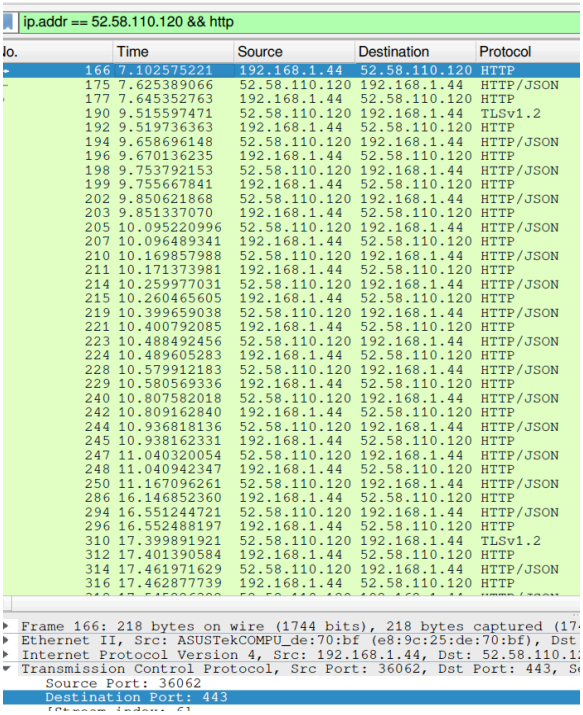
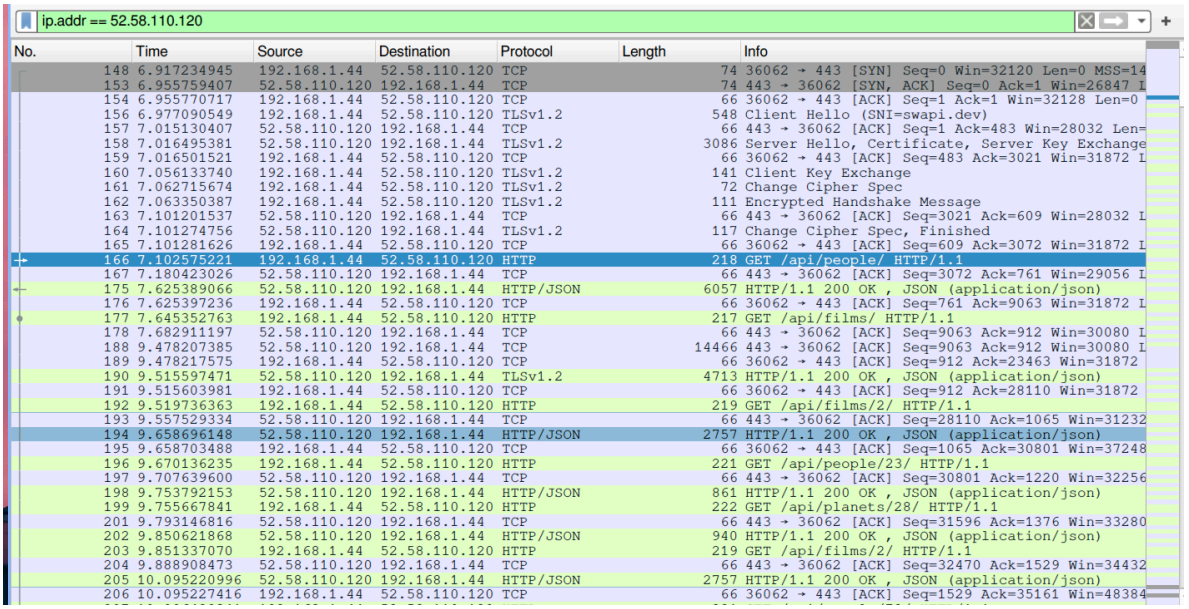
PRÁCTICA 6

Ejercicio 1. ¿Cuál es el puerto utilizado por el servidor? ¿Es el normal de HTTP (80)? ¿Por qué?

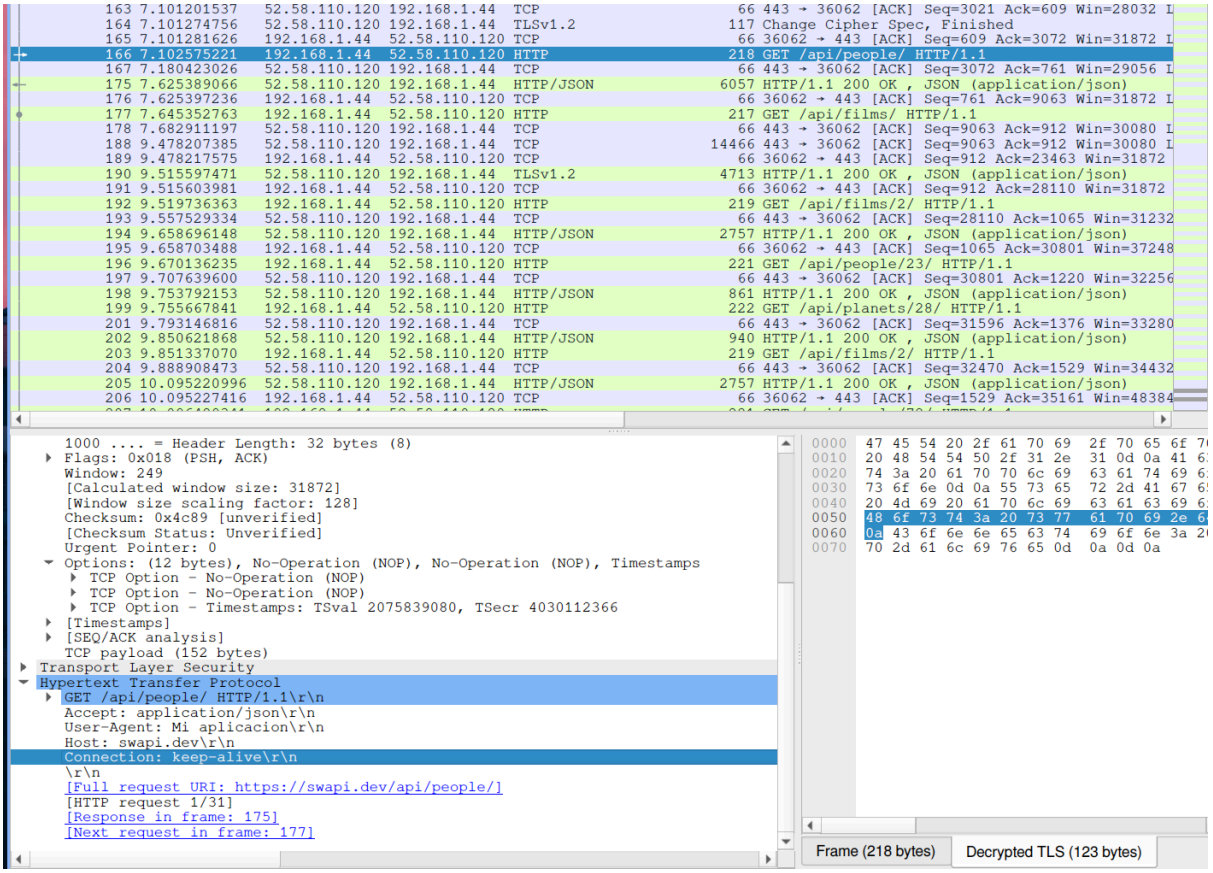


Es el puerto 443. No es el puerto 80 ya que en este caso el servidor utiliza **HTTPS** en vez de HTTP.

Ejercicio 2. Observe el número de conexiones realizadas. ¿Cuántas hace? ¿Usa una conexión permanente (en la misma conexión hace varias peticiones) o no permanente (sólo realiza una por conexión)? En caso de ser permanente, ¿qué cabecera de la petición indica que queremos que sea permanente?



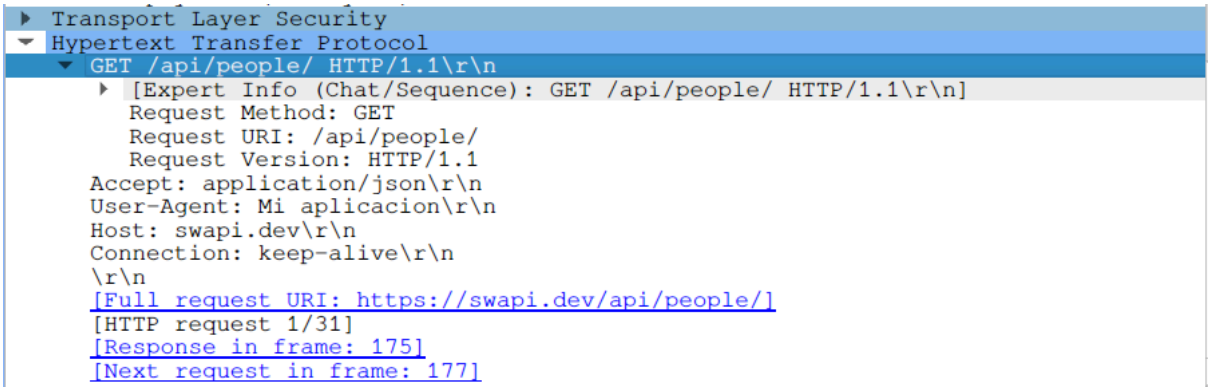
Podemos observar un Client Hello (trama 156) y un Server Hello (trama 158) únicamente por lo tanto hay una única conexión entre cliente y servidor. Como utiliza HTTP/1.1 por defecto se activa la opción keep-alive. De todas formas voy a acceder a las cabeceras de la primera trama HTTP (trama 166) para comprobarlo.



Podemos observar como la cabecera Connection contiene el valor “keep-alive”.

Ejercicio 3. Describa el significado de las cabeceras de una petición y una respuesta (sin incluir las X-*).

Para la explicación de las cabeceras de una petición voy a hacer una captura de pantalla de la primera solicitud HTTP que se hace desde mi ordenador hacia el servidor (trama 166).



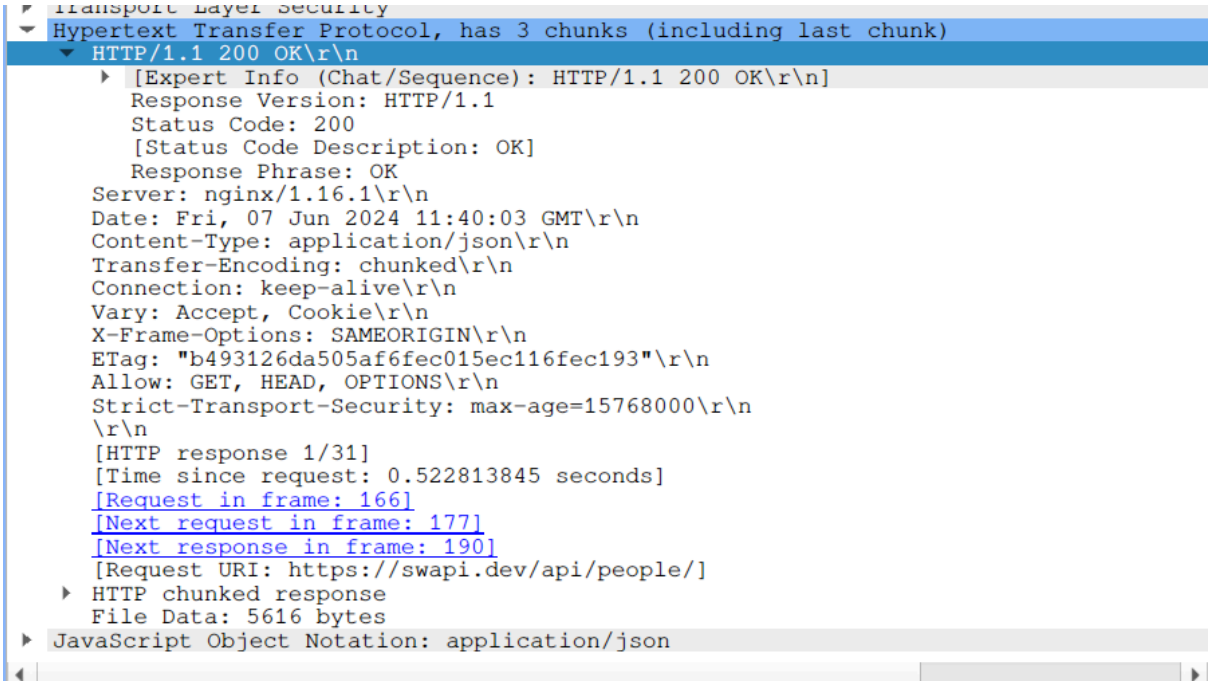
Request Method: Indica el tipo de petición que se realiza sobre el servidor. En este caso GET ya que tratamos de obtener un recurso.

Request URI: Indica a partir de la url base del servidor a que ruta quiero hacer la petición GET. En este caso a “/api/people/”.

Request Version: Indica la versión HTTP usada para la solicitud. En este caso HTTP/1.1

Accept: Indica el tipo de respuesta que espera por parte del servidor. En este caso una respuesta en formato json.
User-Agent: Indica el nombre con el que se identifica mi aplicación. En este caso mi aplicación java con la que realizo peticiones al servidor.
Host: Indica el nombre de la url base a la que intento hacer una petición.
Connection: Indica si la conexión se mantiene abierta tras recibir la respuesta. En este caso como es HTTP/1.1 se mantiene abierta.

Para la explicación de las cabeceras de una respuesta voy a hacer una captura de pantalla de la respuesta a la solicitud GET explicada arriba que se hace desde el servidor (trama 175).



Response Version: Indica la versión HTTP que utiliza en la respuesta
Status Code: Indica el codigo de estado. En este caso 200. Que indica que no ha habido ningún problema.
Response Phrase: Relacionado con el estatus code pero en forma de String.
Server: Indica el nombre del servidor. En este caso se ve que la página utiliza Nginx como proxy.
Date: Indica la fecha de respuesta
Content-Type: Informa sobre el tipo de datos que lleva. En este caso JSON
Transfer-Encoding: Indica el tipo de codificación que utiliza ya que los datos se mandan en partes, en este caso utiliza “chunked” que es el más común.
Connection: Informa si debe seguir la conexión abierta.
Vary: Por lo que he entendido, vary indica a la caché que debe almacenar distintas versiones del tipo pasado. En este caso es Cookie. Almacena en caché cada una de las respuestas que se diferencien por Cookies.
ETag: Identificador único del recurso enviado
Allow: El servidor indica que tipo de peticiones son válidas para este recurso.
Strict-Transport-Security: Indica que solo puede comunicarse con este servidor de forma segura usando HTTPS
File Data: Indica la cantidad de datos en bytes que se han enviado.l

PRÁCTICA 7

Ejercicio 1. Explique cada una de las instrucciones enviadas por el cliente, indicando para qué se usa. ¿Cómo determina el servidor cuándo termina el cuerpo del correo?
La primera instrucción establece la conexión con el servidor. Tras esto indicamos con MAIL FROM el emisor del mensaje y con RCPT TO el receptor (pueden ser varios mandando varias veces este mensaje).
Tras esto informamos al servidor de que vamos a empezar a mandar los datos del mensaje con la cadena DATA.
Primero mandamos las cabeceras From: que contiene el emisor del correo, To: que contiene el destinatario (pueden ser varios como es el caso), Subject: que indica el asunto del correo (el título).
Una vez enviadas las cabeceras enviamos una linea en blanco para separar del cuerpo del mensaje. Enviamos tantas lineas como queramos sin superar el límite establecido. Para terminar mandamos un único punto rodeado de <CR><LF> como indica el servidor cuando le enviamos DATA. Así determina que hemos terminado el mensaje.

Ejercicio 2. ¿Por qué el servidor tras el comando DATA envía un código de tipo 3xx?
Indica que es correcto lo que estoy haciendo pero debo hacer algo más. En este caso tengo que enviar un . final para cerrar el mensaje cuando termine de enviar todo el cuerpo.

Ejercicio 3. ¿Por qué hay tres envíos de comando RCPT TO?
Porque se están indicando los destinatarios del mensaje. Hay que enviar un mesaje RCPT TO: por cada correo al que queramos enviar el mensaje.

Ejercicio 4. Si observa la imágen previa los destinatarios son enviados en los campos Para, CC (Carbon Copy) y BCC (Blind Carbon Copy). ¿Qué diferencia a nivel de comando SMTP y contenido del correo en sí mismo tiene que un destinatario sea indicado en un campo u otro?
Los tres se tienen que definir a nivel SMTP en RCPT TO porque se envía el mensaje a los tres. La diferencia es que en la cabecera se indica quien es el destinatario principal, y el carbon copy. En el mensaje aparece información sobre todo aquel que reciba el correo. Estos dos aparecen. Aquellos correos que no se registren en To ni en CC recibirán una copia sin notificar a los otros receptores.

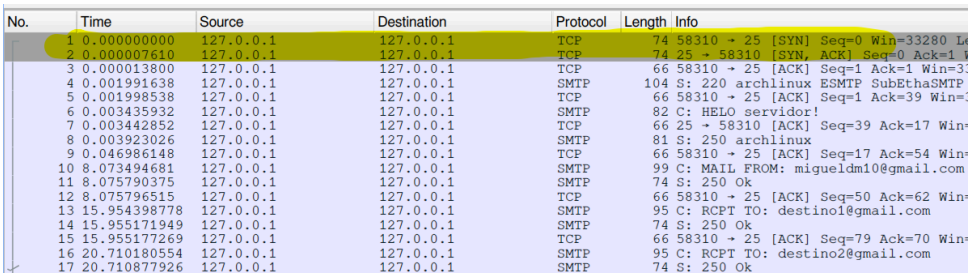
Ejercicio 5. Use la opción Follow TCP Stream de Wireshark para observar el diálogo completo que han mantenido el cliente de correo y el servidor. Adjunte una captura de pantalla donde se observe dicho diálogo.

```
220 archlinux ESMTP SubEthaSMTP null
HELO servidor!
250 archlinux
MAIL FROM: migueldm10@gmail.com
250 Ok
RCPT TO: destino1@gmail.com
250 Ok
RCPT TO: destino2@gmail.com
250 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
From: migueldm10@gmail.com
To: destino1@gmail.com
To: destino2@gmail.com
Subject: Asunto del correo

Linea 1 del mensaje
Linea 2 del mensaje

.
250 Ok
QUIT
221 Bye
```

Ejercicio 6. ¿En cuales mensajes se usa piggybacking? ¿Por qué? En los que no usen esa estrategia, los mensajes de datos ¿confirman algo? ¿El qué?
Toda la configuración de la conexión tienen que esperar una confirmación para continuar enviando. Una vez se comienza a enviar el mensaje “DATA”, solo se recibe ack confirmando que ha llegado al servidor pero no recibe información extra indicando si el servidor acepta o no el formato. Una vez se envía el punto final, entonces el servidor responde con piggybacking informado si el mensaje es correcto o no. Esto se podría hacer para evitar saturar la red con respuestas innecesarias ya que el mensaje no ha terminado todavía. Mejor esperar a tenerlo completo y enviar el estado al final.

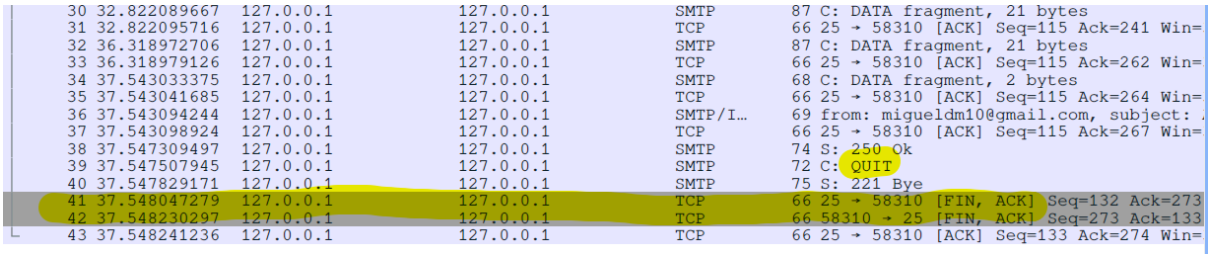
Ejercicio 7. Si observa el interfaz gráfico de FakeSMTP, verá que este correo lo ha recibido varias veces. ¿Por qué?
Se recibe un correo por cada destinatario indicado. Como he enviado a dos destinatarios, aparecen dos correos.

Ejercicio 8. Sabría indicar (quizás mediante un uso “inteligente” del cliente desarrollado), si el servidor FakeSMTP es iterativo o concurrente. Justifique la respuesta y añada capturas de pantalla para apoyar su contestación.
Observando la traza en wireshark podemos ver como solo se establece la conexión una vez, se envían todos los datos a todos los correos y se cierra la conexión.



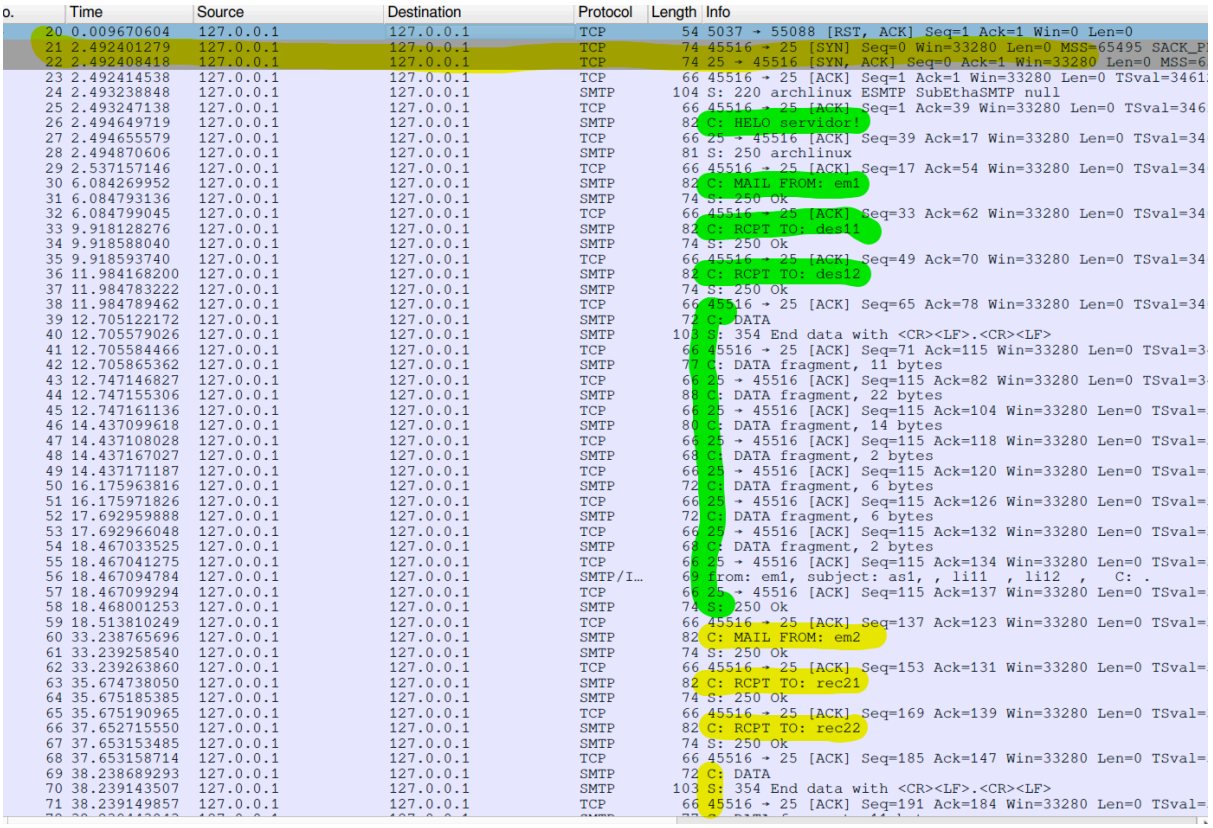
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	58310 → 25 [SYN] Seq=0 Win=33280 L
2	0.000007610	127.0.0.1	127.0.0.1	TCP	74	25 → 58310 [SYN, ACK] Seq=0 Ack=1
3	0.000013800	127.0.0.1	127.0.0.1	TCP	66	58310 → 25 [ACK] Seq=1 Ack=1 Win=3
4	0.001991638	127.0.0.1	127.0.0.1	SMTP	104	S: 220 archlinux ESMTP SubEthaSMTP
5	0.001998538	127.0.0.1	127.0.0.1	TCP	66	58310 → 25 [ACK] Seq=1 Ack=39 Win=
6	0.003435932	127.0.0.1	127.0.0.1	SMTP	82	C: HELO servidor!
7	0.003442852	127.0.0.1	127.0.0.1	TCP	66	25 → 58310 [ACK] Seq=39 Ack=17 Win
8	0.003923026	127.0.0.1	127.0.0.1	SMTP	81	S: 250 archlinux
9	0.046986148	127.0.0.1	127.0.0.1	TCP	66	58310 → 25 [ACK] Seq=17 Ack=54 Win
10	8.073494681	127.0.0.1	127.0.0.1	SMTP	99	C: MAIL FROM: migueldm10@gmail.com
11	8.075790375	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok
12	8.075796515	127.0.0.1	127.0.0.1	TCP	66	58310 → 25 [ACK] Seq=50 Ack=62 Win
13	15.954398778	127.0.0.1	127.0.0.1	SMTP	95	C: RCPT TO: destino1@gmail.com
14	15.955171949	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok
15	15.955177269	127.0.0.1	127.0.0.1	TCP	66	58310 → 25 [ACK] Seq=79 Ack=70 Win
16	20.710180554	127.0.0.1	127.0.0.1	SMTP	95	C: RCPT TO: destino2@gmail.com
17	20.710877926	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok

Una vez se indica al servidor el final del mensaje se manda un QUIT informando de que queremos terminar la conexión.



30	32.822089667	127.0.0.1	127.0.0.1	SMTP	87	C: DATA fragment, 21 bytes
31	32.822095716	127.0.0.1	127.0.0.1	TCP	66	25 → 58310 [ACK] Seq=115 Ack=241 Win=
32	36.318972706	127.0.0.1	127.0.0.1	SMTP	87	C: DATA fragment, 21 bytes
33	36.318979126	127.0.0.1	127.0.0.1	TCP	66	25 → 58310 [ACK] Seq=115 Ack=262 Win=
34	37.543033375	127.0.0.1	127.0.0.1	SMTP	68	C: DATA fragment, 2 bytes
35	37.543041685	127.0.0.1	127.0.0.1	TCP	66	25 → 58310 [ACK] Seq=115 Ack=264 Win=
36	37.543094244	127.0.0.1	127.0.0.1	SMTP/I…	69	from: migueldm10@gmail.com, subject:
37	37.543098924	127.0.0.1	127.0.0.1	TCP	66	25 → 58310 [ACK] Seq=115 Ack=267 Win=
38	37.547309497	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok
39	37.547507945	127.0.0.1	127.0.0.1	SMTP	72	C: QUIT
40	37.547829171	127.0.0.1	127.0.0.1	SMTP	75	S: 221 Bye
41	37.548047279	127.0.0.1	127.0.0.1	TCP	66	25 → 58310 [FIN, ACK] Seq=132 Ack=273
42	37.548230297	127.0.0.1	127.0.0.1	TCP	66	58310 → 25 [FIN, ACK] Seq=273 Ack=133
43	37.548241236	127.0.0.1	127.0.0.1	TCP	66	25 → 58310 [ACK] Seq=133 Ack=274 Win=

Por lo tanto podemos determinar que se establece una conexión concurrente.
He modificado el código y he metido toda la configuración dentro de un while para comprobar si puedo enviar más de un correo sin cerrar la conexión y volverla a abrir. Efectivamente se pueden enviar de forma concurrente información siempre que se indique que un mensaje ha terminado con un “.” y se vuelva a enviar la configuración del nuevo mensaje



o.	Time	Source	Destination	Protocol	Length	Info
20	0.009670604	127.0.0.1	127.0.0.1	TCP	54	5037 → 55088 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
21	2.492401279	127.0.0.1	127.0.0.1	TCP	74	45516 → 25 [SYN] Seq=0 Win=33280 Len=0 MSS=65495 SACK_P
22	2.492408418	127.0.0.1	127.0.0.1	TCP	74	25 → 45516 [SYN, ACK] Seq=0 Ack=1 Win=33280 Len=0 MSS=6
23	2.492414538	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=1 Ack=1 Win=33280 Len=0 TSval=3461
24	2.493238848	127.0.0.1	127.0.0.1	SMTP	104	S: 220 archlinux ESMTP SubEthaSMTP null
25	2.493247138	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=1 Ack=39 Win=33280 Len=0 TSval=346
26	2.494649719	127.0.0.1	127.0.0.1	SMTP	82	C: HELO servidor!
27	2.494655579	127.0.0.1	127.0.0.1	TCP	66	25 → 45516 [ACK] Seq=39 Ack=17 Win=33280 Len=0 TSval=34
28	2.494870606	127.0.0.1	127.0.0.1	SMTP	81	S: 250 archlinux
29	2.537157146	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=17 Ack=54 Win=33280 Len=0 TSval=34
30	6.084269952	127.0.0.1	127.0.0.1	SMTP	82	C: MAIL FROM: em1
31	6.084793136	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok
32	6.084799045	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=33 Ack=62 Win=33280 Len=0 TSval=34
33	9.918128276	127.0.0.1	127.0.0.1	SMTP	81	C: RCPT TO: des11
34	9.918588040	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok
35	9.918593740	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=49 Ack=70 Win=33280 Len=0 TSval=34
36	11.984168200	127.0.0.1	127.0.0.1	SMTP	82	C: RCPT TO: des12
37	11.984783222	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok
38	11.984789462	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=65 Ack=78 Win=33280 Len=0 TSval=34
39	12.705122172	127.0.0.1	127.0.0.1	SMTP	72	C: DATA
40	12.705579026	127.0.0.1	127.0.0.1	SMTP	103	S: 354 End data with <CR><LF>.<CR><LF>
41	12.705584466	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=71 Ack=115 Win=33280 Len=0 TSval=3
42	12.705865362	127.0.0.1	127.0.0.1	SMTP	77	C: DATA fragment, 11 bytes
43	12.747146827	127.0.0.1	127.0.0.1	TCP	66	25 → 45516 [ACK] Seq=115 Ack=82 Win=33280 Len=0 TSval=3
44	12.747155306	127.0.0.1	127.0.0.1	SMTP	88	C: DATA fragment, 22 bytes
45	12.747161136	127.0.0.1	127.0.0.1	TCP	66	25 → 45516 [ACK] Seq=115 Ack=104 Win=33280 Len=0 TSval=
46	14.437099618	127.0.0.1	127.0.0.1	SMTP	80	C: DATA fragment, 14 bytes
47	14.437108028	127.0.0.1	127.0.0.1	TCP	66	25 → 45516 [ACK] Seq=115 Ack=118 Win=33280 Len=0 TSval=
48	14.437167027	127.0.0.1	127.0.0.1	SMTP	68	C: DATA fragment, 2 bytes
49	14.437171187	127.0.0.1	127.0.0.1	TCP	66	25 → 45516 [ACK] Seq=115 Ack=120 Win=33280 Len=0 TSval=
50	16.175963816	127.0.0.1	127.0.0.1	SMTP	72	C: DATA fragment, 6 bytes
51	16.175971826	127.0.0.1	127.0.0.1	TCP	66	25 → 45516 [ACK] Seq=115 Ack=126 Win=33280 Len=0 TSval=
52	17.692959888	127.0.0.1	127.0.0.1	SMTP	72	C: DATA fragment, 6 bytes
53	17.692966048	127.0.0.1	127.0.0.1	TCP	66	25 → 45516 [ACK] Seq=115 Ack=132 Win=33280 Len=0 TSval=
54	18.467033525	127.0.0.1	127.0.0.1	SMTP	68	C: DATA fragment, 2 bytes
55	18.467041275	127.0.0.1	127.0.0.1	TCP	66	25 → 45516 [ACK] Seq=115 Ack=134 Win=33280 Len=0 TSval=
56	18.467094784	127.0.0.1	127.0.0.1	SMTP/I…	69	from: em1, subject: as1, , li11 , li12 , C: .
57	18.467099294	127.0.0.1	127.0.0.1	TCP	66	25 → 45516 [ACK] Seq=115 Ack=137 Win=33280 Len=0 TSval=
58	18.468001253	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok
59	18.513810249	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=137 Ack=123 Win=33280 Len=0 TSval=
60	33.238765696	127.0.0.1	127.0.0.1	SMTP	82	C: MAIL FROM: em2
61	33.239258540	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok
62	33.239263860	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=153 Ack=131 Win=33280 Len=0 TSval=
63	35.674738050	127.0.0.1	127.0.0.1	SMTP	82	C: RCPT TO: rec21
64	35.675185385	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok
65	35.675190965	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=169 Ack=139 Win=33280 Len=0 TSval=
66	37.652715550	127.0.0.1	127.0.0.1	SMTP	82	C: RCPT TO: rec22
67	37.653153485	127.0.0.1	127.0.0.1	SMTP	74	S: 250 Ok
68	37.653158714	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=185 Ack=147 Win=33280 Len=0 TSval=
69	38.238689293	127.0.0.1	127.0.0.1	SMTP	72	C: DATA
70	38.239143507	127.0.0.1	127.0.0.1	SMTP	103	S: 354 End data with <CR><LF>.<CR><LF>
71	38.239149857	127.0.0.1	127.0.0.1	TCP	66	45516 → 25 [ACK] Seq=191 Ack=184 Win=33280 Len=0 TSval=

En verde aparece el primer mensaje y en amarillo el segundo. Como podemos observar al principio se establece la conexión pero entre medio de cada mensaje no se cierra y se vuelve a abrir. Confirmamos que puede ser concurrente.