# 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:

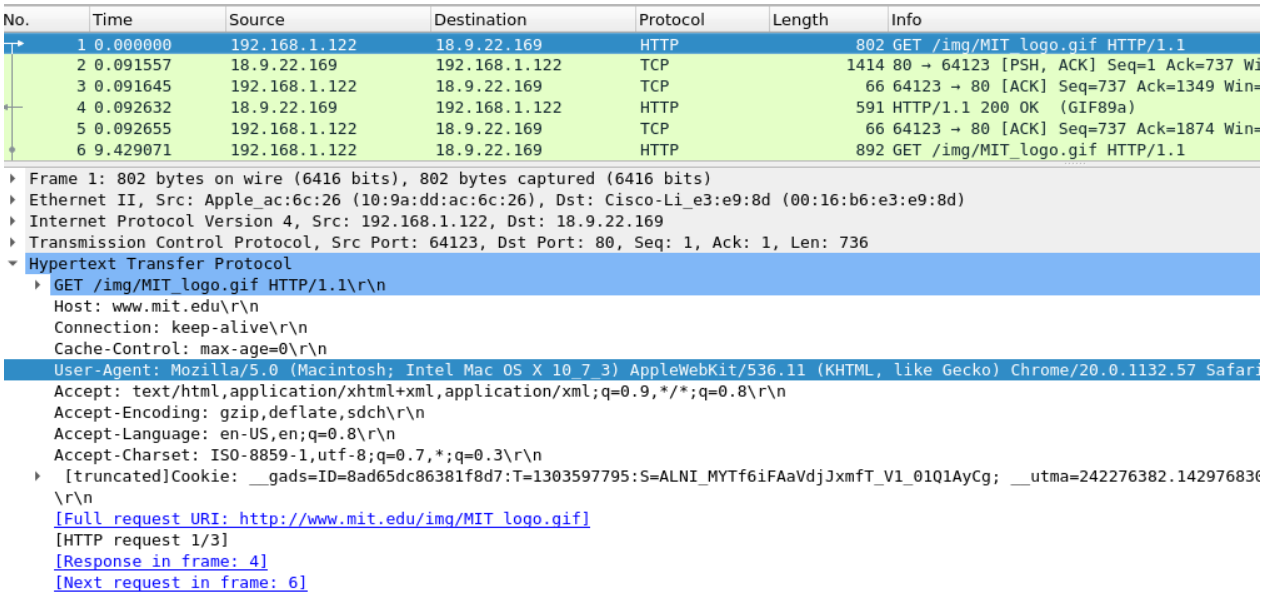



Figure 1: Captura Wireshark de la trama 1 - Primera solicitud GET.

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:

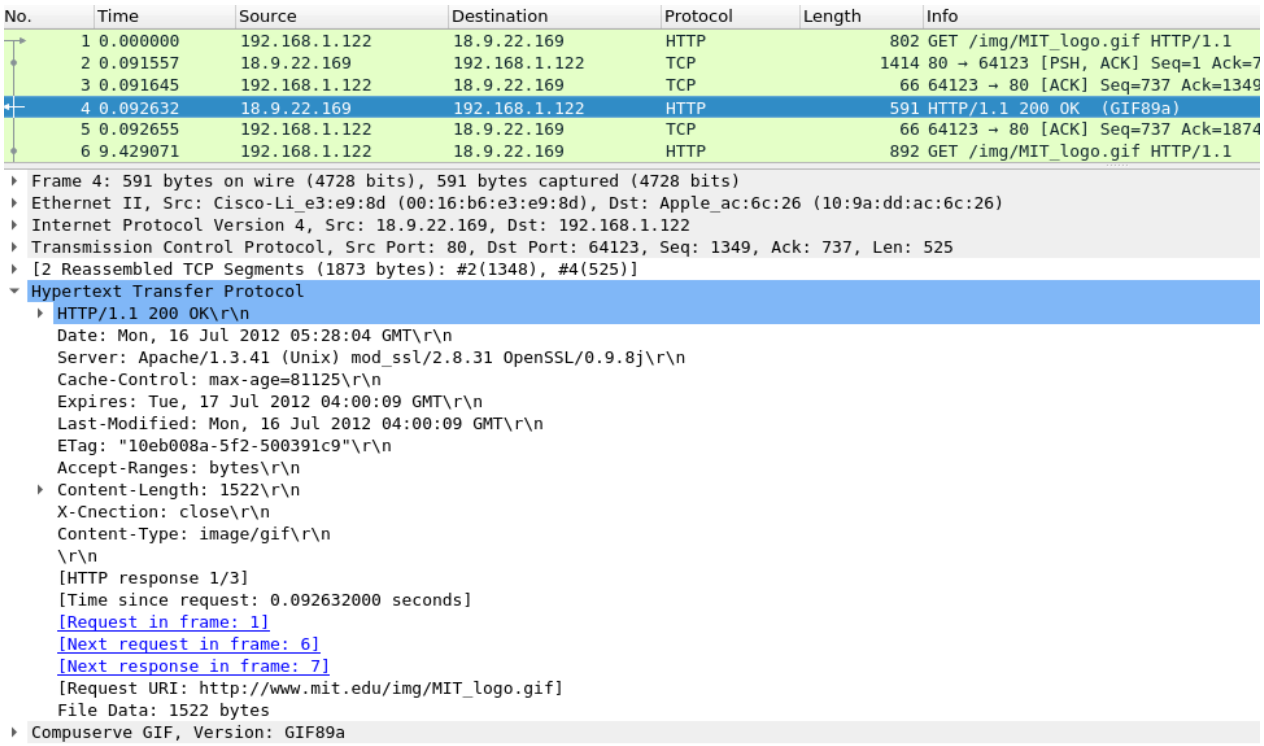



Figure 2: Captura Wireshark de la trama 4 - Respuesta al primer GET.

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):

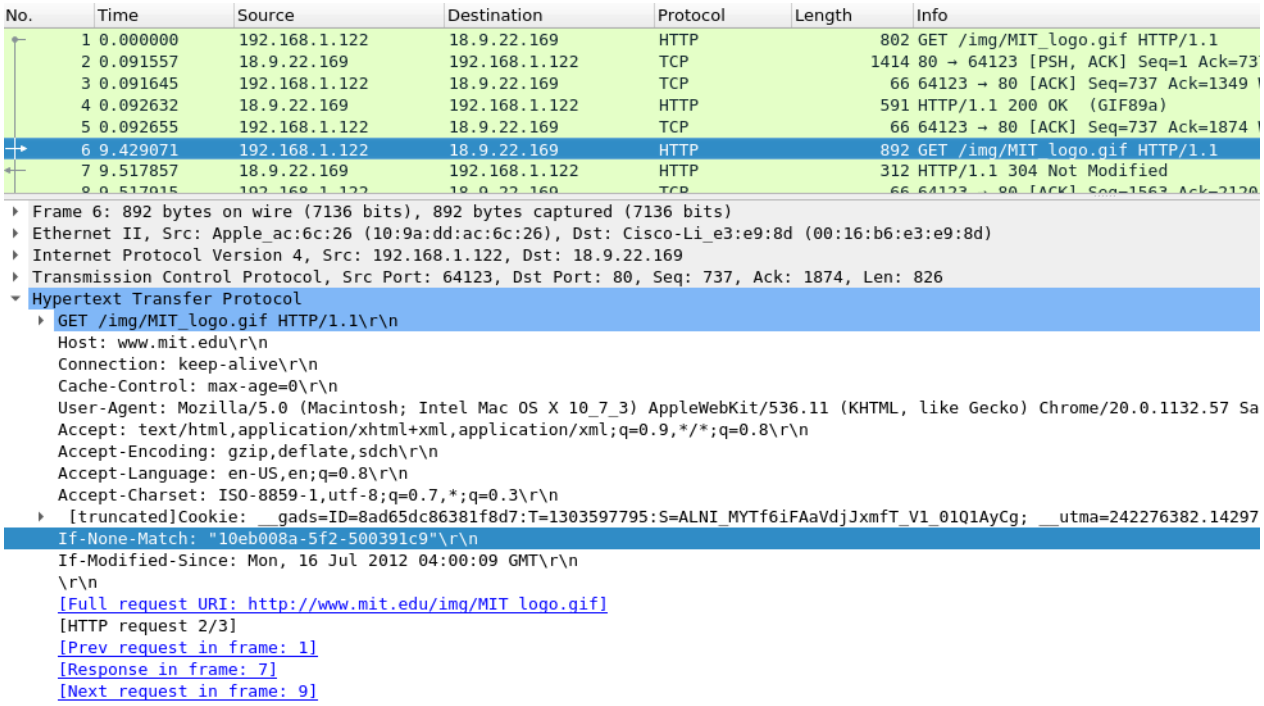



Figure 3: Captura Wireshark de la trama 6 - Segunda solicitud GET.

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 e 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`.

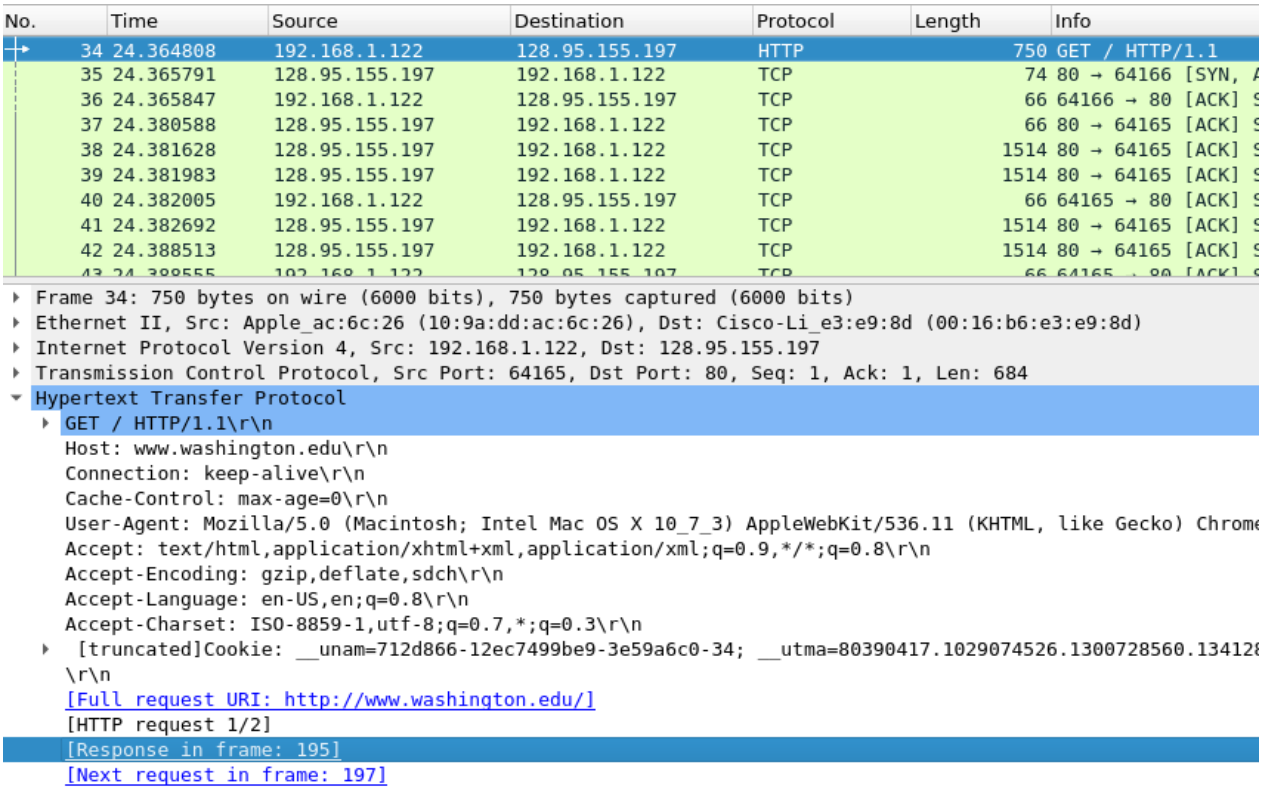



Figure 4: Captura Wireshark de la trama 34 - Cuarta solicitud GET.

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).

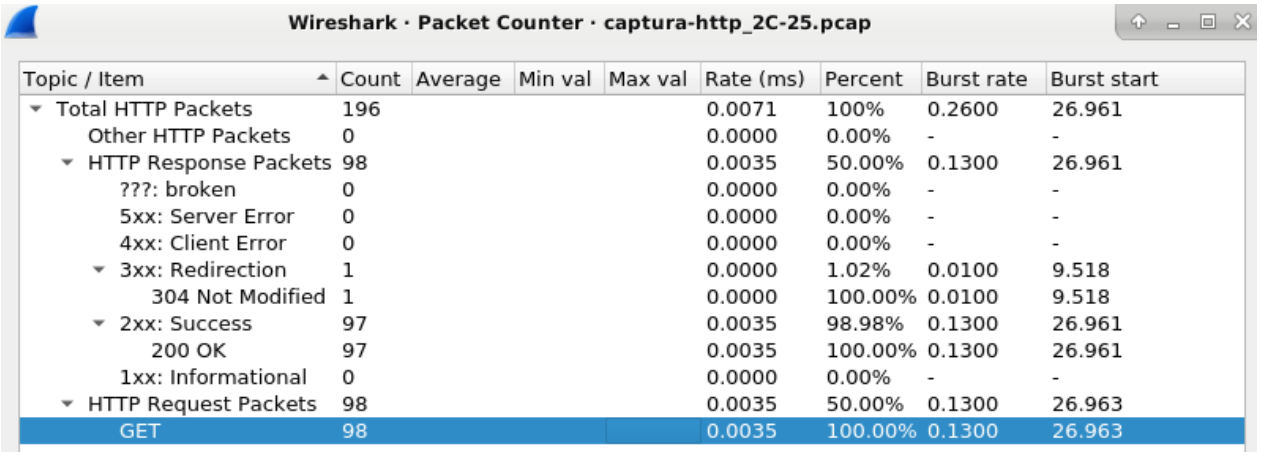
Respuesta:

```
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.

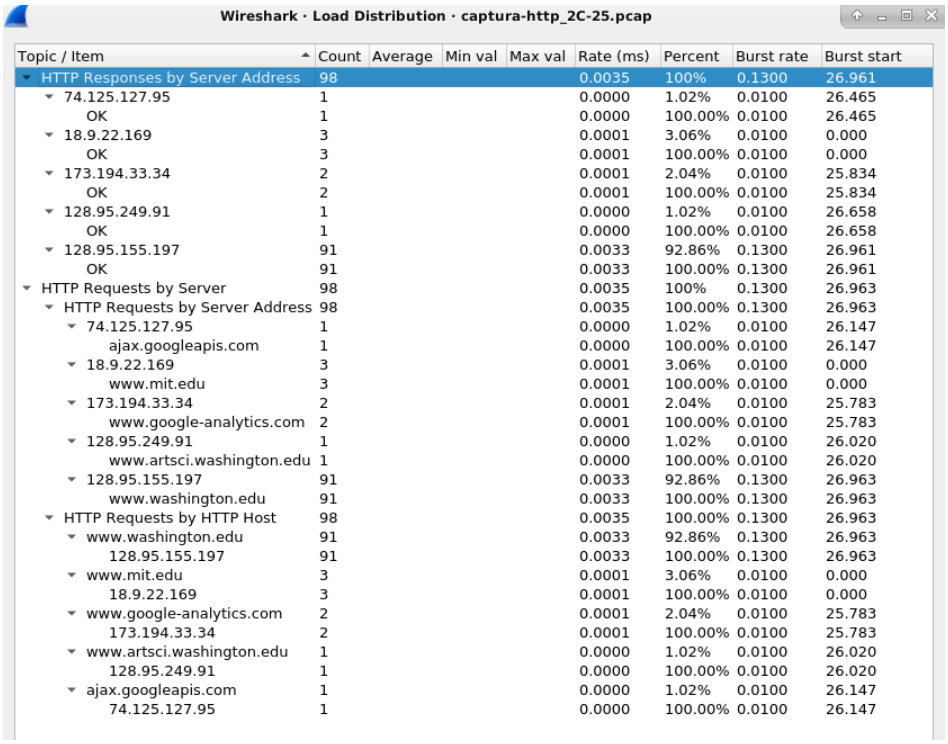
Solicitudes totales:

```
196 GET HTTP}
```

Respuestas recibidas por el cliente:

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

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




| Topic / Item | Count | Average | Min val | Max val | Rate (ms) | Percent | Burst rate | Burst start |
|---|---|---|---|---|---|---|---|---|
| HTTP Responses by Server Address | 98 | | | | 0.0035 | 100% | 0.1300 | 26.961 |
| 74.125.127.95 | 1 | | | | 0.0000 | 1.02% | 0.0100 | 26.465 |
| OK | 1 | | | | 0.0000 | 100.00% | 0.0100 | 26.465 |
| 18.9.22.169 | 3 | | | | 0.0001 | 3.06% | 0.0100 | 0.000 |
| OK | 3 | | | | 0.0001 | 100.00% | 0.0100 | 0.000 |
| 173.194.33.34 | 2 | | | | 0.0001 | 2.04% | 0.0100 | 25.834 |
| OK | 2 | | | | 0.0001 | 100.00% | 0.0100 | 25.834 |
| 128.95.249.91 | 1 | | | | 0.0000 | 1.02% | 0.0100 | 26.658 |
| OK | 1 | | | | 0.0000 | 100.00% | 0.0100 | 26.658 |
| 128.95.155.197 | 91 | | | | 0.0033 | 92.86% | 0.1300 | 26.961 |
| OK | 91 | | | | 0.0033 | 100.00% | 0.1300 | 26.961 |
| HTTP Requests by Server | 98 | | | | 0.0035 | 100% | 0.1300 | 26.963 |
| HTTP Requests by Server Address | 98 | | | | 0.0035 | 100.00% | 0.1300 | 26.963 |
| 74.125.127.95 | 1 | | | | 0.0000 | 1.02% | 0.0100 | 26.147 |
| ajax.googleapis.com | 1 | | | | 0.0000 | 100.00% | 0.0100 | 26.147 |
| 18.9.22.169 | 3 | | | | 0.0001 | 3.06% | 0.0100 | 0.000 |
| www.mit.edu | 3 | | | | 0.0001 | 100.00% | 0.0100 | 0.000 |
| 173.194.33.34 | 2 | | | | 0.0001 | 2.04% | 0.0100 | 25.783 |
| www.google-analytics.com | 2 | | | | 0.0001 | 100.00% | 0.0100 | 25.783 |
| 128.95.249.91 | 1 | | | | 0.0000 | 1.02% | 0.0100 | 26.020 |
| www.artsci.washington.edu | 1 | | | | 0.0000 | 100.00% | 0.0100 | 26.020 |
| 128.95.155.197 | 91 | | | | 0.0033 | 92.86% | 0.1300 | 26.963 |
| www.washington.edu | 91 | | | | 0.0033 | 100.00% | 0.1300 | 26.963 |
| HTTP Requests by HTTP Host | 98 | | | | 0.0035 | 100.00% | 0.1300 | 26.963 |
| www.washington.edu | 91 | | | | 0.0033 | 92.86% | 0.1300 | 26.963 |
| 128.95.155.197 | 91 | | | | 0.0033 | 100.00% | 0.1300 | 26.963 |
| www.mit.edu | 3 | | | | 0.0001 | 3.06% | 0.0100 | 0.000 |
| 18.9.22.169 | 3 | | | | 0.0001 | 100.00% | 0.0100 | 0.000 |
| www.google-analytics.com | 2 | | | | 0.0001 | 2.04% | 0.0100 | 25.783 |
| 173.194.33.34 | 2 | | | | 0.0001 | 100.00% | 0.0100 | 25.783 |
| www.artsci.washington.edu | 1 | | | | 0.0000 | 1.02% | 0.0100 | 26.020 |
| 128.95.249.91 | 1 | | | | 0.0000 | 100.00% | 0.0100 | 26.020 |
| ajax.googleapis.com | 1 | | | | 0.0000 | 1.02% | 0.0100 | 26.147 |
| 74.125.127.95 | 1 | | | | 0.0000 | 100.00% | 0.0100 | 26.147 |

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).

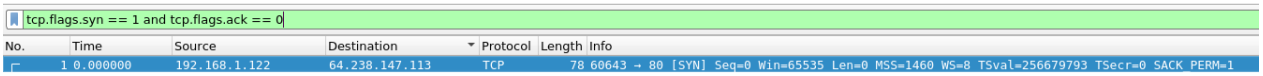



Figure 7: Sesiones TCP iniciadas.

Resultado: una sola sesión iniciada (trama 1).

## 2 Inicio de sesión TCP

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

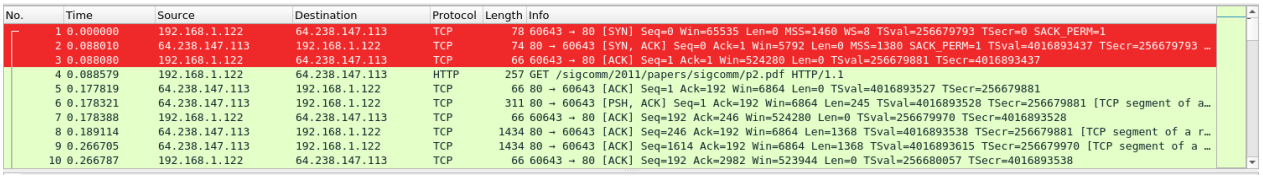



Figure 8: Three Way Handshake (tramas 1 a 3).

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 manteniendose constante durante toda la conexión.

En la trama 1:

$$W = 65535, \quad MSS = 1460 \tag{1}$$

Con factor de escala WS=8 se puede calcular la ventana efectiva Wef:

$$W_{ef} = W * WS = 65535 * 8 = 524280 \; bytes \tag{2}$$

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

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

## 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 podria 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.
```

$$RTT = 0.088080s \approx 0.088 \; s = 88 \; ms \tag{4}$$

*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.

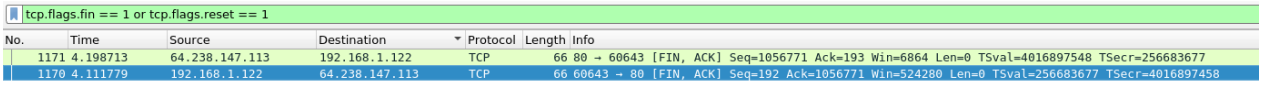



Figure 9: Finalización de sesión TCP.

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)*:

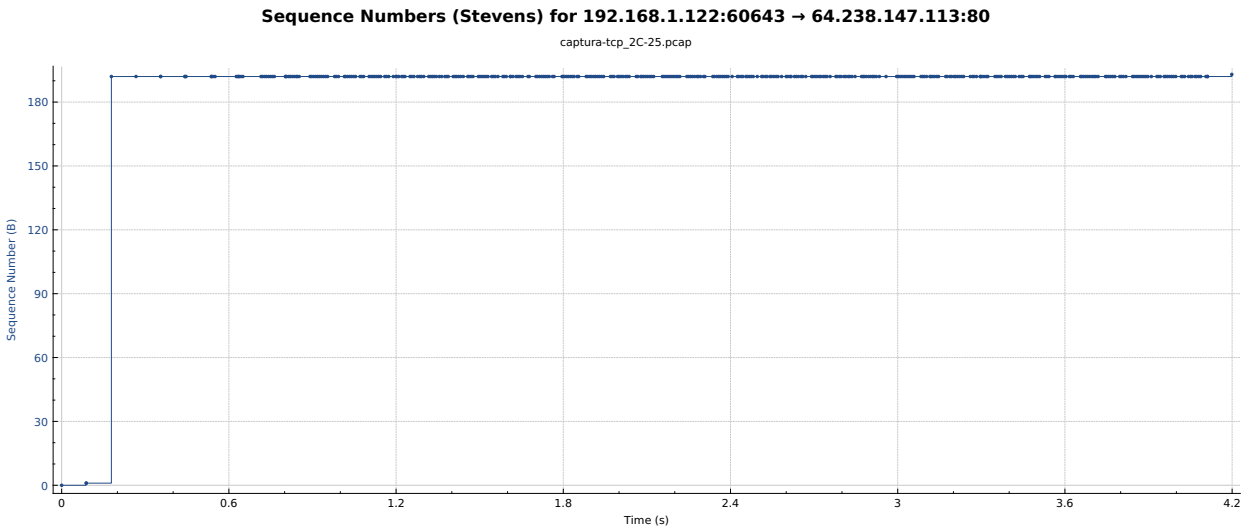



Figure 10: SN vs Tiempo: cliente → servidor.

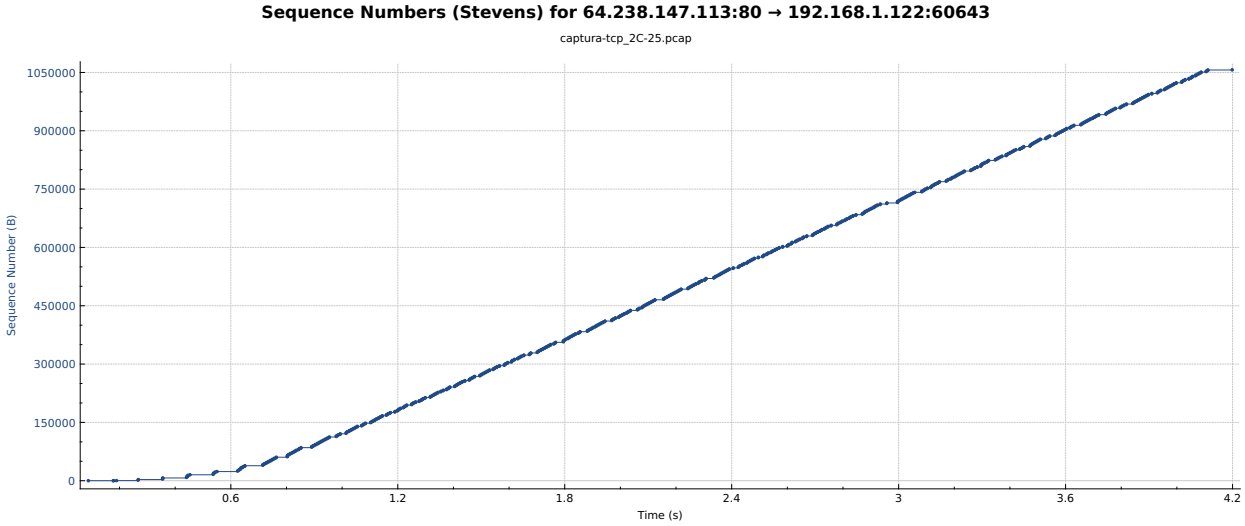



Figure 11: SN vs Tiempo: servidor → cliente.

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.