80 → 63492 [SYN, ACK] Seq=0 Ack=1 Win=62920 Len=0 MSS=1412 SACK_PERM=1 WS=256 |
| 25135 | 59.722780 | 192.168.100.20 | 64.233.190.99 | TCP | 54 | 63492 → 80 [ACK] Seq=1 Ack=1 Win=262400 Len=0 |
| 25136 | 59.723300 | 192.168.100.20 | 64.233.190.99 | HTTP | 94 | GET / HTTP/1.1 |
| 25137 | 59.726561 | 64.233.190.99 | 192.168.100.20 | TCP | 60 | 80 → 63492 [ACK] Seq=1 Ack=41 Win=62976 Len=0 |
| 25148 | 59.766452 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=1 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25149 | 59.766453 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=1413 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25150 | 59.766455 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=2825 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25151 | 59.766455 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=4237 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25152 | 59.766456 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=5649 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25153 | 59.766456 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=7061 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25154 | 59.766457 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=8473 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25155 | 59.766457 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=9885 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25156 | 59.766458 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=11297 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25157 | 59.766458 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=12709 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25158 | 59.766536 | 192.168.100.20 | 64.233.190.99 | TCP | 54 | 63492 → 80 [ACK] Seq=41 Ack=14121 Win=262400 Len=0 |
| 25160 | 59.767834 | 192.168.100.20 | 64.233.190.99 | TCP | 54 | 63492 → 80 [RST, ACK] Seq=41 Ack=14121 Win=0 Len=0 |
| 25161 | 59.769553 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=14121 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25162 | 59.769554 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=15533 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25163 | 59.769555 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=16945 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25164 | 59.769555 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=18357 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25165 | 59.769556 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=19769 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25166 | 59.770349 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=21181 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25167 | 59.770350 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=22593 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25168 | 59.770352 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=24005 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25169 | 59.770352 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=25417 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25170 | 59.770353 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=26829 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25171 | 59.770354 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=28241 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25172 | 59.770354 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=29653 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25173 | 59.770355 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=31065 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25174 | 59.770355 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=32477 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25175 | 59.770356 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=33889 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25176 | 59.770356 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=35301 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25177 | 59.771002 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=36713 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25178 | 59.771003 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=38125 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25179 | 59.771004 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | 80 → 63492 [ACK] Seq=39537 Ack=41 Win=62976 Len=1412 [TCP segment of a reassembled PDU] |
| 25180 | 59.771004 | 64.233.190.99 | 192.168.100.20 | TCP | 1466 | HTTP/1.1 200 OK [TCP segment of a reassembled PDU] |
| 25204 | 59.858791 | 108.167.165.205 | 192.168.100.20 | TCP | 60 | 80 → 63491 [ACK] Seq=1 Ack=38 Win=29312 Len=0 |
| 25300 | 60.227763 | 108.167.165.205 | 192.168.100.20 | HTTP | 265 | HTTP/1.1 301 Moved Permanently |
| 25301 | 60.227891 | 192.168.100.20 | 108.167.165.205 | TCP | 54 | 63491 → 80 [FIN, ACK] Seq=38 Ack=212 Win=262400 Len=0 |
| 25338 | 60.368899 | 108.167.165.205 | 192.168.100.20 | TCP | 60 | 80 → 63491 [FIN, ACK] Seq=212 Ack=39 Win=29312 Len=0 |
| 25339 | 60.368961 | 192.168.100.20 | 108.167.165.205 | TCP | 54 | 63491 → 80 [ACK] Seq=39 Ack=213 Win=262400 Len=0 |

Figura 10: Consultas HTTP realizadas por servidor desde dos puertos diferentes al puerto 80, vistas en *Wireshark*.