72.07 Protocolos de Comunicación

1er Cuatrimestre 2022 - ITBA

Trabajo Práctico Especial: "Servidor Proxy SOCKS Versión 5"

Grupo 3

Integrantes

Federico Gustavo Rojas Pelliccia - 60239
Leonardo Agustín D'Agostino - 60335
Santiago Sandrini - 61447
Roberto Franco Rodriguez Tulasne - 60089

Índice

1. Introducción

El objetivo de este trabajo fue implementar un servidor proxy para el protocolo SOCKSv5, y a su vez desarrollar e implementar un protocolo propio para la administración de dicho servidor.

2. Protocolos Utilizados

2.1 SOCKS V5

RFC-1928 RFC-1929

Al necesitar desarrollar un servidor proxy para dicho protocolo, naturalmente se debió implementar el "protocolo SOCKS versión 5" utilizando el RFC-1928 que lo describe. Además como el protocolo tiene la opción de autenticación de usuario y contraseña, se siguieron los lineamientos detallados en el RFC-1929. Protocolo en capa 5 del modelo OSI (capa de sesión).

2.2 TCP

RFC-793

Siguiendo el RFC-1928 de SOCKSv5, se especifica que el protocolo de transporte que se utiliza es TCP. Protocolo en capa 4 del modelo OSI (capa de transporte).

2.3 POP3

RFC-1939

Dado que nuestro servidor proxy SOCKSv5 debe tener un registro de sniffing de credenciales similar a ettercap sobre el protocolo POP3, se tuvo que implementar un parser de la autenticación de usuario en el mismo. Protocolo en capa 1 del modelo OSI (capa de aplicación).

2.4 UDP

RFC-768

Tomando en consideración que no es necesario mantener una sesión abierta en el servidor de administración, y teniendo en cuenta además que todas las funciones planteadas pueden manejarse en un solo datagrama, con excepción del listado de usuarios (se detalla cómo se contempla esto en la sección 5) se decidió desarrollar el protocolo propio utilizando UDP como protocolo de transporte. Aprovechando las ventajas del datagrama UDP se pudo simplificar considerablemente la implementación de los pedidos (requests) del protocolo. Protocolo en capa 4 del modelo OSI (capa de transporte).

2.5 DNS

RFC-1035

El protocolo DNS es utilizado por la función getaddrinfo(), para resolver un pedido al proxy con un fqdn. Protocolo en capa 1 del modelo OSI (capa de aplicación).

2.6 DOG

Protocolo propio desarrollado e implementado para ofrecer un servicio de administración del servidor proxy SOCKSv5. Su especificación se encuentra en la siguiente sección. Protocolo en capa 1 del modelo OSI (capa de aplicación).

3. DOG - Protocolo de administración de servidor SOCKS v5

3.1 Introducción

Siguiendo con la tradicional familia de software DOG, se desarrolló e implementó el 'DOG protocol', el cual permite a un administrador monitorear métricas, modificar valores pertinentes y realizar un manejo de usuarios sobre el servidor proxy SOCKSv5.

El mismo es un protocolo binario, no orientado a sesión el cual funciona sobre UDP.

El creador del servidor debe disponer de una variable de entorno de 4 bytes llamada DOG_TOKEN, la cual se utilizará como clave de acceso al servicio de administración del proxy. Un cliente del servicio deberá también crear la misma variable de entorno para que los pedidos sean aceptados por el servidor del servicio.

3.2 Especificaciones del protocolo

La unidad para los valores indicados en los datagramas y estructuras es el byte y la codificación de los mismos es en big endian. Los valores son del tipo unsigned int.

Los datagramas de pedido al servidor tienen la siguiente estructura:

DOG_HEADER	DATA
10	VARIABLE

En donde DOG_HEADER tiene la siguiente estructura:

VER	TYPE	CMD	ID	TOKEN
1	1	2	2	4

El valor VER refiere a la versión utilizada del protocolo. Se debe usar la versión 1 con el valor 0x01.

El valor TYPE refiere al tipo de comando a utilizar.

Los valores válidos para TYPE son los siguientes:

0x00 para un pedido Get (consultar)

0x01 para un pedido Alter (modificar)

El valor CMD refiere a un comando específico para el TYPE pedido.

Los valores válidos para CMD con TYPE Get (0x00) son:

- 0x00 para ver página de la lista de usuarios
- 0x01 para ver la cantidad histórica de conexiones al servidor
- 0x02 para ver la cantidad de conexiones concurrentes
- 0x03 para ver la cantidad de bytes transferidos
- 0x04 para ver el estado del sniffing de contraseñas sobre POP3
- 0x05 para ver el estado de autenticación en el servidor
- 0x06 para ver el tamaño de página actual del listado de usuarios

Los valores válidos para CMD con TYPE Alter (0x01) son:

- 0x00 para agregar un usuario
- 0x01 para borrar un usuario
- 0x02 para encender/apagar el sniffer de contraseñas sobre POP3
- 0x03 para encender/apagar la autenticación en el servidor
- 0X04 para especificar el tamaño de página de listado de usuarios

El valor ID identifica al pedido al servidor y debe coincidir con el ID en la respuesta.

El valor TOKEN es una clave para poder utilizar el servicio, solo los administradores del servidor deben tener esta credencial.

El valor DATA es de tamaño variable para poder enviar los argumentos necesarios para utilizar el servicio (TYPE + CMD) requerido. El campo puede ser vacío o estar formado por enteros sin signo en codificación big endian: de 8 bits, de 16 bits o de 32 bits. También se aceptan strings null-terminated. En el caso de que DATA deba ser vacía, se puede enviar cualquier argumento y no será tenido en cuenta.

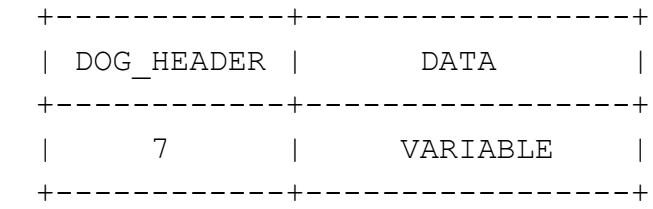
En el caso de pedido de listado de usuarios se debe contemplar que una página por defecto contiene hasta 200 usuarios. Nótese que en el caso de que todos estos usuarios utilicen la longitud máxima permitida por SOCKSv5 en credencial de nombre, el tamaño de esta página sería de 255 char * 200 usuarios = 51 000 bytes que, agregando los campos adicionales de pedido, queda muy por debajo del tamaño máximo de datos soportado por un datagrama UDP: aproximadamente 64 KB, dejando así espacio libre para transportar algún otro dato imprevisto.

También se dispone de un comando para cambiar el tamaño de página entre 1 y 200 usuarios por página.

Tabla de operaciones de referencia para **pedidos** al servidor

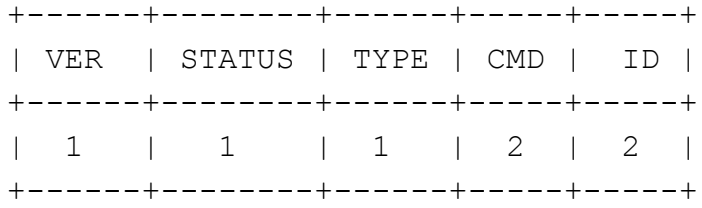
Comando	**Tipo (TYPE)**	**Valor de comando (CMD)**	**Argumentos (DATA)**
Ver página de lista de usuarios	0x00 (Get)	0x00	Uint de 8 bits Página solicitada
Ver cantidad histórica de conexiones	0x00 (Get)	0x01	-
Ver cantidad de conexiones concurrentes	0x00 (Get)	0x02	-
Ver cantidad de bytes transferidos	0x00 (Get)	0x03	-
Ver estado del sniffing de contraseñas sobre POP3	0x00 (Get)	0x04	-
Ver estado de autenticación en el servidor	0x00 (Get)	0x05	-
Ver tamaño de página de listado de usuarios	0x00 (Get)	0x06	-
Agregar un usuario	0x01 (Alter)	0x00	String null-terminated “nombre:contraseña\0”
Borrar un usuario	0x01 (Alter)	0x01	String null-terminated “nombre\0”
Encender o apagar sniffer de contraseñas sobre POP3	0x01 (Alter)	0x02	Uint de 8 bits 1 - encender 0 - apagar
Encender o apagar autenticación	0x01 (Alter)	0x03	Uint de 8 bits 1 - encender 0 - apagar
Cambiar tamaño de página de listado de usuarios	0x01 (Alter)	0x04	Uint de 8 bits 1 - mínimo 200 - máximo

Los datagramas de respuesta del servidor tienen la siguiente estructura:



DOG_HEADER	DATA
7	VARIABLE

Donde DOG_HEADER tiene la siguiente estructura:



VER	STATUS	TYPE	CMD	ID
1	1	1	2	2

El valor VER debe ser el mismo explicitado en la sección de datagrama de pedido, a menos que la versión especificada en el mismo no sea válida donde se debe retornar una respuesta con STATUS 0x01 de versión inválida.

El valor STATUS refiere al código de respuesta del servidor, donde los valores posibles son los siguientes:

- 0x00 indica que el pedido fue exitoso
- 0x01 indica que el pedido lleva una versión inválida
- 0x02 indica que el token en el pedido es inválido
- 0x03 indica que se encontró un TYPE inválido en el pedido
- 0x04 indica que se encontró un CMD inválido en el pedido
- 0x05 indica que se encontró un argumento inválido en el pedido
- 0x06 indica que el servidor está lleno, no acepta nuevos usuarios
- 0x07 indica que el usuario que se intentó agregar ya se encuentra registrado en el servidor
- 0x08 indica que el usuario que se quiso eliminar no fue encontrado
- 0x09 indica que hubo un fallo general en el servidor

Los valores TYPE y CMD coinciden con los valores indicados en el pedido. La decisión de incluir estos valores en la respuesta es para facilitar la elección de tamaño del campo de tamaño variable DATA.

El valor ID debe corresponder con el valor enviado en el pedido al cual se está respondiendo para poder identificarlo.

El valor DATA se utiliza para enviar la información adicional de respuesta, de ser necesaria, siguiendo los lineamientos establecidos en el datagrama de pedido. Es decir se utilizan los mismos tipos de datos y en caso de no necesitar enviar información adicional al STATUS, se deja vacío.

Tabla de operaciones de referencia para **respuestas** del servidor

Comando	**Tipo (TYPE)**	**Valor de comando (CMD)**	**Respuesta(DATA)**
Ver lista de usuarios	0x00 (Get)	0x00	Secuencia de strings separados por '\n' terminada en null Ej: "usu1\nusu2\0"
Ver cantidad histórica de conexiones	0x00 (Get)	0x01	Uint de 32 bits Cantidad
Ver cantidad de conexiones concurrentes	0x00 (Get)	0x02	Uint de 16 bits Cantidad
Ver cantidad de bytes transferidos	0x00 (Get)	0x03	Uint de 32 bits Cantidad
Ver estado del sniffing de contraseñas sobre POP3	0x00 (Get)	0x04	Uint de 8 bits 1 - encendido 0 - apagado
Ver estado de autenticación en el servidor	0x00 (Get)	0x05	Uint de 8 bits 1 - encendido 0 - apagado
Agregar un usuario	0x01 (Alter)	0x00	-
Borrar un usuario	0x01 (Alter)	0x01	-
Encender o apagar sniffer de contraseñas sobre POP3	0x01 (Alter)	0x02	-
Encender o apagar autenticación	0x01 (Alter)	0x03	-

4. Aplicaciones desarrolladas

4.1 Main

La aplicación main es la encargada de iniciar los sockets pasivos IPv4 e IPv6 para el servidor proxy y los sockets ipv4 e ipv6 para el administrador. Para eso primero se debieron parsear los argumentos de entrada y así verificar si se deben iniciar los sockets en sus direcciones estándar o en alguna especial indicada por el usuario. Los 4 sockets pasivos siempre se inicializan en una dirección estándar (0.0.0.0 ::0 para el SOCKSv5 y 127.0.0.1 ::1 para el DOG manager), queda a cargo del usuario indicar si quiere que algún socket se inicialice en alguna dirección especial utilizando el comando -l o -L al crear el servidor. Si no se pudo crear al menos un socket pasivo para el proxy o para el administrador luego la ejecución termina.

Luego de crear los socket, se crea un selector al cual se suscriben el timeout y handlers para el proxy y administrador, comenzando la ejecución de las rutinas respectivas para los mismos.

4.2 Proxy SOCKSv5

4.2.1 Funcionamiento general

El servidor fue implementado utilizando un selector de manera no bloqueante que permite monitorear un máximo de 1024 file descriptors por la utilización de la syscall “pselect”.

En base al estado actual de la conexión (ver sección 13.2), el selector permite suscribir alguno de los fd a cierto interés: lectura, escritura o ninguna operación. Además, a cada fd se le asigna un handler para las acciones a llevar a cabo.

La conexión entre el cliente y el proxy se realiza en distintos pasos definidos en el RFC-1928, primero se envía un

paquete “HELLO” de identificación del protocolo y de los métodos de autenticación del usuario. Luego, el servidor elige el método de autenticación que desea utilizar, en caso de tener la autenticación prendida solo permite accesos con autenticación, y en caso de no ponerse de acuerdo con el usuario, le responde con el mensaje de error correspondiente y cierra la conexión.

Si la autenticación se encuentra prendida, el próximo paso sería que el cliente envíe sus credenciales para autenticarse con el servidor.

Finalmente, si la autenticación fue exitosa o si se encontraba apagada, el cliente envía su pedido “REQUEST” de conexión y el servidor intenta crear la conexión con el servidor destino especificado. Para lograr dicho objetivo, el proxy puede tomar dos caminos:

- En caso de que el cliente haya especificado una dirección IPv4 o IPv6, se intentará establecer la conexión. Si la misma es exitosa o no, se le informa al cliente.

- En caso de que el cliente haya especificado un FQDN, se utiliza la función “getaddrinfo” sobre un hilo aparte para resolver el fqdn especificado, ya que esta tarea es bloqueante.

Los pedidos de conexión junto con su estado de respuesta y demás información relevante quedan registrados en la salida estándar del servidor.

Una vez realizada la conexión con el destino, los paquetes enviados tanto por el cliente como por el destino son enviados a través del proxy y redirigidos correspondientemente.

El proxy corre en un único hilo (a excepción de cuando se solicita un fqdn) utilizando el selector mencionado anteriormente para atender a diversos fds cuando estén listos.

Adicionalmente, el proxy cuenta con un sniffer de credenciales POP3 prendido por defecto, aunque puede decidirse crear el servidor con el sniffer apagado o desde el administrador de servidor DOG también se puede cambiar y además visualizar su estado.

El sniffer POP3 "parsea" los paquetes correspondientes que pasan por el proxy, el primer paquete enviado pertenece al servidor POP3 destino y contiene el mensaje "+OK".
Una vez identificado el paquete, se continúa "parseando" por los mensajes "USER <nombre>" y "PASS <contraseña>". Si luego de "parsear" estos mensajes se encuentra un "+OK" luego las credenciales ingresadas eran válidas y se muestra un log del lado del servidor proxy con la información. De lo contrario se vuelve a esperar que se "parseen" los mensajes "USER <nombre>" y "PASS <contraseña>". Nuestra implementación no fue hecha para soportar pipelining (POP3 soporta pipelining), aunque la misma puede sniffear este tipo de comandos pero imprime el resultado con basura de los comandos ingresados.

Finalmente, cuando el cliente o servidor proxy cierra la conexión, se liberan todos los recursos reservados. Si el administrador del servidor decide cerrar el proxy, se cierran todas las conexiones abiertas y se liberan todos los recursos.

4.2.2 Métricas para el monitoreo del sistema

Se implementó una estructura de datos llamada socks5_stats con el fin de registrar las **conexiones históricas**, **concurrentes**, y también la **cantidad de bytes transferidos** durante el tiempo de vida del servidor.

Para monitorear la cantidad de conexiones utilizamos un mecanismo que incrementa la cantidad de conexiones concurrentes cada vez que el socket SOCKS5 pasivo acepta una conexión de cliente. Este a su vez es encargado de incrementar las conexiones históricas del servidor. En el caso de la cantidad de transferidos, es importante aclarar que se tienen en cuenta los

bytes que se transfieren durante todas los estados, es decir, los bytes transferidos durante el saludo, durante la autenticación (si la hay), durante la request y reply socks5 y finalmente durante el envío de la información al servidor origin y del origin al cliente.

4.2.3.Decisiones de implementación

- Se considera que cuando el servidor es ejecutado indicando usuarios (por ejemplo ./socks5d -u usr:pass) la autenticación está encendida. Esto provoca que no se pueda utilizar el servidor proxy sin autenticarse correctamente.

- En caso de que el servidor sea ejecutado sin especificar ningún usuario (por ejemplo ./socks5d), entonces se considera que la autenticación está apagada. En consecuencia, el servidor proxy no considerará si el usuario está autenticado o no, pudiendo hacer pedidos independientemente de si al hacer la petición se aclara un usuario o no.

- Los 4 sockets (SOCKSv5 + DOG manager) se crean SIEMPRE en su dirección default salvo que para cada uno se especifique una dirección válida.
  Por ejemplo: si se especifica una dirección ipv4 para el socket del SOCKSv5, la dirección ipv6 del mismo seguirá siendo la default.

4.3 Administrador de servidor DOG

Siguiendo los lineamientos especificados en la **sección 3**, el servidor implementado posee dos sockets para recibir peticiones del protocolo DOG, procesarlas y enviar una respuesta según corresponda. Además, se implementó un cliente para el protocolo del estilo terminal, el cual posee un comando help que describe los distintos comandos del cliente:

```
ssandrini@hp-laptop:~/ITBA/Protos/TPE/SOCKSv5-Proxy-Server$ ./dog 0.0.0.0 8080
dog client >> help
+------------+--------------------+-------------------------------------------------------------------------+
|  Command   |       Usage        |                              Description                                |
+------------+--------------------+-------------------------------------------------------------------------+
| list       | list <page_number> | Returns the specified page of the list of users registered on the server |
| hist       | hist               | Returns the amount of historic connections over the server              |
| conc       | conc               | Returns the amount of concurrent connections over the server            |
| bytes      | bytes              | Returns the amount of bytes transfered over the server                  |
| checksniff | checksniff         | Returns the status of the password disector over the server             |
| checkauth  | checkauth          | Returns the status of authentication over the server                    |
| getpage    | getpage            | Returns the amount of users per page (max 200)                          |
| add        | add user:pass      | Command to add a user                                                   |
| del        | del user           | Command to delete a user                                                |
| sniff      | sniff on/off       | Command to toggle POP3 credential sniffer over the server               |
| auth       | auth on/off        | Command to toggle authentication over the server                        |
| setpage    | setpage <page_size> | Command to set page size (between 1 and 200)                           |
+------------+--------------------+-------------------------------------------------------------------------+
```

Es importante aclarar que para el correcto funcionamiento, tanto el administrador del servidor como la persona que usa el cliente, deben generar una variable de entorno llamada DOG_TOKEN de 4 bytes, que es la que se utiliza para enviar el token en las requests DOG.

4.4 Logging

La información de las conexiones y credenciales POP3 “sniffeadas” se imprime en la pantalla (STDOUT) del servidor. Se imprime la fecha de la conexión (en formato ISO-8601), el nombre de usuario del proxy que la inició (si el mismo se encuentra registrado), una letra especificando el tipo de log (A para acceso y P de POP3), la dirección IP y puerto del cliente y la dirección IP y puerto destino, en caso de ser un log de POP3, también se incluyen las credenciales sniffear y finalmente, el estado respuesta del servidor proxy.

5. Problemas encontrados

5.1 Decisiones sobre el protocolo

Al comenzar el TPE, y basándonos en que ya debíamos implementar autenticación para nuestro servidor proxy SOCKSv5, se había pensado en qué DOG protocol sea un protocolo orientado a sesión, que funcionase sobre TCP aprovechando la negociación de autenticación ya implementada. Pero posteriormente, se llegó a la conclusión de que lo más razonable para la implementación del protocolo considerando la funcionalidad que debe proveer fue que no fuese orientado a sesión y que el mismo funcionara sobre UDP.

Las razones por las que se tomó esta decisión son las siguientes:

- No hay necesidad de mantener una sesión abierta con el administrador de servidor. La autenticación de un usuario que debe poder acceder al servicio se puede resolver con un token clave administrado mediante una variable de entorno.
- Tanto los pedidos como las respuestas del administrador entran en un datagrama por lo que no habría peligro en perder paquetes o que los mismos lleguen en un orden incorrecto.
- Para solucionar la pérdida de confiabilidad al utilizar UDP, se implementó un timeout en el cliente para re-solicitar un paquete si el mismo no llegó en un tiempo determinado.
- El único comando que podría no entrar en un datagrama es el de listar usuarios, por lo que se decidió agregar una paginación de usuarios en el protocolo, esperando un argumento de página al solicitar el listado, para no tener una restricción de tamaño de lista originada por el tamaño de datagrama UDP. El paginado se implementó en el servicio de administración.

- Aprovechando el envío de un único datagrama UDP, mayor simplicidad a la hora de "parsear" los pedidos e implementarlos.
- UDP es más rápido que TCP para el envío de información (no hay conexión ni acknowledges), lo cual es beneficioso para respuestas simples y veloces como las esperadas en el servicio de administración.

5.2 Pruebas de estrés y load sobre el servidor

Al nunca haber realizado pruebas de estrés o load, se presentó una traba a la hora de verificar el cumplimiento del enunciado (al menos 500 conexiones concurrentes) y los límites de nuestro servidor. Finalmente, luego de invertir más tiempo del esperado, se pudo utilizar Jmeter correctamente junto a un script en bash para alcanzar estos objetivos.

6. Limitaciones

- La implementación del servidor soporta hasta 10 usuarios. Aunque esta limitación existe en la implementación actual, el protocolo de administración soporta escalabilidad en la cantidad de usuarios al implementar paginación en los pedidos de listado de usuario.

- La cantidad de conexiones simultáneas del servidor proxy se encuentra limitada por la cantidad de file descriptors que es capaz de administrar la syscall “pselect”, utilizada en el selector que provee soporte a la multiplexación de conexiones. El límite es de 1024 fds por lo que teóricamente se puede calcular que la cantidad de conexiones simultáneas máxima es de 509 = (1024 - 4 sockets (proxy + DOG) - 2 (stderr + stdout)) / 2 (1 fd para conectarse al proxy y 1 fd para que el proxy se conecte al origen). Una vez calculado y comprobado (ver sección 9.2) este valor fue utilizado en el código para prevenir que se acepten más de 509 conexiones simultáneas, lo que generaría una escasez de fds para las conexiones a destino. De esta forma si hay más de 509 pedidos, a medida que se resuelvan los primeros, los siguientes serán tomados.

7. Posibles extensiones

7.1 DOG protocol

Entre las posibles extensiones pensadas para nuevas versiones del DOG protocol se encuentran:

- Cambiar y ver el tamaño del buffer de lectura/escritura sobre el servidor.
- Métricas para cada usuario individualmente (por ej cuantos bytes fueron transferidos por el usuario 'usr1', o cuando fue la última conexión de un usuario).
- Cambiar y ver el valor del timeout del servidor.

7.2 Servidor SOCKSv5

En cuanto al servidor proxy SOCKS5 implementado se podría considerar en nuevas implementaciones:

- Aumentar la cantidad de usuarios (autenticación) máxima del servidor
- Utilizar un hashmap para el guardado de usuarios en el servidor. Esto daría mayor eficiencia al realizar operaciones de búsqueda de un usuario en particular
- Poseer distintos credential sniffers sobre distintos protocolos (Ej sobre HTTP).

8. Conclusiones

Al finalizar el TPE los integrantes del grupo estuvimos de acuerdo en lo esencial que fue el trabajo realizado para poder consolidar los conocimientos obtenidos durante la cursada. Al tener que embarrarse en la implementación de un protocolo existente y en el desarrollo y posterior implementación de un protocolo propio, se pudo abordar las problemáticas y decisiones discutidas en clase de una forma desarrollativa que no puede obtenerse de un enfoque teórico o práctico (utilización de los protocolos ya implementados) como únicamente poseíamos previo a la realización del TPE. Ir más allá de la lectura del RFC e implementar un protocolo a través del servidor SOCKSv5 nos permitió comprender en mayor profundidad las decisiones que se deben considerar y tomar a la hora de desarrollar un protocolo.

El trabajo práctico especial presentó muchos obstáculos para alcanzar nuestros objetivos sobre el mismo, sin embargo el equipo se encuentra satisfecho con el resultado obtenido y con un mayor entendimiento del contenido abordado en la materia.

9. Ejemplos de prueba

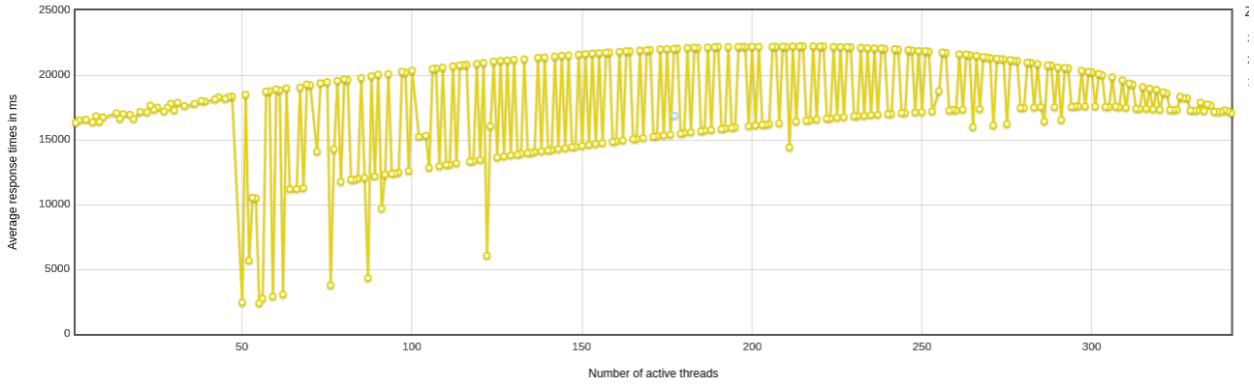
9.1 Pruebas de estrés

Para la prueba de estrés se utilizó la herramienta Jmeter con 500 hilos con ramp-up de 25 segundos (es decir 20 pedidos por segundo) los cuales hacían pedidos a un archivo de 1 MB montado en un servidor nginx local. El objetivo de este test es observar cómo se deteriora el servidor ante mayor cantidad de conexiones.

Requests	Executions		
Label	#Samples	FAIL	Error %
Total	**500**	**0**	**0.00%**
bar	500	0	0.00%

Response Times (ms)						
Average	Min	Max	Median	90th pct	95th pct	99th pct
16864.31	**2377**	**22248**	**19085.00**	**22121.90**	**22202.70**	**22233.99**
16864.31	2377	22248	19085.00	22121.90	22202.70	22233.99

Throughput	Network (KB/sec)	
Transactions/s	Received	Sent
12.22	**12514.30**	**1.26**
12.22	12514.30	1.26



Como se puede ver en el gráfico, el peor momento de respuesta ocurre aproximadamente en la mitad del mismo cuando la

mayor parte de las conexiones fueron establecidas. Conforme avanza el gráfico y se resuelven conexiones, las mismas se cierran provocando que el tiempo de respuesta se reduzca, acelerando el tiempo de resolución. Se puede ver que inicialmente las respuestas tardan aproximadamente 16 segundos, luego en el peor momento 22.248 segundos, y al finalizar nuevamente unos 16 segundos, un 30% menos que en el peor momento.

9.2 Load test

Para estos tests se buscó ver cómo se degrada el servidor a mayor cantidad de requests. En orden de lograr esto se utilizó el programa Jmeter y se realizaron 3 experiencias, en la primera se utilizaron 100 hilos, en la segunda 300 hilos y en la última se usaron 500 hilos. Todas estas experiencias utilizaron un ramp-up de 0 (es decir, que todos los hilos arranquen lo antes posible) y una sola iteración. Todas estas experiencias se realizaron configurando para que las requests usen nuestro proxy y que hicieran su pedido a un archivo local de 500k montado en nginx.

100 HILOS

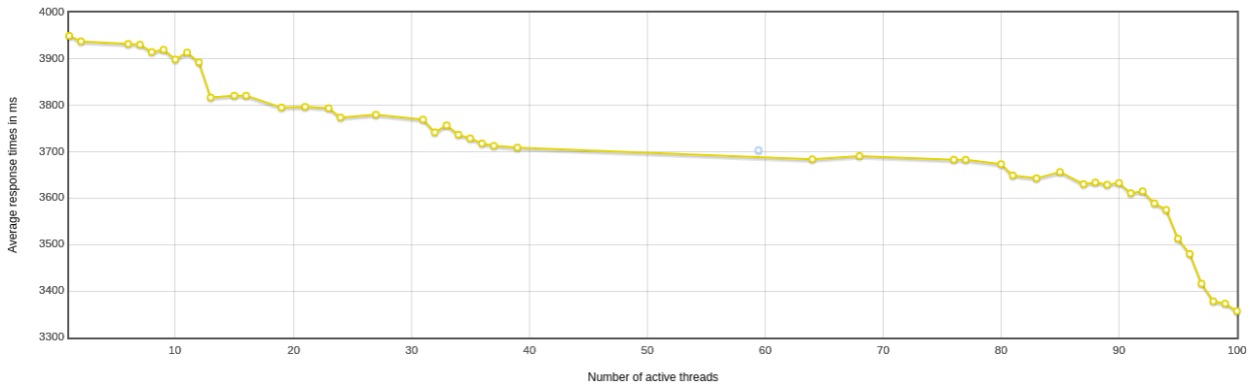
Estadísticas generales:

Requests	Executions		
Label	#Samples	FAIL	Error %
Total	**100**	**0**	**0.00%**
bar	100	0	0.00%

Response Times (ms)						
Average	Min	Max	Median	90th pct	95th pct	99th pct
3703.72	**3357**	**3950**	**3681.00**	**3912.50**	**3930.90**	**3949.88**
3703.72	3357	3950	3681.00	3912.50	3930.90	3949.88

Throughput	Network (KB/sec)	
Transactions/s	Received	Sent
25.27	12642.62	2.62
25.27	12642.62	2.62

Gráfico que muestra el tiempo de respuesta en función de los hilos activos:



300 HILOS

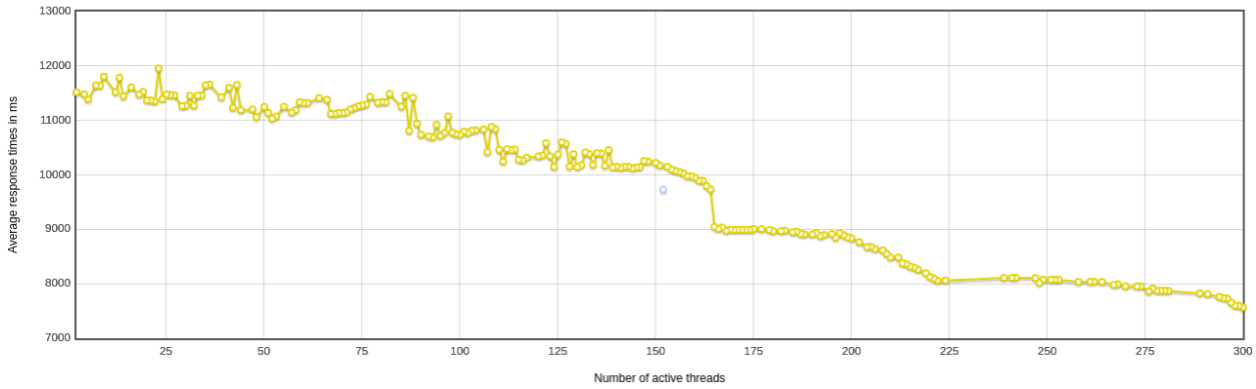
Estadísticas generales:

Requests	Executions		
Label	#Samples	FAIL	Error %
Total	300	0	0.00%
bar	300	0	0.00%

Response Times (ms)						
Average	Min	Max	Median	90th pct	95th pct	99th pct
9729.40	7568	11955	10138.00	11455.90	11512.80	11804.96
9729.40	7568	11955	10138.00	11455.90	11512.80	11804.96

Throughput	Network (KB/sec)	
Transactions/s	Received	Sent
24.64	12325.93	2.55
24.64	12325.93	2.55

Gráfico que muestra el tiempo de respuesta en función de los hilos activos:



500 HILOS

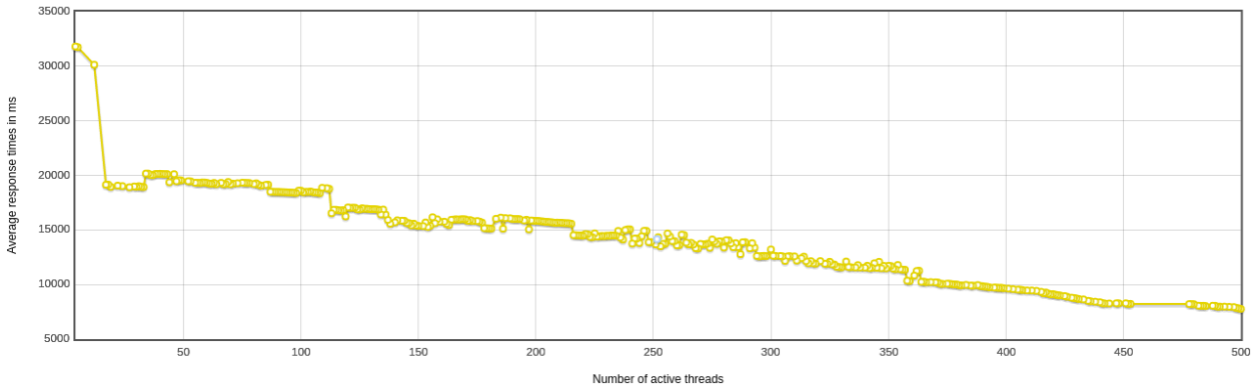
Estadísticas generales:

Requests	Executions		
Label	#Samples	FAIL	Error %
Total	500	0	0.00%
bar	500	0	0.00%

Response Times (ms)						
Average	Min	Max	Median	90th pct	95th pct	99th pct
14159.97	7770	31819	14126.50	19271.50	19549.55	31731.07
14159.97	7770	31819	14126.50	19271.50	19549.55	31731.07

Throughput	Network (KB/sec)	
Transactions/s	Received	Sent
15.18	7593.40	1.57
15.18	7593.40	1.57

Gráfico que muestra el tiempo de respuesta en función de los hilos activos:



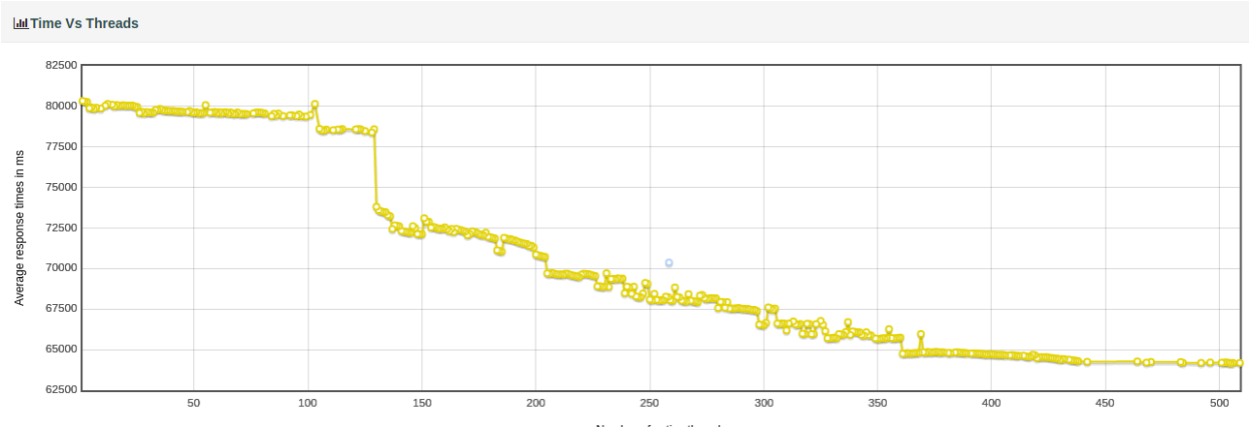
Como se puede observar tanto en los gráficos como en las tablas, es claro que el tiempo de respuesta del servidor se deteriora significativamente a mayor cantidad de pedidos. El throughput pasó de 25 transacciones por segundo (en la experiencia de 100) a 15 (en la experiencia de 500). No obstante, hay que tener en cuenta que estamos comparando 100 hilos frente a 500, es decir con 5 veces más conexiones las transacciones por segundo bajaron un 40%, lo cual consideramos que es una buena relación.

AUMENTO A 509 HILOS - LÍMITE TEÓRICO CALCULADO

Luego para finalizar las pruebas de load se ejecutó la misma prueba pero con el límite máximo calculado de 509 conexiones simultáneas y se obtuvieron los resultados esperados, ninguna falla:

Requests	Executions		
Label	#Samples	FAIL	Error %
Total	**509**	**0**	**0.00%**
bar	509	0	0.00%

Gráfico mostrando el tiempo de respuesta en función de los hilos activos análogo a los gráficos para casos anteriores:



En esta oportunidad se utilizó la siguiente limitación de descarga en el archivo jmeter.properties:
httpclient.socket.http.cps=8000
httpclient.socket.https.cps=8000

9.3 Pruebas de performance e integridad

En el siguiente test se verificó como foco del test la integridad de datos transportados pero también la velocidad del proxy implementado. Para lograr esto se descargó una imágen ISO (imágen de una versión vieja de Ubuntu) de 2GB utilizando curl con el comando time "pipeado" a sha1sum. Se hizo el mismo pedido primero con y luego sin el proxy y como se ve el tiempo de descarga fue análogo si se considera la diferencia de velocidad de descarga promedio y como se puede ver, la integridad de los datos es correcta al coincidir ambos checksum del sha1sum.

```
ficored@DESKTOP-76MVC3O:~/winhomef/downloads$ time curl -x socks5://localhost:1080 http://old-releases.ubuntu.com/releases/19.04/ubun
tu-19.04-desktop-amd64.iso | sha1sum
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2000M  100 2000M    0     0  9077k      0  0:03:45  0:03:45 --:--:-- 9089k
47064866141c7729b3f447890dd6d5bc2fc35cf7  -

real    3m45.645s
user    0m18.346s
sys     0m16.601s
ficored@DESKTOP-76MVC3O:~/winhomef/downloads$ time curl http://old-releases.ubuntu.com/releases/19.04/ubuntu-19.04-desktop-amd64.iso
| sha1sum
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2000M  100 2000M    0     0  8940k      0  0:03:49  0:03:49 --:--:-- 10.4M
47064866141c7729b3f447890dd6d5bc2fc35cf7  -

real    3m49.091s
user    0m20.356s
sys     0m15.887s
```

El mismo test fue realizado con archivos de 1.1GB:

```
ficored@DESKTOP-76MVC3O:~/winhomef/downloads$ time curl -x socks5://127.0.0.1:1080 https://releases.ubuntu.com/14.04.6/ubuntu-14.04.6
-desktop-amd64.iso | sha1sum
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1104M  100 1104M    0     0  8787k      0  0:02:08  0:02:08 --:--:-- 3611k
2d3675f14a6884bb42917838a5c4246916fe73b5  -

real    2m8.679s
user    0m11.132s
sys     0m7.425s
```

```
ficored@DESKTOP-76MVC3O:~/winhomef/downloads$ time curl https://releases.ubuntu.com/14.04.6/ubuntu-14.04.6-desktop-amd64.iso | sha1su
m
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1104M  100 1104M    0     0  8912k      0  0:02:06  0:02:06 --:--:-- 8358k
2d3675f14a6884bb42917838a5c4246916fe73b5  -

real    2m6.864s
user    0m12.592s
sys     0m8.939s
```

y 632MB:

```
ficored@DESKTOP-76MVC3O:~/winhomef/downloads$ time curl -x socks5://127.0.0.1:1080 https://releases.ubuntu.com/14.04.6/ubuntu-14.04.6
-server-amd64.iso | sha1sum
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  632M  100  632M    0     0  8839k      0  0:01:13  0:01:13 --:--:-- 9342k
13bfe163ca8ad8a6e5676b0460ca60d03387ec24  -

real    1m13.239s
user    0m6.450s
sys     0m4.153s
```

```
ficored@DESKTOP-76MVC3O:~/winhomef/downloads$ time curl https://releases.ubuntu.com/14.04.6/ubuntu-14.04.6-server-amd64.iso | sha1sum
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  632M  100  632M    0     0  8540k      0  0:01:15  0:01:15 --:--:-- 8058k
13bfe163ca8ad8a6e5676b0460ca60d03387ec24  -

real    1m15.791s
user    0m7.543s
sys     0m4.647s
```

9.4 Herramientas de análisis de código

Para realizar un análisis de la calidad de código de las aplicaciones desarrolladas se utilizaron herramientas tanto para análisis dinámico como estático, como por ejemplo *valgrind*, *pvs-studio* y *cppcheck*. A su vez se utilizó *GCC* con el flag *-fsanitize=address* durante todo el desarrollo del proyecto. También se compiló y utilizó el proyecto en *pampero*.

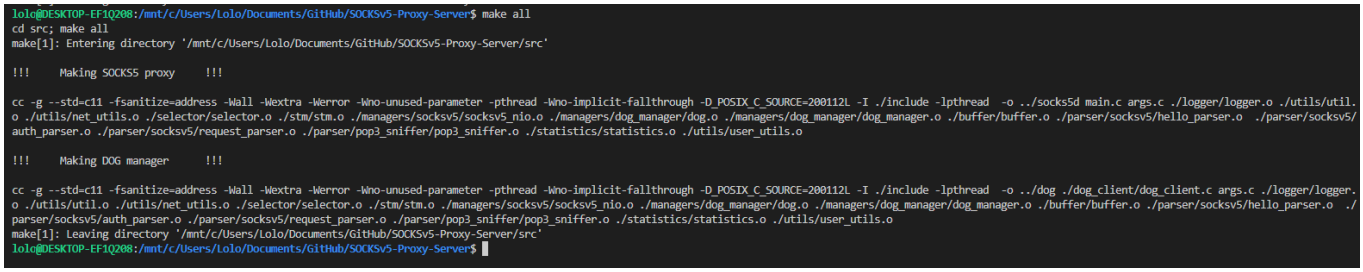
10. Instalación

Para la creación de los ejecutables necesarios del proyecto basta con posicionarse en la carpeta raíz del mismo y ejecutar el comando make all. De esta manera se crearán dos ejecutables, socks5d y dog. Una vez creados los ejecutables, es necesario crear la variable de entorno DOG_TOKEN tanto en la terminal donde se ejecutará el proxy SOCKSv5 como en la terminal en donde se ejecutará el cliente de administrador DOG, la misma consiste de un valor alfanumérico de 4 bytes. El fin de esta variable es proporcionar una clave requerida para que solo un usuario que la posea pueda usar el administrador. Cabe destacar que en ambas terminales el token debe coincidir; caso contrario el cliente no va a poder hacer pedidos(dado que tiene una clave incorrecta).
A continuación mostraremos los pasos a seguir:

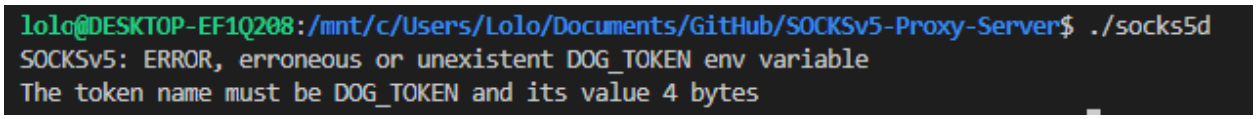
1- Dentro del directorio raíz del proyecto ejecutamos el comando make all para crear los ejecutables del servidor y del cliente:

```
lolc@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ make all
cd src; make all
make[1]: Entering directory '/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server/src'
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o logger/log
ger.o logger/logger.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o utils/util
.o utils/util.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o utils/net_
utils.o utils/net_utils.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o selector/s
elector.o selector/selector.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o stm/stm.o
stm/stm.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o managers/s
ocksv5/socksv5_nio.o managers/socksv5/socksv5_nio.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o managers/d
og_manager/dog.o managers/dog_manager/dog.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o managers/d
og_manager/dog_manager.o managers/dog_manager/dog_manager.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o buffer/buf
fer.o buffer/buffer.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o parser/soc
ksv5/hello_parser.o parser/socksv5/hello_parser.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o parser/socksv5/auth_parser.o parser/socksv5/auth_parser.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o parser/socksv5/request_parser.o parser/socksv5/request_parser.
c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o parser/pop3_sniffer/pop3_sniffer.o parser/pop3_sniffer/pop3_sn
iffer.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o statistics/statistics.o statistics/statistics.c
cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include   -c -o utils/user_utils.o utils/user_utils.c

!!!     Making SOCKS5 proxy     !!!

cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include -lpthread  -o ../socks5d main.c args.c ./logger/logger.o ./utils/util.
o ./utils/net_utils.o ./selector/selector.o ./stm/stm.o ./managers/socksv5/socksv5_nio.o ./managers/dog_manager/dog.o ./managers/dog_manager/dog_manager.o ./buffer/buffer.o ./parser/socksv5/hello_parser.o  ./parser/socksv5/
auth_parser.o ./parser/socksv5/request_parser.o ./parser/pop3_sniffer/pop3_sniffer.o ./statistics/statistics.o ./utils/user_utils.o

!!!     Making DOG manager      !!!

cc -g --std=c11 -fsanitize=address -Wall -Wextra -Werror -Wno-unused-parameter -pthread -Wno-implicit-fallthrough -D_POSIX_C_SOURCE=200112L -I ./include -lpthread  -o ../dog ./dog_client/dog_client.c args.c ./logger/logger.
o ./utils/util.o ./utils/net_utils.o ./selector/selector.o ./stm/stm.o ./managers/socksv5/socksv5_nio.o ./managers/dog_manager/dog.o ./managers/dog_manager/dog_manager.o ./buffer/buffer.o ./parser/socksv5/hello_parser.o  ./
parser/socksv5/auth_parser.o ./parser/socksv5/request_parser.o ./parser/pop3_sniffer/pop3_sniffer.o ./statistics/statistics.o ./utils/user_utils.o
make[1]: Leaving directory '/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server/src'
```

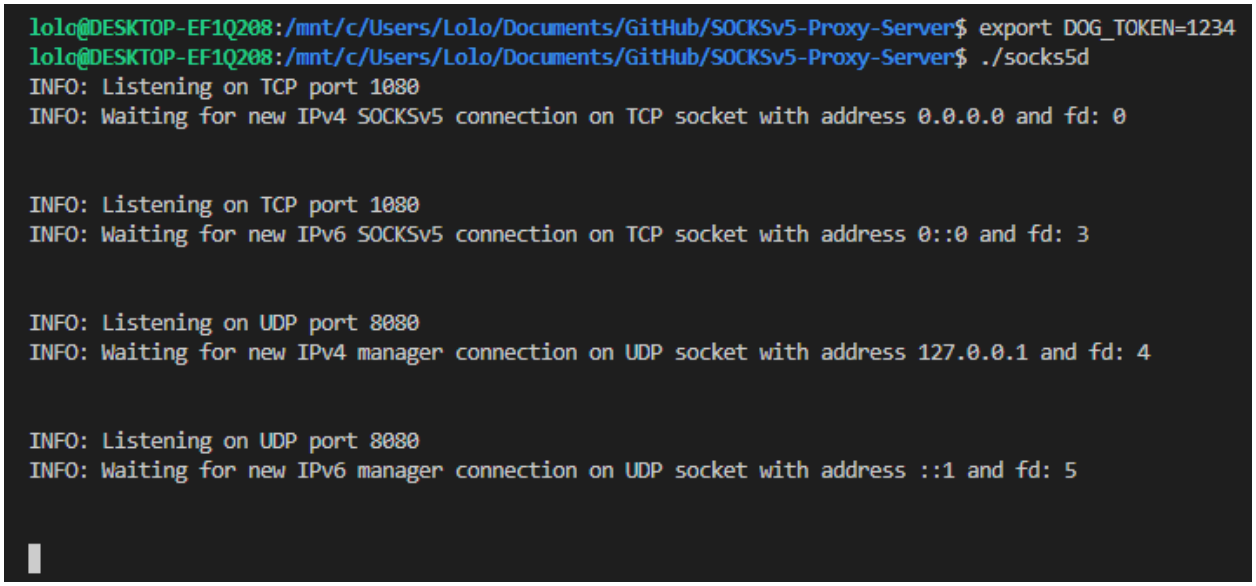
Nótese que si volvemos a ejecutar dicho comando no se realizarán de nuevo todas las compilaciones a menos que sea necesario:



2- Una vez creados los ejecutables nos podemos ver tentados a correr el servidor sin más, pero al hacerlo obtendremos lo siguiente:



Esto sucede por lo mencionado anteriormente, es necesario definir la variable DOG_TOKEN de la siguiente manera (ejemplo LINUX):



Esto es análogo para el caso de que se quiera ejecutar el administrador de servidor, con la particularidad de que el token debe ser el mismo que el definido en el servidor. En caso de que se tuviera el token incorrecto sucedería lo siguiente:

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ export DOG_TOKEN=abcd
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./dog 127.0.0.1 8080
dog client >> bytes
Error: Authentication failed.
dog client >>
```

Ahora si definimos correctamente el token:

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ export DOG_TOKEN=1234
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./dog 127.0.0.1 8080
dog client >> bytes
Amount of bytes transfered: 0
dog client >>
```

3- Por último basta con correr el servidor proxy especificando las direcciones y puertos deseados tanto para el proxy como para el administrador de servidor DOG, o en su defecto utilizando los default. Para mayor detalle ver la siguiente sección.

11. Instrucciones de configuración

Como ya se mencionó previamente, una vez compilados los ejecutables del servidor y del cliente, es necesario definir una variable de entorno llamada DOG_TOKEN la cual debe ser un valor de 4 bytes alfanumérico. Dicha variable debe estar definida tanto en la terminal en la que se ejecuta el servidor como en la terminal en la que se ejecuta el cliente de administrador de servidor(y sus valores deben coincidir, caso contrario los pedidos realizados por el usuario del administrador de servidor serán rechazados).

Una vez hecha esta configuración, **correr el servidor** es muy sencillo, solamente hay que ejecutarlo de la siguiente manera:

```
./socks5d
```

Esto iniciará el servidor proxy sin autenticación y el administrador de servidor en los puertos y direcciones por defecto:

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./socks5d
INFO: Listening on TCP port 1080
INFO: Waiting for new IPv4 SOCKSv5 connection on TCP socket with address 0.0.0.0 and fd: 0


INFO: Listening on TCP port 1080
INFO: Waiting for new IPv6 SOCKSv5 connection on TCP socket with address 0::0 and fd: 3


INFO: Listening on UDP port 8080
INFO: Waiting for new IPv4 manager connection on UDP socket with address 127.0.0.1 and fd: 4


INFO: Listening on UDP port 8080
INFO: Waiting for new IPv6 manager connection on UDP socket with address ::1 and fd: 5
```

En caso de no saber cómo cambiar alguna configuración específica se puede ejecutar el servidor proxy con el flag -h ver un listado de posibles configuraciones:

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./socks5d -h
Usage: ./socks5d [OPTION]...

   -h               Prints help info and finishes.
   -l <SOCKS addr>  Adress (ipv4 or ipv6) where SOCKS server will serve.
   -L <mng  addr>   Adress (ipv4 or ipv6) where server manager will serve.
   -p <SOCKS port>  Port for incoming connection on SOCKS server.
   -P <mng port>    Port for incoming connection on server manager
   -u <name>:<pass> Username and password of SOCKS server allowed users. Max 10.
   -N               Turn off POP3 credential sniffing.
   -v               Prints current version info and finishes.
```

En cuanto a la **ejecución del cliente**, es necesario que al ejecutarse se indique la dirección IP del servidor a la cual quiere conectarse y su puerto, caso contrario el programa finalizará luego de mostrar la siguiente salida:
Usage: ./dog <dog_server_addr> <dog_server_port>

En caso de ejecutar el cliente con la variable de entorno y parámetros correctos, se abrirá una terminal vinculada con el servidor en donde se puede ejecutar el comando 'help' para obtener mayor detalle de los comandos disponibles para la administración del servidor.

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./dog 127.0.0.1 8080
dog client >> help
+------------+--------------------+-------------------------------------------------------------------------+
|  Command   |       Usage        |                                Description                              |
+------------+--------------------+-------------------------------------------------------------------------+
| list       | list <page_number> | Returns the specified page of the list of users registered on the server |
| hist       | hist               | Returns the amount of historic connections over the server              |
| conc       | conc               | Returns the amount of concurrent connections over the server            |
| bytes      | bytes              | Returns the amount of bytes transfered over the server                  |
| checksniff | checksniff         | Returns the status of the password disector over the server             |
| checkauth  | checkauth          | Returns the status of authentication over the server                    |
| getpage    | getpage            | Returns the amount of users per page (max 200)                          |
| add        | add user:pass      | Command to add a user                                                   |
| del        | del user           | Command to delete a user                                                |
| sniff      | sniff on/off       | Command to toggle POP3 credential sniffer over the server               |
| auth       | auth on/off        | Command to toggle authentication over the server                        |
| setpage    | setpage <page_size> | Command to set page size (between 1 and 200)                           |
+------------+--------------------+-------------------------------------------------------------------------+
dog client >>
```

12. Ejemplos de configuración y monitoreo

12.1 Cambios de ip y puertos del servidor

```
ficored@DESKTOP-76MVC3O:~/winhomef/documents/SOCKSv5-Proxy-Server$ ./socks5d -P 9091 -p 1111 -l ::1 -l 127.0.0.1 -L 0.0.0.0 -L ::0
INFO: Listening on TCP port 1111
INFO: Waiting for new IPv4 SOCKSv5 connection on TCP socket with address 127.0.0.1 and fd: 0


INFO: Listening on TCP port 1111
INFO: Waiting for new IPv6 SOCKSv5 connection on TCP socket with address ::1 and fd: 3


INFO: Listening on UDP port 9091
INFO: Waiting for new IPv4 manager connection on UDP socket with address 0.0.0.0 and fd: 4


INFO: Listening on UDP port 9091
INFO: Waiting for new IPv6 manager connection on UDP socket with address ::0 and fd: 5
```

Ejecutamos el comando de netstat -nlp|grep socks5d (nótese que aparecen 4 resultados porque hay que considerar ipv6 e ipv4)

```
ficored@DESKTOP-76MVC3O:~/winhomef/documents/SOCKSv5-Proxy-Server$ netstat -nlp|grep socks5d
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 127.0.0.1:1111          0.0.0.0:*               LISTEN      1715/./socks5d
tcp6       0      0 ::1:1111                :::*                    LISTEN      1715/./socks5d
udp        0      0 0.0.0.0:9091            0.0.0.0:*                           1715/./socks5d
udp6       0      0 :::9091                 :::*                                1715/./socks5d
```

12.2 Registro de acceso

En esta oportunidad probamos correr el servidor con sus valores default y realizar un curl a foo.leak.com.ar a través del proxy:

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./socks5d
INFO: Listening on TCP port 1080
INFO: Waiting for new IPv4 SOCKSv5 connection on TCP socket with address 0.0.0.0 and fd: 0


INFO: Listening on TCP port 1080
INFO: Waiting for new IPv6 SOCKSv5 connection on TCP socket with address 0::0 and fd: 3


INFO: Listening on UDP port 8080
INFO: Waiting for new IPv4 manager connection on UDP socket with address 127.0.0.1 and fd: 4

INFO: Listening on UDP port 8080
INFO: Waiting for new IPv6 manager connection on UDP socket with address ::1 and fd: 5
```

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ curl -x socks5://localhost foo.leak.com.ar
hola mundo!
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ curl foo.leak.com.ar
hola mundo!
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$
```

(el segundo es un curl normal para ver que se obtiene el mismo resultado, aunque esto también fue observado con más detalle en la sección de pruebas de performance con sha1sum)
Por otra parte, en la terminal en donde se está ejecutando el servidor se puede observar la siguiente salida mostrando un registro de acceso:

```
INFO: Connect in progress
INFO: Connection success
INFO: 2022-06-20T10:25:52Z            A      127.0.0.1:33758 52.218.242.10:80       status=0
```

12.3 Conexión con el dog client

Nuevamente ejecutamos el servidor con sus valores por defecto (aunque obviamente podrían utilizarse otros):

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./socks5d
INFO: Listening on TCP port 1080
INFO: Waiting for new IPv4 SOCKSv5 connection on TCP socket with address 0.0.0.0 and fd: 0


INFO: Listening on TCP port 1080
INFO: Waiting for new IPv6 SOCKSv5 connection on TCP socket with address 0::0 and fd: 3


INFO: Listening on UDP port 8080
INFO: Waiting for new IPv4 manager connection on UDP socket with address 127.0.0.1 and fd: 4


INFO: Listening on UDP port 8080
INFO: Waiting for new IPv6 manager connection on UDP socket with address ::1 and fd: 5
```

A continuación corremos el cliente y ejecutamos un comando de tipo GET para probar que efectivamente el servidor nos responde:

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./dog 127.0.0.1 8080
dog client >> bytes
Amount of bytes transfered: 0
dog client >>
```

Una observación a realizar es que en caso de que el cliente dog fuese ejecutado con una ip y un puerto distintos a los que espera el servidor, el cliente mostrará la terminal, pero al hacer una consulta no obtendrá ningún tipo de respuesta:

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./dog 127.0.0.1 8081
dog client >> bytes
Destination unreachable.
dog client >>
```

Una vez lograda una conexión exitosa se puede utilizar el comando "help" para obtener una tabla de comandos disponibles:

```
dog client >> help
+------------+--------------------+-------------------------------------------------------------------+
|  Command   |       Usage        |                            Description                            |
+------------+--------------------+-------------------------------------------------------------------+
| list       | list <page_number> | Returns the specified page of the list of users registered on the server |
| hist       | hist               | Returns the amount of historic connections over the server        |
| conc       | conc               | Returns the amount of concurrent connections over the server      |
| bytes      | bytes              | Returns the amount of bytes transfered over the server            |
| checksniff | checksniff         | Returns the status of the password disector over the server       |
| checkauth  | checkauth          | Returns the status of authentication over the server              |
| getpage    | getpage            | Returns the amount of users per page (max 200)                    |
| add        | add user:pass      | Command to add a user                                             |
| del        | del user           | Command to delete a user                                          |
| sniff      | sniff on/off       | Command to toggle POP3 credential sniffer over the server         |
| auth       | auth on/off        | Command to toggle authentication over the server                  |
| setpage    | setpage <page_size> | Command to set page size (between 1 and 200)                     |
+------------+--------------------+-------------------------------------------------------------------+
```

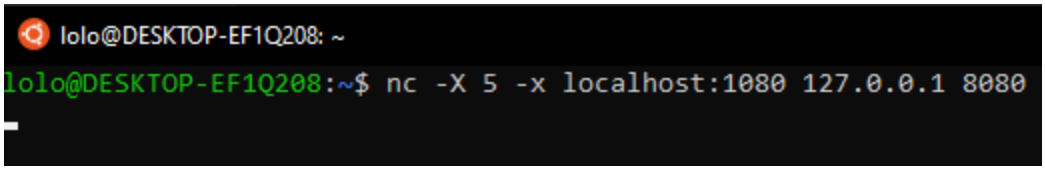
12.4 Conexiones concurrentes

En este caso se busca mostrar el uso del comando que devuelve las conexiones concurrentes sobre el servidor. Para ello se utilizó el proxy con sus valores por defecto y se conectaron 2 terminales con netcat (una escuchando en el puerto 8080 sin utilizar el proxy y la otra pasando por el proxy):

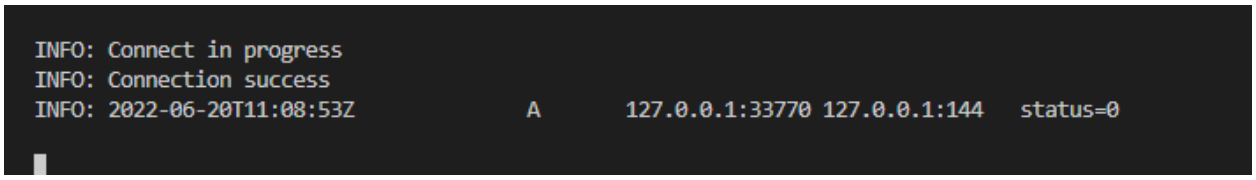
Terminal en escucha:

```
lolo@DESKTOP-EF1Q208: ~
lolo@DESKTOP-EF1Q208:~$ nc -l localhost 8080
```

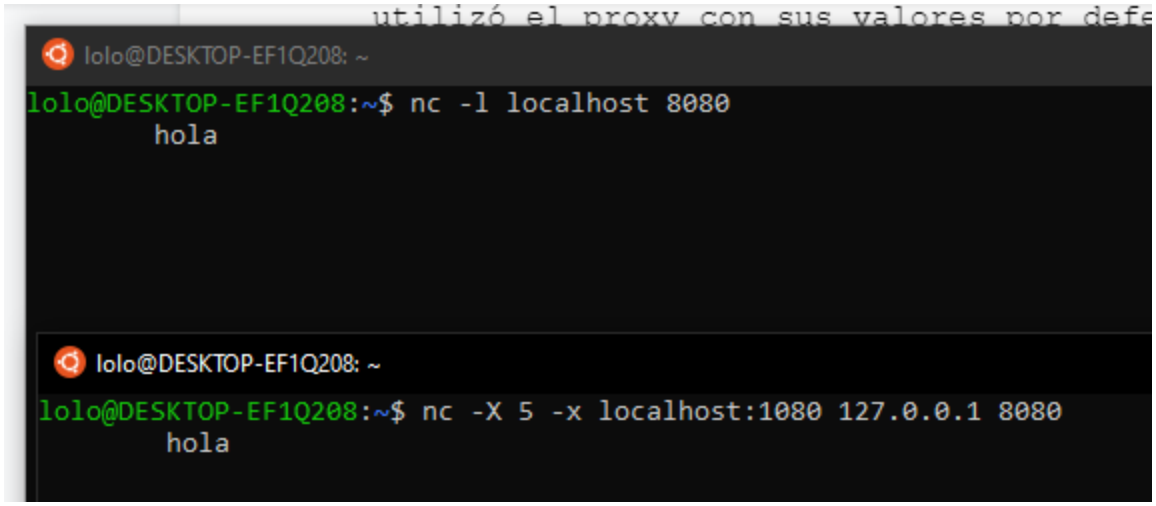
Terminal que usa el proxy:



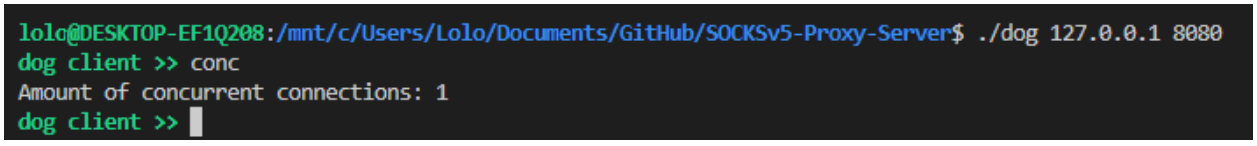
Una vez hecho esto en el servidor se muestra lo siguiente:



Y obviamente las terminales pueden comunicarse entre sí:



Ahora si desde el dog client ejecutamos el comando 'conc' obtendremos lo siguiente:



12.5 Bytes transferidos

Ejecutamos el servidor en sus valores default y hacemos un curl a foo.leak.com.ar
Luego conectamos el cliente de la manera correspondiente y ejecutamos el comando 'bytes':

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./dog 127.0.0.1 8080
dog client >> bytes
Amount of bytes transfered: 450
dog client >>
```

12.6 Manejo de usuarios

12.6.1 Creación de usuarios desde el server

Como se mencionó en la sección de 'Decisiones de implementación', cuando el servidor es ejecutado definiendo usuarios, entonces se considera que la autenticación está encendida.
Dicho esto, la manera de inicializar el servidor ya con usuarios definidos es la siguiente:

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./socks5d -u usr:pass -u usr2:pass2 -u pepe:pass3
INFO: Listening on TCP port 1080
INFO: Waiting for new IPv4 SOCKSv5 connection on TCP socket with address 0.0.0.0 and fd: 0


INFO: Listening on TCP port 1080
INFO: Waiting for new IPv6 SOCKSv5 connection on TCP socket with address 0::0 and fd: 3


INFO: Listening on UDP port 8080
INFO: Waiting for new IPv4 manager connection on UDP socket with address 127.0.0.1 and fd: 4


INFO: Listening on UDP port 8080
INFO: Waiting for new IPv6 manager connection on UDP socket with address ::1 and fd: 5
```

Ahora si desde el cliente preguntamos por la autenticación con el comando 'checkauth' obtendremos lo siguiente:

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./dog 127.0.0.1 8080
dog client >> checkauth
Authentication status: 1
dog client >>
```

Comprobando que efectivamente la autenticación está encendida.
Si se quiere realizar un request pasando por el proxy, va a ser necesario autenticarse de forma correcta:

```
lolo@DESKTOP-EF1Q208:~$ curl -x socks5://localhost foo.leak.com.ar
curl: (7) Unable to receive initial SOCKS5 response.
lolo@DESKTOP-EF1Q208:~$ curl -x socks5://wrongusr:pass@localhost foo.leak.com.ar
curl: (7) User was rejected by the SOCKS5 server (1 1).
lolo@DESKTOP-EF1Q208:~$ curl -x socks5://usr:pass@localhost foo.leak.com.ar
hola mundo!
```

12.6.2 Listado de usuarios

Para comprobar los usuarios disponibles se puede utilizar el comando 'list' dentro del dog client:

```
lolo@DESKTOP-EF1Q208:/mnt/c/Users/Lolo/Documents/GitHub/SOCKSv5-Proxy-Server$ ./dog 127.0.0.1
8080
dog client >> list 1
Users:
usr
usr2
pepe
dog client >> list 2
Users:

dog client >>
```

Una aclaración importante a realizar es que es **NECESARIO** indicar la página de usuarios que se desea obtener. Recordemos que por defecto cada página soporta hasta 200 usuarios, pero esto se puede comprobar con el comando 'getpage':

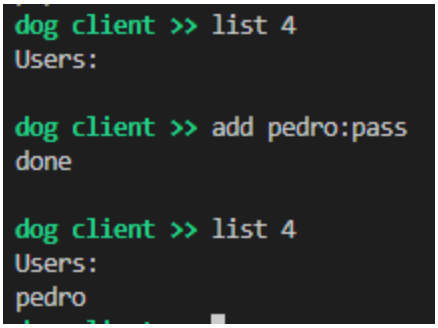
```
dog client >> getpage
Users per page: 200
dog client >>
```

También es posible cambiar el tamaño de la página:

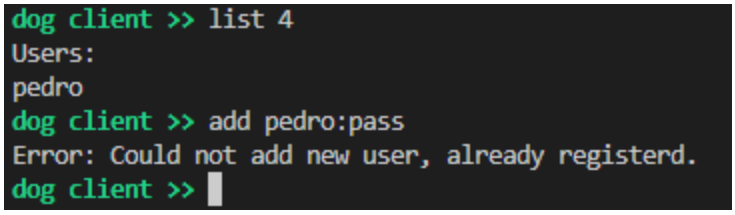
```
dog client >> setpage 1
done

dog client >> list 1
Users:
usr
dog client >> list 2
Users:
usr2
dog client >> list 3
Users:
pepe
dog client >> list 4
Users:
```

12.6.3 Agregado de usuarios desde el cliente

Es posible agregar un usuario desde el cliente con el comando 'add' seguido del usuario y la contraseña de la siguiente manera:

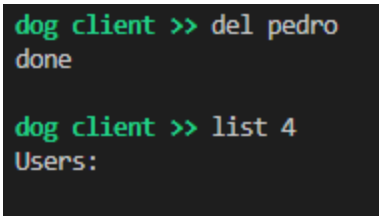


En caso de que se quiera agregar un usuario ya existente se obtiene lo siguiente:

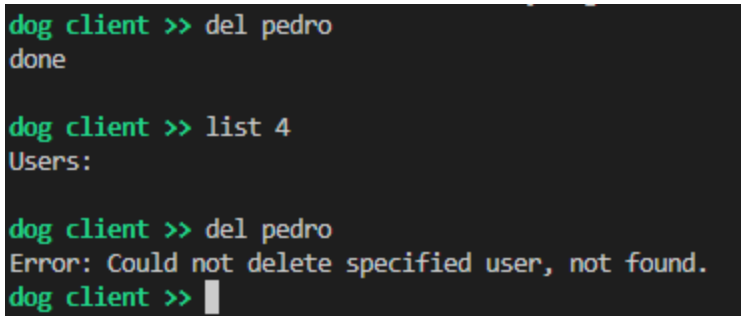


12.6.4 Borrado de usuarios

Para borrar un usuario se utiliza el comando 'del' seguido del usuario que se desea borrar:

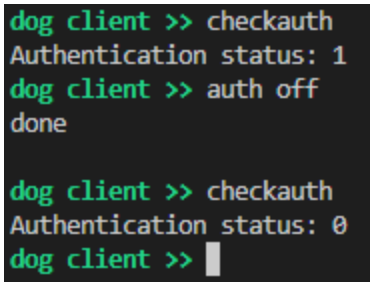


En caso de que se quiera borrar un usuario inexistente se obtiene la siguiente salida:

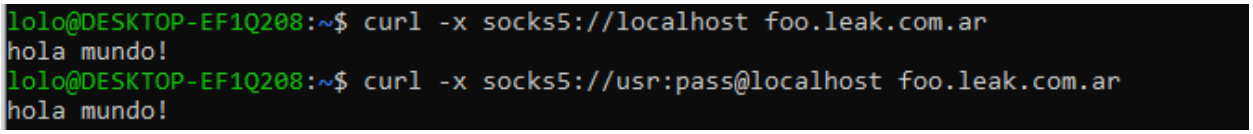


12.6.5 Apagar autenticación

Para apagar la autenticación basta con ejecutar el comando 'auth' seguido del parámetro 'off' (para encenderla es análogo):



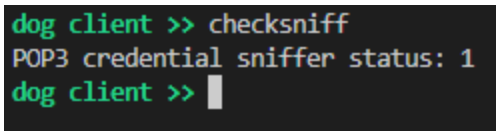
Y ahora podremos realizar pedidos sin autenticarse:



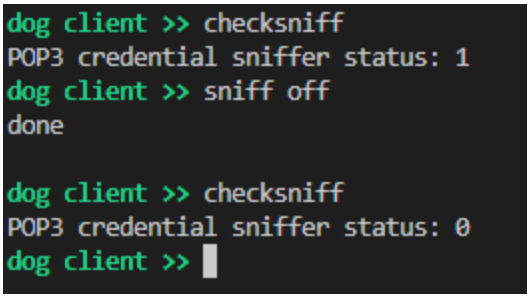
Nótese que como la autenticación está apagada, el servidor ignora si se especificó un usuario o no.

12.7 Sniffer de contraseñas sobre POP3

Para ver el estado del sniffing de contraseñas en el servidor basta con ejecutar el comando 'checksniff' el cual devuelve 1 si está encendido y 0 si no:



En caso de querer apagarlo basta con ejecutar el comando 'sniff' seguido de 'off':



Análogamente para encenderlo. Cabe destacar que por defecto el sniffer está encendido.

Ejemplo en funcionamiento:

```
ficored@DESKTOP-76MVC3O:~/winhomef/documents/SOCKSv5-Proxy-Server$ nc -X 5 -x localhost:1080 127.0.0.1 110
+OK Dovecot (Ubuntu) ready.
USER ficored
+OK
PASS testing
+OK Logged in.
```

```
ficored@DESKTOP-76MVC3O:~/winhomef/documents/SOCKSv5-Proxy-Server$ ./socks5d
INFO: Listening on TCP port 1080
INFO: Waiting for new IPv4 SOCKSv5 connection on TCP socket with address 0.0.0.0 and fd: 0


INFO: Listening on TCP port 1080
INFO: Waiting for new IPv6 SOCKSv5 connection on TCP socket with address 0::0 and fd: 3


INFO: Listening on UDP port 8080
INFO: Waiting for new IPv4 manager connection on UDP socket with address 127.0.0.1 and fd: 4


INFO: Listening on UDP port 8080
INFO: Waiting for new IPv6 manager connection on UDP socket with address ::1 and fd: 5


INFO: new connection, curr conn: 1
INFO: Connect in progress
INFO: Connection success
INFO: 2022-06-20T13:23:57Z      A       127.0.0.1:60588 127.0.0.1:110   status=0

INFO: 2022-06-20T13:24:03Z              P       POP3    127.0.0.1:110   sniffed credentials: ficored:testing    status=0
```

12.8 Conexiones históricas

Iniciando el servidor en su configuración por defecto, se utiliza el siguiente script para crear 10 conexiones:

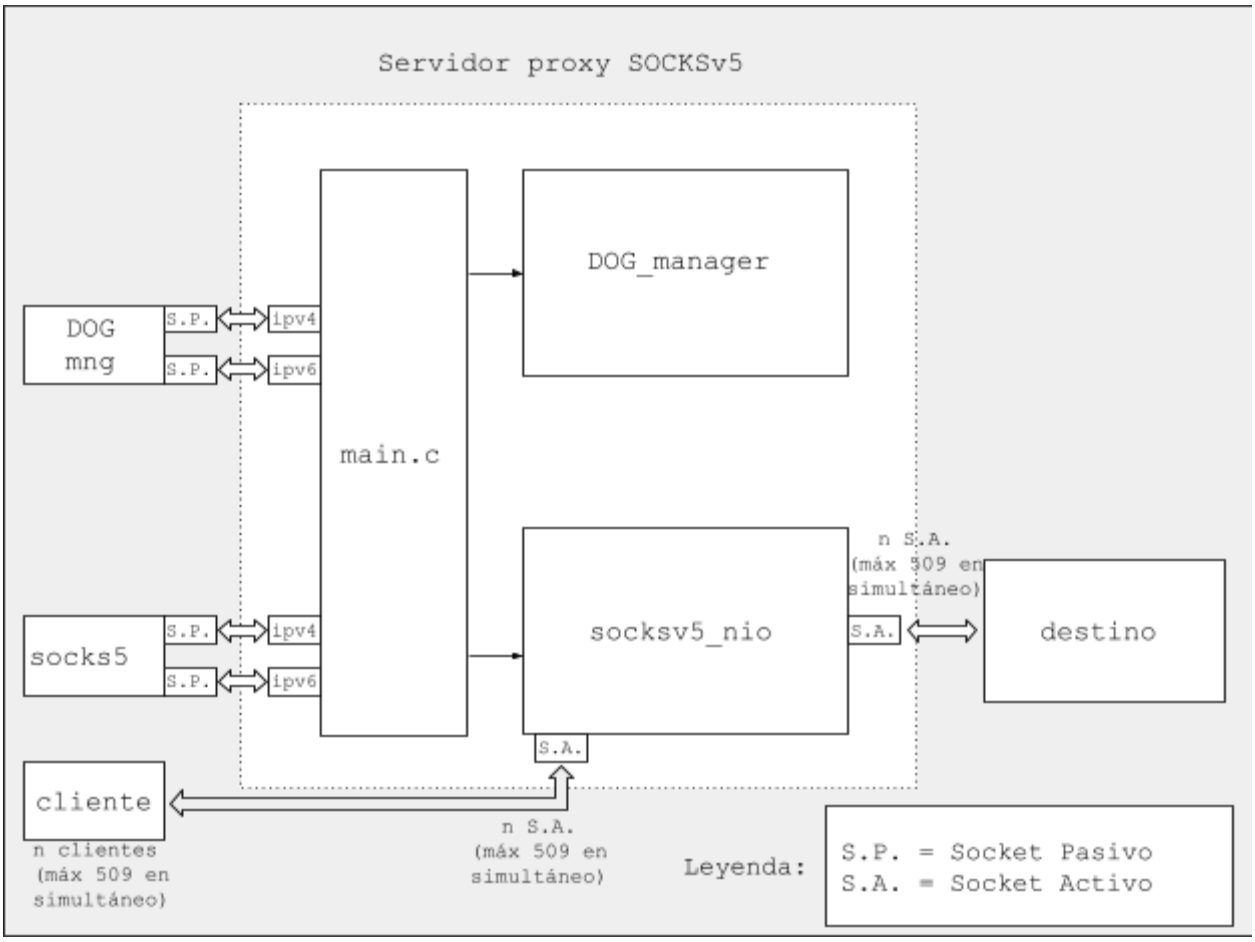
```
ficored@DESKTOP-76MVC3O:~$ for i in {1..10}; do curl --limit-rate 1k -x socks5h://0.0.0.0 https://old-releases.ubuntu.com/releases/4.
10/warty-release-install-amd64.list & done
```

Luego en el administrador de servidor:

```
dog client >> hist
Amount of historic connections: 10
```

13. Arquitectura de la aplicación

13.1 Diseño general



13.2 Diagrama de estados de una conexión exitosa