

# Niveles de Aplicación y Transporte

Análisis del tráfico con Wireshark

L. Falcon

Septiembre 2025

## Nivel de Aplicación

### 1 Primera solicitud GET

En la figura 1 se muestra la captura de Wireshark de la primera trama GET:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.122	18.9.22.169	HTTP	802	GET /img/MIT logo.gif HTTP/1.1
2	0.091557	18.9.22.169	192.168.1.122	TCP	1414	80 → 64123 [PSH, ACK] Seq=1 Ack=737 Wi
3	0.091645	192.168.1.122	18.9.22.169	TCP	66	64123 → 80 [ACK] Seq=737 Ack=1349 Win-
4	0.092632	18.9.22.169	192.168.1.122	HTTP	591	HTTP/1.1 200 OK (GIF89a)
5	0.092655	192.168.1.122	18.9.22.169	TCP	66	64123 → 80 [ACK] Seq=737 Ack=1874 Win-
6	9.429071	192.168.1.122	18.9.22.169	HTTP	892	GET /img/MIT_logo.gif HTTP/1.1

Frame 1: 802 bytes on wire (6416 bits), 802 bytes captured (6416 bits)  
Ethernet II, Src: Apple\_ac:6c:26 (10:9a:dd:ac:6c:26), Dst: Cisco-Li\_e3:e9:8d (00:16:b6:e3:e9:8d)  
Internet Protocol Version 4, Src: 192.168.1.122, Dst: 18.9.22.169  
Transmission Control Protocol, Src Port: 64123, Dst Port: 80, Seq: 1, Ack: 1, Len: 736  
Hypertext Transfer Protocol  
  GET /img/MIT\_logo.gif HTTP/1.1\r\n  
    Host: www.mit.edu\r\n    Connection: keep-alive\r\n    Cache-Control: max-age=0\r\nUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_7\_3) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.57 Safari/  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8\r\nAccept-Encoding: gzip,deflate,sdch\r\nAccept-Language: en-US,en;q=0.8\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.3\r\n[truncated]Cookie: \_\_gads=ID=8ad65dc86381f8d7:T=1303597795:S=ALNI\_MYTf6iFAaVdjJxmfT\_V1\_01Q1AyCg; \_\_utma=242276382.142976836\r\n[Full request URI: http://www.mit.edu/img/MIT\_logo.gif]  
[HTTP request 1/3]  
[Response in frame: 4]  
[Next request in frame: 6]

Figure 1: Captura Wireshark de la trama 1 - Primera solicitud GET.

El mensaje HTTP GET siempre lo envía el cliente, ya que es una solicitud.

Recurso solicitado:

/img/MIT\\_\\_logo.gif}

Cliente:

IP: 192.168.1.12 → IP privada (192.168.x.x reservado por IANA).  
Puerto TCP: 64123 → puerto efímero (>1023).

Servidor:

IP: 18.9.22.169 → IP pública (no pertenece a rangos privados).  
Puerto TCP: 80 → puerto well-known reservado para HTTP.

Host destino:

www.mit.edu

### Campos Connection, Cache-Control y User-Agent

- **Connection:** indica cómo manejar la conexión TCP.  
En este caso es `keep-alive`, lo que significa que el cliente solicita mantener la conexión abierta para reutilizarla.
- **Cache-Control:** políticas de caché.  
En este caso es `max-age=0`, lo que indica que el cliente no quiere usar caché y pide el recurso actualizado desde el servidor.
- **User-Agent:** identifica el software cliente que hace la solicitud. Sirve para que el servidor adapte el contenido según el navegador, sistema operativo o dispositivo del cliente. En este caso, el software del cliente es:

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3)
AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.57
Safari/536.11
```

En la figura 2 se observa la respuesta a la primera solicitud:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.122	18.9.22.169	HTTP	802	GET /img/MIT_logo.gif HTTP/1.1
2	0.091557	18.9.22.169	192.168.1.122	TCP	1414	80 → 64123 [PSH, ACK] Seq=1 Ack=7
3	0.091645	192.168.1.122	18.9.22.169	TCP	66	64123 → 80 [ACK] Seq=737 Ack=1349
4	0.092632	18.9.22.169	192.168.1.122	HTTP	591	HTTP/1.1 200 OK (GIF89a)
5	0.092655	192.168.1.122	18.9.22.169	TCP	66	64123 → 80 [ACK] Seq=737 Ack=1874
6	9.429071	192.168.1.122	18.9.22.169	HTTP	892	GET /img/MIT_logo.gif HTTP/1.1

Frame 4: 591 bytes on wire (4728 bits), 591 bytes captured (4728 bits)  
Ethernet II, Src: Cisco-Li\_e3:e9:8d (00:16:b6:e3:e9:8d), Dst: Apple\_ac:6c:26 (10:9a:dd:ac:6c:26)  
Internet Protocol Version 4, Src: 18.9.22.169, Dst: 192.168.1.122  
Transmission Control Protocol, Src Port: 80, Dst Port: 64123, Seq: 1349, Ack: 737, Len: 525  
[2 Reassembled TCP Segments (1873 bytes): #2(1348), #4(525)]  
Hypertext Transfer Protocol  
HTTP/1.1 200 OK\r\nDate: Mon, 16 Jul 2012 05:28:04 GMT\r\nServer: Apache/1.3.41 (Unix) mod\_ssl/2.8.31 OpenSSL/0.9.8j\r\nCache-Control: max-age=81125\r\nExpires: Tue, 17 Jul 2012 04:00:09 GMT\r\nLast-Modified: Mon, 16 Jul 2012 04:00:09 GMT\r\nETag: "10eb008a-5f2-500391c9"\r\nAccept-Ranges: bytes\r\nContent-Length: 1522\r\nX-Connection: close\r\nContent-Type: image/gif\r\n\r\n[HTTP response 1/3]  
[Time since request: 0.092632000 seconds]  
[Request in frame: 1]  
[Next request in frame: 6]  
[Next response in frame: 7]  
[Request URI: http://www.mit.edu/img/MIT\_logo.gif]  
File Data: 1522 bytes  
Compuserve GIF, Version: GIF89a

Figure 2: Captura Wireshark de la trama 4 - Respuesta al primer GET.

En la trama Nro. 4 se observa en el campo Info de Wireshark la respuesta del servidor a la solicitud del cliente:

```
HTTP/1.1 200 OK (GIF89a)
```

El código 200 significa que el servidor entregó correctamente el recurso solicitado; y el identificador GIF89a indica el tipo de contenido devuelto (cabecera de un GIF).

El campo length=591 corresponde a los bytes transferidos en esa trama.

## Campos Server y Last-Modified

- **Server:** identifica el software del servidor. En la trama 4:

```
Apache/1.341 (Unix) mod_ssl/2.8.31 OpenSSL/0.9.8j
```

- **Last-Modified:** fecha de última modificación del recurso. En este caso:

```
Mon, 1 Jul 2012 04:00:09 GMT
```

## 2 Segunda solicitud GET

En la figura 3 se muestra la segunda solicitud (trama 6) y su respuesta (trama 7):

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.122	18.9.22.169	HTTP	802	GET /img/MIT_logo.gif HTTP/1.1
2	0.091557	18.9.22.169	192.168.1.122	TCP	1414	80 → 64123 [PSH, ACK] Seq=1 Ack=73
3	0.091645	192.168.1.122	18.9.22.169	TCP	66	64123 → 80 [ACK] Seq=737 Ack=1349
4	0.092632	18.9.22.169	192.168.1.122	HTTP	591	HTTP/1.1 200 OK (GIF89a)
5	0.092655	192.168.1.122	18.9.22.169	TCP	66	64123 → 80 [ACK] Seq=737 Ack=1874
6	9.429071	192.168.1.122	18.9.22.169	HTTP	892	892 GET /img/MIT_logo.gif HTTP/1.1
7	9.517857	18.9.22.169	192.168.1.122	HTTP	312	HTTP/1.1 304 Not Modified
8	9.517915	192.168.1.122	18.9.22.169	TCP	66	64123 → 80 [ACK] Seq=1563 Ack=2120

```

Frame 6: 892 bytes on wire (7136 bits), 892 bytes captured (7136 bits)
Ethernet II, Src: Apple_ac:6c:26 (10:9a:dd:ac:6c:26), Dst: Cisco-Li_e3:e9:8d (00:16:b6:e3:e9:8d)
Internet Protocol Version 4, Src: 192.168.1.122, Dst: 18.9.22.169
Transmission Control Protocol, Src Port: 64123, Dst Port: 80, Seq: 737, Ack: 1874, Len: 826
Hypertext Transfer Protocol
  GET /img/MIT_logo.gif HTTP/1.1\r\n
  Host: www.mit.edu\r\n
  Connection: keep-alive\r\n
  Cache-Control: max-age=0\r\n
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.57 Safari/536.11\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Encoding: gzip,deflate,sdch\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
  [truncated]Cookie: __gads=ID=8ad65dc86381f8d7:T=1303597795:S=ALNI_MYTf6iFAaVdjJxmfT_V1_01Q1AyCg; __utma=242276382.14297
If-None-Match: "10eb008a-5f2-560391c9"\r\n
If-Modified-Since: Mon, 16 Jul 2012 04:00:09 GMT\r\n
\r\n
[Full request URI: http://www.mit.edu/img/MIT_logo.gif]
[HTTP request 2/3]
[Prev request in frame: 1]
[Response in frame: 7]
[Next request in frame: 9]
```

Figure 3: Captura Wireshark de la trama 6 - Segunda solicitud GET.

La solicitud de tipo GET que ocurre dentro de la captura incluye la cabecera cache:

```
If-None-Match: "10eb008a-5f2-500391c9"
If-Modified-Since: Mon, 16 Jul 2012 04:00:09 GMT
```

Esas cabeceras no aparecen en la primera petición; Éstas indican que el cliente ya tiene copia local y consulta si cambió. Como consecuencia, la solicitud ocupa más bytes que la primera (982 vs 802).

En la trama Nro. 7 se observa en el campo Info de Wireshark la respuesta del servidor:

```
HTTP/1.1 304 Not Modified
```

La respuesta confirma que el recurso no cambió y permite al cliente usar su copia en caché, evitando la retransmisión de datos y optimizando el ancho de banda.

### 3 Cuarta solicitud GET

Figura 4: solicitud de la página principal de [www.washington.edu](http://www.washington.edu).

No.	Time	Source	Destination	Protocol	Length	Info
34	24.364808	192.168.1.122	128.95.155.197	HTTP	750	GET / HTTP/1.1
35	24.365791	128.95.155.197	192.168.1.122	TCP	74	80 → 64166 [SYN, ACK]
36	24.365847	192.168.1.122	128.95.155.197	TCP	66	64166 → 80 [ACK]
37	24.380588	128.95.155.197	192.168.1.122	TCP	66	80 → 64165 [ACK]
38	24.381628	128.95.155.197	192.168.1.122	TCP	1514	80 → 64165 [ACK]
39	24.381983	128.95.155.197	192.168.1.122	TCP	1514	80 → 64165 [ACK]
40	24.382005	192.168.1.122	128.95.155.197	TCP	66	64165 → 80 [ACK]
41	24.382692	128.95.155.197	192.168.1.122	TCP	1514	80 → 64165 [ACK]
42	24.388513	128.95.155.197	192.168.1.122	TCP	1514	80 → 64165 [ACK]
43	24.390555	192.168.1.122	128.95.155.197	TCP	66	64165 → 80 [ACK]

Frame 34: 750 bytes on wire (6000 bits), 750 bytes captured (6000 bits)
 ▶ Ethernet II, Src: Apple\_ac:6c:26 (10:9a:dd:ac:6c:26), Dst: Cisco-Li\_e3:e9:8d (00:16:b6:e3:e9:8d)
 ▶ Internet Protocol Version 4, Src: 192.168.1.122, Dst: 128.95.155.197
 ▶ Transmission Control Protocol, Src Port: 64165, Dst Port: 80, Seq: 1, Ack: 1, Len: 684
 ▶ Hypertext Transfer Protocol
 ▶ GET / HTTP/1.1\r\n
 Host: www.washington.edu\r\n
 Connection: keep-alive\r\n
 Cache-Control: max-age=0\r\n
 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_7\_3) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/23.0.1271.95 Safari/536.11\r\n
 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8\r\n
 Accept-Encoding: gzip,deflate,sdch\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
 ▶ [truncated]Cookie: \_\_unam=712d866-12ec7499be9-3e59a6c0-34; \_\_utma=80390417.1029074526.1300728560.134128\r\n
 \r\n
 [Full request URI: http://www.washington.edu/]
 [HTTP request 1/2]
 [Response in frame: 195]
 [Next request in frame: 197]

Figure 4: Captura Wireshark de la trama 34 - Cuarta solicitud GET.

Recurso solicitado:

```
/ HTTP/1.1 -> la página principal del servidor
```

Nombre del host del servidor:

```
www.washington.edu
```

Eventos principales:

1. Conexión establecida (SYN, SYN-ACK, ACK).
2. Cliente envía un GET (trama 34)
3. Servidor responde (trama 195).
4. Entre medio hay segmentos TCP que llevan la data de la respuesta (se divididen en varias tramas).

Respuesta:

```
HTTP/1.1 200 OK (text/html)
```

La respuesta indica que el servidor está informando al cliente que la petición fue exitosa y que le entregará una página en formato HTML, según lo especificado.

## 4 Captura general

Con la herramienta *Statistics → HTTP → Packet Counter* de Wireshark (figura 5):

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Total HTTP Packets	196				0.0071	100%	0.2600	26.961
Other HTTP Packets	0				0.0000	0.00%	-	-
▼ HTTP Response Packets	98				0.0035	50.00%	0.1300	26.961
????: broken	0				0.0000	0.00%	-	-
5xx: Server Error	0				0.0000	0.00%	-	-
4xx: Client Error	0				0.0000	0.00%	-	-
▼ 3xx: Redirection	1				0.0000	1.02%	0.0100	9.518
304 Not Modified	1				0.0000	100.00%	0.0100	9.518
▼ 2xx: Success	97				0.0035	98.98%	0.1300	26.961
200 OK	97				0.0035	100.00%	0.1300	26.961
1xx: Informational	0				0.0000	0.00%	-	-
▼ HTTP Request Packets	98				0.0035	50.00%	0.1300	26.963
GET	98				0.0035	100.00%	0.1300	26.963

Figure 5: Packet Counter en Wireshark.

Solicitudes totales:

```
196 GET HTTP}
```

Respuestas recibidas por el cliente:

```
97 respuestas 200 OK
1 respuesta 304 Not Modified
```

Con *Statistics → HTTP → Load Distribution* (figura 6) :

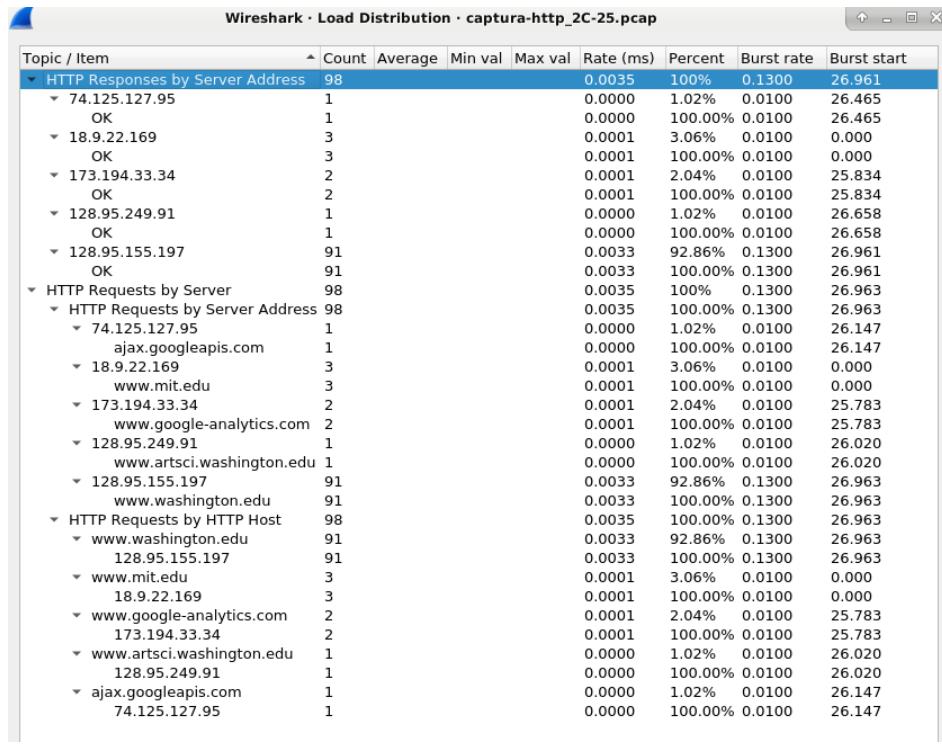


Figure 6: Distribución de hosts en Wireshark.

Cantidad de servidores que recibieron dichas consultas:

98

Los nombres de los hosts:

```
www.washington.com
www.mit.edu
www.google-analytics.com
www.artsci.wahington.edu ajax.googleapis.com
```

# Nivel de Transporte

## 1 Sesiones TCP: Handshake

Una sesión TCP se reconoce por el three-way handshake: SYN → SYN-ACK → ACK.

Para detectarlas se filtró con: `tcp.flags.syn == 1 and tcp.flags.ack == 0` (figura 7).

tcp.flags.syn == 1 and tcp.flags.ack == 0						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.122	64.238.147.113	TCP	78	60643 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TStamp=256679793 TSect=0 SACK PERM=1

Figure 7: Sesiones TCP iniciadas.

Resultado: una sola sesión iniciada (trama 1).

## 2 Inicio de sesión TCP

Figura 8: primer three-way handshake (tramas 1 a 3).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.122	64.238.147.113	TCP	78	60643 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TStamp=256679793 TSect=0 SACK PERM=1
2	0.088810	64.238.147.113	192.168.1.122	TCP	74	80 → 60643 [SYN, ACK] Seq=1 Ack=1 Win=5792 Len=0 MSS=1380 SACK PERM=1 TStamp=4016893437 TSect=256679793 ...
3	0.088880	192.168.1.122	64.238.147.113	TCP	66	60643 → 80 [ACK] Seq=1 Ack=1 Win=524280 Len=0 TStamp=256679881 TSect=4016893437
4	0.088579	192.168.1.122	64.238.147.113	HTTP	257	GET /sigcomm/2011/papers/sigcomm/p2.pdf HTTP/1.1
5	0.177819	64.238.147.113	192.168.1.122	TCP	66	80 → 60643 [ACK] Seq=1 Ack=192 Win=6864 Len=0 TStamp=4016893527 TSect=256679881
6	0.178321	64.238.147.113	192.168.1.122	TCP	311	80 → 60643 [PSH, ACK] Seq=1 Ack=192 Win=6864 Len=245 TStamp=4016893528 TSect=256679881 [TCP segment of a ...
7	0.178388	192.168.1.122	64.238.147.113	TCP	66	60643 → 80 [ACK] Seq=192 Ack=246 Win=524280 Len=0 TStamp=256679970 TSect=4016893528
8	0.180114	64.238.147.113	192.168.1.122	TCP	1434	80 → 60643 [ACK] Seq=246 Ack=192 Win=6864 Len=1368 TStamp=4016893538 TSect=256679881 [TCP segment of a ...
9	0.266705	64.238.147.113	192.168.1.122	TCP	1434	80 → 60643 [ACK] Seq=1614 Ack=192 Win=6864 Len=1368 TStamp=4016893615 TSect=256679970 [TCP segment of a ...
10	0.266787	192.168.1.122	64.238.147.113	TCP	66	60643 → 80 [ACK] Seq=192 Ack=2982 Win=523944 Len=0 TStamp=256688057 TSect=4016893538

Figure 8: Three Way Handshake (tramas 1 a 3).

Cliente:

IP: 192.68.1.122 → IP privada  
Puerto: 60643 → puerto efímero

Servidor:

IP: 64.38.147.113 → IP pública  
Puerto: 80 → puerto HTTP

Existe tráfico a nivel de aplicación esperable dentro de la sesión, porque en la trama 4 aparece:

HTTP GET /sigcomm/2011/papers/sigcomm/p2.pdf HTTP/1.1 → HTTP  
(petición GET/transferencia de un recurso web)

Es decir que el cliente solicita un PDF por HTTP/1.1. Por tanto, dentro de esa sesión se espera intercambio de mensajes HTTP (GET, respuestas 200 con el contenido del PDF, posibles fragmentos TCP con payloads de bytes del fichero).

## Ventana del cliente

La ventana publicada por el cliente ( $W$ ): es el valor que el cliente “anuncia” al servidor que puede recibir. Mientras que el Maximum Segment Size (MSS) es el tamaño máximo de datos (payload) que TCP puede enviar en un solo segmento y solo se anuncia al inicio de la sesión manteniéndose constante durante toda la conexión.

En la trama 1:

$$W = 65535, \quad MSS = 1460 \quad (1)$$

Con factor de escala  $WS=8$  se puede calcular la ventana efectiva  $W_{ef}$ :

$$W_{ef} = W * WS = 65535 * 8 = 524280 \text{ bytes} \quad (2)$$

Trama 3 muestra la ventana efectiva del cliente (y es el valor de ventana efectivo que verá el servidor) confirmado lo anterior dicho:

`Win = 524280 bytes`

La ventana de transmisión es lo que el emisor todavía puede enviar sin violar la capacidad de recepción del receptor.

Ventana de transmisión:

$$W_{tran} = W_{ef(receptor)} - \text{Datosenvuelo} \quad (3)$$

## Ventana del servidor

En la trama 2:

`W = 5792, MSS = 1380`

El servidor anuncia una ventana inicial más pequeña, 5792 bytes. En este caso el no hay window scale (pero podría haber) por lo tanto la ventana efectiva es igual a la ventana y se desomina buffer de recepción.

Sin window scale:

`Buffer de recepción = 5792 bytes`

## RTT

El RTT es el tiempo que tarda un paquete en ir del emisor al receptor y volver con un ACK.

Por lo tanto para calcular el RTT que conlleva el establecimiento del inicio de sesión:

`SYN (trama 1) en Time = 0.000000 s.  
SYN,ACK (trama 2) en Time = 0.088010 s.  
ACK final (trama 3) en Time = 0.088080 s.`

$$RTT = 0.088080s \approx 0.088 s = 88 ms \quad (4)$$

*Nota: Los campos TSval(Timestamp Value) y TSecr (Timestamp Echo Reply) de las tramas de inicio de sesión son opciones TCP de timestamp, sirven para medir el RTT, pero como se uso Wireshark con el tiempo de llegada entre SYN y SYN-ACK alcanzó para estimar el RTT.*

### 3 Finalización de la sesión

Figura 9: tramas 1170 y 1171.

tcp.flags.fin == 1 or tcp.flags.reset == 1						
No.	Time	Source	Destination	Protocol	Length	Info
1171	4.198713	64.238.147.113	192.168.1.122	TCP	66	80 → 60643 [FIN, ACK] Seq=1056771 Ack=193 Win=6864 Len=0 TSval=4016897548 TSecr=256683677
1170	4.111779	192.168.1.122	64.238.147.113	TCP	66	60643 → 80 [FIN, ACK] Seq=192 Ack=1056771 Win=524280 Len=0 TSval=256683677 TSecr=4016897458

Figure 9: Finalización de sesión TCP.

Cierre normal vía FIN, ACK (four-way handshake):

Trama 1170: 60643 → 80 [FIN, ACK]  
Trama 1171: 80 → 60643 [FIN, ACK]

Campos relevantes y valores que aparecen en ambas tramas:

#### Flags

- FIN: indica que el emisor no enviará más datos.
- ACK: confirma la recepción de lo anterior.

#### Seq y Ack

- En la trama 1170: Seq=192, Ack=1056771.
- En la trama 1171: Seq=1056771, Ack=193.  
(Cada lado confirma la recepción hasta el último byte válido).

#### Ventana (Win)

- Cliente anuncia Win=524280.
- Servidor anuncia Win=6864.  
(Indican la ventana de recepción en ese momento).

*Nota: En ambas tramas de cierre, Len = 0 porque no se transmiten datos solo los flags.*

Otras opciones de cierre que pueden darse son:

Cierre abrupto: Se da con un RST (Reset),  
que corta inmediatamente la sesión sin terminar de vaciar buffers.

Timeout: Si ninguna de las partes responde,  
la sesión expira y el sistema operativo la da por cerrada después de un tiempo.

### 4 Evolución de los números de secuencia

Con *Statistics → TCP Stream Graphs → Time Sequence (Stevens)*:

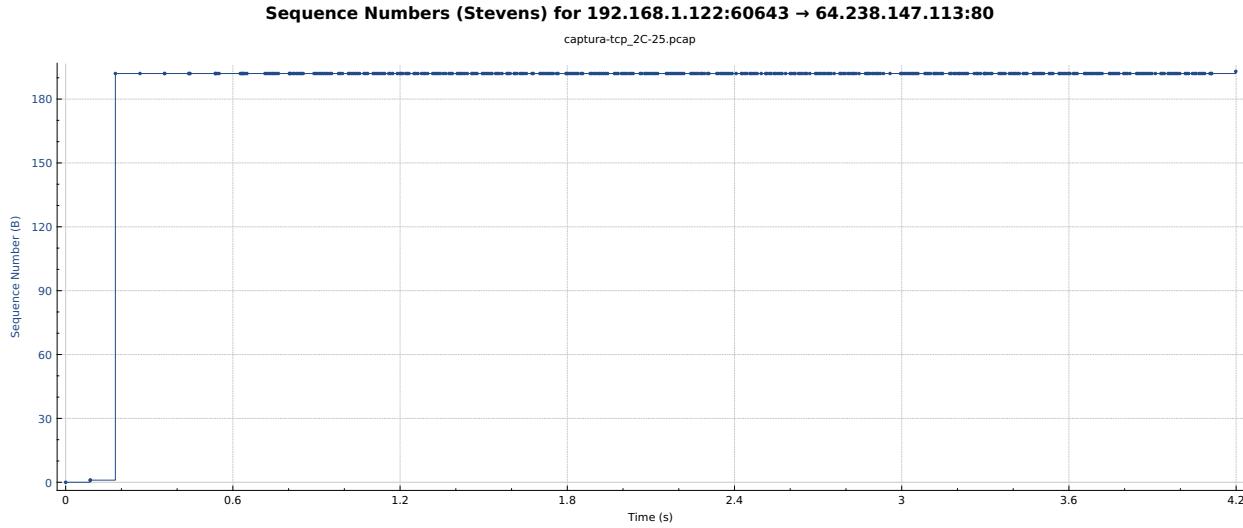


Figure 10: SN vs Tiempo: cliente → servidor.

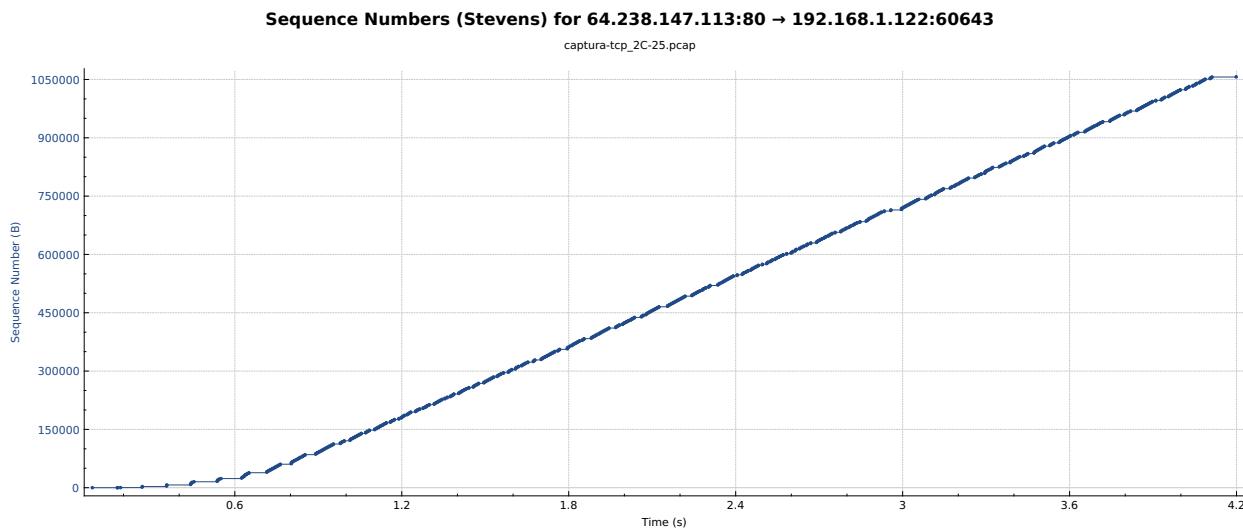


Figure 11: SN vs Tiempo: servidor → cliente.

En la figura 10: **Cliente - crecimiento suave (solicitudes y ACKs):**  
Se observa un crecimiento suave y limitado de los números de secuencia, correspondiente principalmente a solicitudes y confirmaciones. Al inicio aparecen dos valores bajos que reflejan los primeros envíos del cliente (handshake y petición HTTP). Los puntos posteriores muestran incrementos pequeños asociados a confirmaciones (ACKs) y algún dato adicional, pero no se aprecia una subida abrupta, ya que el cliente no transmite grandes volúmenes de información, sino que se limita a iniciar la sesión y enviar la solicitud inicial.

En la figura 11: **Servidor: crecimiento abrupto (envío de datos, páginas, archivos):**  
Aunque también se distinguen valores bajos del handshake como en el caso del cliente, en este caso el crecimiento de los números de secuencia es fuerte y marcado. Esto ocurre porque el servidor es el encargado de transmitir la carga principal de la comunicación (contenido HTTP como páginas e imágenes), por lo que cada bloque de datos enviado provoca un incremento significativo en el número de secuencia, generando una curva con pendiente pronunciada.