



Trabajo Práctico Especial: Servidor

Proxy Socks5

72.07 - Protocolos de Comunicación

Primer Cuatrimestre 2025

Grupo G-04

Ana Negre - 63074

Camila Lee - 63382

Juan Oliva - 64105

Matías Leporini - 62872

Índice

1. Protocolos y aplicaciones.....	2
1.1. Aplicaciones.....	2
1.1.1. Servidor SOCKS5 (socks5d).....	2
1.1.2. Servidor de Management (CalSetting).....	2
1.1.3. Cliente.....	2
1.2. Protocolos.....	3
1.2.1. Servidor - Decisiones tomadas.....	3
1.2.2. Monitoreo - CalSetting.....	5
2. Problemas encontrados.....	10
3. Limitaciones.....	12
4. Posibles extensiones.....	12
5. Ejemplos de prueba.....	13
6. Conclusiones.....	18
7. Guía de instalación.....	18
7.1. Compilación.....	18
7.2.1. Servidor.....	19
7.2.2. Cliente.....	19
8. Instrucciones para la configuración.....	20
9. Ejemplos de configuración y monitoreo.....	20
10. Documentos de diseño.....	27
10.1. Servidor.....	27
10.2. Cliente.....	29

1. Protocolos y aplicaciones

1.1. Aplicaciones

1.1.1. Servidor SOCKS5 (socks5d)

Como foco principal del trabajo se buscó realizar la correcta implementación del protocolo SOCKS5v, basando la misma en las especificaciones del [RFC 1928](#) y [RFC 1929](#). Se utilizaron las librerías provistas por la cátedra, si bien con algunas modificaciones que se detallarán más adelante.

El servidor SOCKS provee la funcionalidad de resolución DNS, la cual a su vez tiene soporte *dual-stack* tanto para clientes como para destinos. Adicionalmente, se decidió implementar un *timeout* configurable para evitar conexiones “zombie” que pudieran afectar el comportamiento del servidor.

Para la utilización del proxy se implementó la autenticación usuario/contraseña, con soporte de hasta diez usuarios totales configurables vía línea de comandos al iniciar el servidor (pudiendo agregarse o eliminarse mediante el protocolo diseñado). Además, se permitió el uso sin autenticación del proxy para casos de uso sin restricciones. Si bien el servidor distingue entre administradores y usuarios, ambos grupos pueden utilizar el proxy de manera autenticada.

Al iniciar el servidor, es posible configurar los parámetros a utilizar; se pueden agregar usuarios y administradores y establecer los puertos y direcciones de ambos servicios (Socks5v y CalSetting). Se puede consultar una descripción detallada de los argumentos aceptados en la guía de instalación.

1.1.2. Servidor de Management (CalSetting)

Para el procesamiento de las solicitudes y respuestas mediante el protocolo diseñado, se implementó un mecanismo de recolección de métricas volátiles del uso del proxy, las cuales luego son solicitadas por el protocolo de monitoreo y entregadas al cliente. Se cuentan la cantidad de bytes recibidos y transmitidos, la cantidad de conexiones históricas y actuales, la cantidad máxima de conexiones concurrentes históricas y la cantidad de errores (junto a su clasificación por tipo).

El servidor de management utiliza CalSetting para atender consultas referidas a las métricas recolectadas y a los logs (también volátiles) de acceso al servidor SOCKS. Los logs guardados refieren a los accesos remotos tanto de usuarios como de clientes no autenticados denominados “anonymous”. De los registros de acceso, se almacena el nombre de usuario, dirección IP de origen, dirección IP de destino, puertos de origen y destino, y código de estado (referido al estado en que terminó la conexión).

1.1.3. Cliente

En la implementación de la aplicación cliente, los usuarios pueden optar por dos formas de uso: consola o interfaz gráfica, siendo esta última la UI de facto. Ambas cuentan con total paridad en la funcionalidad ofrecida, permitiendo:

- Ver la lista de los logs del servidor

- Ver la lista de usuarios
- Ver métricas del servidor
- Cambiar el tamaño del *buffer*
- Cambiar el tiempo de *timeout*
- Ver las configuraciones actuales
- Agregar usuarios
- Remover usuarios

Estas funciones son accedidas mediante un proceso de autenticación, en el cual únicamente se permite el ingreso de usuarios antes definidos como administradores. Esto se decidió en base a que lo ofrecido implica información y procesos que solo alguien de confianza debería poder acceder; sería poco conveniente que un usuario común pueda ver la lista completa de sus pares, o removerlos.

Una característica notable es la posibilidad de reconectarse ante cierres del servidor. De entrar a alguna de las características ofrecidas con el servidor caído o cerrado y elegir la opción de reconexión, al usuario se le pedirá que se reautentique. Una vez hecho esto exitosamente, será redirigido a la ventana en la que se encontraba justo antes del fallo.

1.2. Protocolos

1.2.1. Servidor - Decisiones tomadas

A la hora de implementar la multiplexación I/O, se decidió modificar la librería selector.h provista por la cátedra para utilizar la system call [epoll\(\)](#) en lugar de *pselect()*. Esta decisión se tomó puesto a que se consideró *epoll()* más adecuada para manejar una cantidad elevada de *file descriptors*, además de ser más eficiente y estar pensada para entornos modernos. Las ventajas de *epoll* por sobre otros mecanismos de *polling* no solo fue vista en clase, sino que también pudo evidenciarse en la mejora del funcionamiento una vez implementado este cambio.

Para los *buffers* de escritura y lectura del *proxy*, se decidió hacer uso de la librería provista por la cátedra. Como la misma soporta escritura y lectura sobre el mismo *buffer*, se evaluó la posibilidad de utilizar uno solo para la conexión con el cliente y con el remoto respectivamente, con la finalidad de no saturar los recursos del sistema en términos de asignación de memoria. En la preentrega, sin embargo, se comentó la potencial pérdida de *performance* en caso de tráfico asimétrico, por lo que se decidió priorizar otras mejoras en detrimento de la modificación de los *buffers*. No obstante, no se descarta su eventual implementación como mejora del proyecto.

Para la resolución de dominios se decidió crear un *thread* separado por cada resolución DNS (que fue la única instancia del estilo aceptada por la cátedra). Esta decisión se tomó para mantener la responsividad del servidor principal durante resoluciones lentas y evitar bloquear el *loop* principal con llamadas a [getaddrinfo](#).

Para evitar *race conditions* en situaciones de múltiples resoluciones de DNS simultáneas, por recomendación de la cátedra se optó por reutilizar la infraestructura existente del selector en lugar de implementar un sistema de sincronización propio. Así,

se extendió la función `selector_notify_block()` agregando un parámetro adicional para poder transferir el resultado (`struct addrinfo*`) directamente a través del mecanismo de `blocking_job` ya provisto. Esta decisión permite aprovechar el `mutex` único del selector en lugar de crear uno por cada conexión y detectar automáticamente las sesiones que se cierran durante la resolución.

Por el otro lado, para el manejo de notificaciones de los *threads* de resolución DNS al hilo principal, se decidió modificar nuevamente la implementación del selector, pasando a utilizar `eventfd`. Durante el desarrollo se observó que la señal de `p_thread_kill()` no estaba siendo procesada correctamente por `epoll_wait()`, con lo cual las notificaciones de resolución no estaban siendo atendidas. Si bien lo más probable es que esta notificación no se estuviera atendiendo por un error de implementación humano, igualmente se consideró más conveniente utilizar `eventfd` para el manejo de notificaciones. Al estar utilizando `epoll`, además, el costo del *file descriptor* utilizado por `eventfd` no resultó relevante. Cabe destacar que `eventfd` resulta más acorde al paradigma del selector implementado, y evita las tendencias *error-prone* del manejo de señales.

Otra decisión importante tomada durante el desarrollo es la incorporación del `timeout`. El objetivo de esta decisión fue incorporar un mecanismo para liberar recursos luego de un cierto tiempo de inactividad, evitando de esta manera conexiones “zombies”. Adicionalmente, se proporcionó la posibilidad de configurar este `timeout` desde el protocolo de monitoreo, con el fin de otorgar mayor flexibilidad de administración. Para la utilización del `timeout` se implementaron los parámetros correspondientes en cada sesión para que, al iniciar la conexión, se calcule el próximo `timeout` de dicha sesión. Este es restablecido cada vez que se lee de o escribe al cliente. El selector, por el otro lado, calcula el `timeout` más próximo para utilizarlo como el `-valga la redundancia-timeout` de `epoll_wait()`. De esta manera, incluso si ninguna otra sesión ha registrado eventos para ser atendidos, el selector se despertará en el momento indicado para limpiar las sesiones inactivas. En la figura 1.1 se detalla el flujo de funcionamiento del `timeout`.

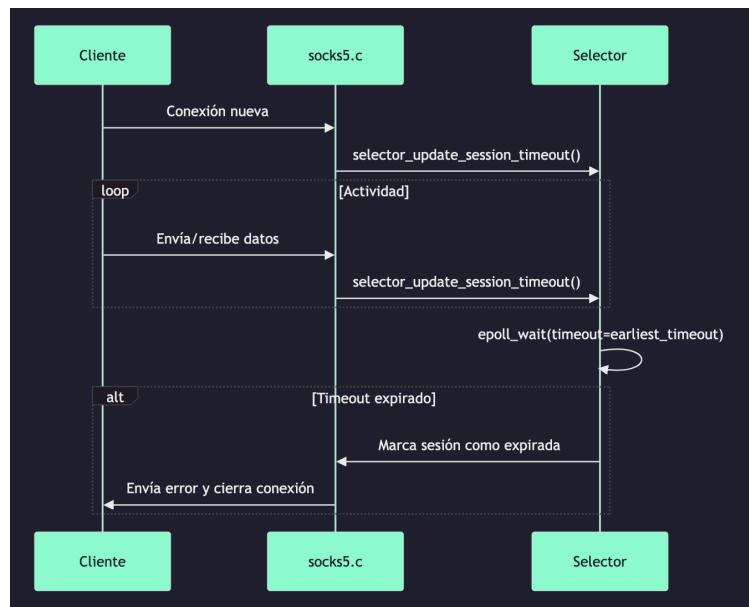


Figura 1.1 - Flujo del timeout.

La implementación del *timeout*, sin embargo, tuvo un impacto negativo sobre la *performance* del servidor. En particular, al incrementar la cantidad de sesiones activas, puesto a que (en la implementación actual) para limpiar recursos, luego de procesar los nuevos eventos, el selector debe recorrer todas las sesiones para verificar que ningún *timeout* haya expirado. Este recorrido tiene complejidad $O(N)$, por lo que eventualmente se vuelve un cuello de botella para el procesamiento de clientes. Se decidió proceder con su implementación, sin embargo, puesto a que la concurrencia exigida era relativamente baja, y puesto a que resultó un manejo defensivo más robusto ante eventuales conexiones maliciosas.

Con respecto a la recolección de registros de acceso, se evaluaron varias posibilidades respecto al almacenamiento de las mismas. En primera instancia, buscando evitar el agotamiento de recursos de memoria, se evaluó la posibilidad de guardar los registros en un archivo de disco. Esto, a su vez, permitiría un almacenamiento no volátil, lo cual puede resultar útil para llevar registros más extensos sobre el servidor. El costo elevado de leer/escribir a disco, sin embargo, hizo que se optara por guardar los registros en memoria. Para la implementación, entonces, se decidió guardar un *array* estático de `MAX_RECENT_LOGS`, sobrescribiendo los más viejos en caso de llegar al máximo permitido.

1.2.2. Monitoreo - CalSetting

Para que los administradores puedan monitorear y configurar el servidor Socks5, se diseñó el protocolo “CalSetting”, operando el mismo sobre TCP. Para la estructura de los paquetes se decidió utilizar un diseño similar a los paquetes de SOCKS5v, al resultar relativamente simples de manipular, priorizando enviar información en tamaños fijos siempre que esto sea posible.

En cada una de las tablas a continuación, los números decimales refieren a la longitud (en octetos) del campo. Por otro lado, para especificar valores se da uso de la sintaxis X'hh.

Handshake: El cliente y el servidor realizan un *handshake* antes de pasar a cualquier otra operación. Para esto, el cliente envía el siguiente mensaje:

version	ulen	pwdlen	username	password
1	1	1	variable	variable

donde -en este y todo el resto de los casos- el campo ‘version’ debe estar fijado en X'01 para la versión actual del protocolo. Luego, el servidor debe responder con:

version	response_code
1	1

donde los posibles códigos de respuesta son:

- X'01: Éxito al ingresar como cliente
- X'02: Éxito al ingresar como administrador
- X'03: Error de autenticación
- X'04: Error general del servidor
- X'05: Versión incorrecta

Debido a que solo se permite que los administradores ingresen al cliente, si el servidor no encuentra una cuenta de *admin* asociada al par usuario/contraseña, envía un error de autenticación (X'06: Petición no permitida) y cierra la conexión.

Autenticación: Una vez realizado el *handshake*, es posible comenzar a mandar peticiones, ya sea para monitorear el estado del servidor o para realizar cambios en la configuración. En cualquiera de los casos, el servidor devuelve un código de respuesta entre los listados a continuación:

- X'00: Éxito
- X'04: Error general del servidor
- X'05: Versión incorrecta
- X'06: Petición no permitida
- X'0B: Petición incorrecta o mal formada

Vista de logs: El administrador puede solicitar ver los logs del servidor con la petición:

version	CMD	N	offset

1	1	1	1
---	---	---	---

donde ‘CMD’ debe setearse en X’00. ‘N’ refiere al número de logs solicitados, y ‘offset’ refiere al desplazamiento sobre el cual comenzar a recolectar los logs, permitiendo visualizar los logs de manera paginada. El servidor debe responder con:

version	response_code	CMD	count	log_entry1	log_entry2	...	log_entryN
1	1	1	1	586	586		586

donde ‘CMD’ debe setearse en X’00; de aquí en adelante, las respuestas a pedidos siempre repiten en número de comando. ‘count’ se refiere a la cantidad de log_entries en la respuesta, y cada log_entry tiene el formato:

date	ulen	username	register_type	origin_ip	origin_port
21	1	255	1	46	2

destination_ATYP	destination_address	destination_port	status_code
1	256	2	1

donde ‘date’ está en formato ISO-8601, register_type es X’41 (valor ASCII del carácter ‘A’), status_code refiere al código de estado de Socks5 y ‘destination_ATYP’ toma un valor según el tipo de dirección a la que el cliente se intentó conectar:

- X'01: IPv4
- X'03: Dominio
- X'04: IPv6

Vista de usuarios: El administrador puede solicitar ver la lista de usuarios del servidor con la petición:

version	CMD	N	offset
1	1	1	1

donde ‘CMD’ debe setearse en X’01. ‘N’ y ‘offset’ tienen el mismo propósito que para los logs, permitiendo que el administrador pueda visualizar los usuarios de forma paginada. El servidor debe responder con:

version	response_code	CMD	nusers
---------	---------------	-----	--------

1	1	1	1
---	---	---	---

ulen1	utype1	username1	...	ulenN	utypeN	usernameN
1	1	255		1	1	255

donde ‘nuserst’ se refiere a la cantidad de usuarios en la respuesta y ‘utype’ es X’00 si el usuario es común y X’01 si es administrador.

Vista de métricas: El administrador puede solicitar ver las métricas del servidor con la petición:

version	CMD	RSV
1	1	2

donde ‘CMD’ debe setearse en X’02 y los bytes ‘RSV’ deben estar seteados en X’00. El servidor debe responder con:

version	server_state	concurrent_connections	total_connections	max_concurrent	bytes_in
1	1	4	8	4	8

bytes_out	total_bytes	total_errors	uptime_seconds	nnetwork_errors	protocol_errors
8	8	4	4	4	4

auth_errors	system_errors	timeout_errors	memory_errors	other_errors
4	4	4	4	4

Cambiar el tamaño del buffer: El administrador puede solicitar cambiar el tamaño del buffer del servidor con la petición:

version	CMD	buff_size_kb	RSV
1	1	1	1

donde ‘CMD’ debe setearse en X’03 y el byte ‘RSV’, como en todos los casos que siguen, debe estar seteado en X’00. El servidor debe responder con:

<i>version</i>	<i>response_code</i>	CMD
1	1	1

Cambiar timeout: El administrador puede solicitar cambiar el tiempo para *timeout* del servidor con la petición:

<i>version</i>	CMD	<i>timeout</i>	RSV
1	1	1	1

donde ‘CMD’ debe setearse en X’04. El servidor debe responder con:

<i>version</i>	<i>response_code</i>	CMD
1	1	1

Vista de configuración: El administrador puede solicitar ver la configuración actual del servidor con la petición:

<i>version</i>	CMD	RSV
1	1	1

donde ‘CMD’ debe setearse en X’05. El servidor debe responder con:

<i>version</i>	<i>response_code</i>	CMD	RSV	<i>buff_size_kb</i>	<i>timeout_seconds</i>
1	1	1	1	1	1

Agregar usuario: El administrador puede solicitar agregar un usuario con la petición:

<i>version</i>	CMD	<i>Ulen</i>	<i>PWDlen</i>	<i>username</i>	<i>password</i>
1	1	1	1	1	1

donde ‘CMD’ debe setearse en X’06. El servidor debe responder con:

version	response_code	CMD
1	1	1

donde, más allá de los códigos de error generales, *response_code* puede setearse en:

- X'09: El usuario ya existe
- X'0A: Límite de usuarios (10) alcanzado

Eliminar usuario: El administrador puede solicitar borrar un usuario con la petición:

version	CMD	ulen	RSV	username
1	1	1	1	1

donde ‘CMD’ debe setearse en X’08 (originalmente, X’07 era agregar un administrador, pero la funcionalidad se descartó). El servidor debe responder con:

version	response_code	CMD
1	1	1

donde adicionalmente *response_code* puede setearse en:

- X'07: Usuario no encontrado

2. Problemas encontrados

A lo largo del desarrollo se encontraron una gran variedad de problemas en relación a la implementación del servidor SOCKS, la mayoría referidos a una mala utilización de los recursos y las librerías provistas por la cátedra. En particular, la sincronización de liberación de *file descriptors* resultó un desafío, puesto a que se notó que los mismos no estaban siendo liberados en la misma medida que los clientes y remotos cerraban la conexión. Para esto se mejoró el evento de cierre, impidiendo “double frees” sobre los *file descriptors* y manejando la liberación de recursos correctamente.

Adicionalmente, como fue mencionado en la sección [1.2.1. Servidor - Decisiones tomadas](#), se encontraron *race conditions* en el manejo de resoluciones DNS y la liberación de recursos. Este problema puede evidenciarse en los logs de la figura 2.1. Para solucionarlo, se emplearon las técnicas previamente mencionadas.

```

[pid 82545] epoll_ctl(4, EPOLL_CTL_MOD, 5, {events=5, data={u32=1052, u64=108508053766776}}) = 0
DEBUG: src/server/socks5.c:1029, [DNS_THREAD] Starting DNS resolution for localhost:8082
DEBUG: src/server/socks5.c:1067, [DNS_THREAD] DNS resolution succeeded for localhost
INFO: src/server/socks5.c:1634, [DNS_RESOLVE] Resolved addresses for domain 'localhost':
INFO: src/server/socks5.c:1649, [DNS_RESOLVE] [1] IPv4: 127.0.0.1:8082
INFO: src/server/socks5.c:1649, [DNS_RESOLVE] [2] IPv4: 127.0.0.1:8082
INFO: src/server/socks5.c:1653, [DNS_RESOLVE] Total addresses resolved: 2
INFO: src/server/socks5.c:1075, [DNS_RESOLUTION_THREAD] Incremented timeout, notifying selector block...
[pid 8250] *** exited with 0 ***
INFO: src/server/libs(selector.c:550, [SELECTOR_SELECT] Idle connection timeout for fd=5
epoll_ctl(4, EPOLL_CTL_MOD, 5, {events=EPOLLOUT, data={u32=632, u64=108508053766776}}) = 0
DEBUG: src/server/socks5.c:314, [HANDLE_BLOCK] Ignoring block event - not resolving domain
DEBUG: src/server/socks5.c:1610, [HANDLE_ERROR] Handling error state for fd=5
DEBUG: src/server/socks5.c:1600, [SEND SOCKS_ERROR] Prepared error response with code: 0x06
epoll_ctl(4, EPOLL_CTL_MOD, 5, {events=EPOLLOUT, data={u32=632, u64=108508053766776}}) = 0
DEBUG: src/server/socks5.c:1618, [HANDLE_ERROR] Switching to STATE_ERROR_WRITE to send error response
INFO: src/server/socks5.c:527, [WRITE_TO_CLIENT] Sent 10/10 bytes to client.
DEBUG: src/server/socks5.c:539, [WRITE_TO_CLIENT] All data sent. Shutting down socket.
DEBUG: src/server/socks5.c:259, [SOCKS5_HANDLE_CLOSE] *** CLOSE HANDLER CALLED *** for fd=5
DEBUG: src/server/socks5.c:278, [SOCKS5_HANDLE_CLOSE] Client fd=5 closed
DEBUG: src/server/socks5.c:300, [SOCKS5_HANDLE_CLOSE] Both fds closed, cleaning up session
DEBUG: src/server/socks5.c:259, [SOCKS5_HANDLE_CLOSE] *** CLOSE HANDLER CALLED *** for fd=5
DEBUG: src/server/socks5.c:269, [SOCKS5_HANDLE_CLOSE] Session already being cleaned up, skipping
epoll_ctl(4, EPOLL_CTL_DEL, 5, NULL) = 0
DEBUG: src/server/socks5.c:1670, [CLEANUP] Starting session cleanup for client_fd=-1, remote_fd=-1
DEBUG: src/server/socks5.c:1680, [CLEANUP] Freeing domain name: localhost
DEBUG: src/server/socks5.c:1686, [CLEANUP] Freeing dst_addresses (atyp=DOMAIN)
DEBUG: src/server/socks5.c:1738, [CLEANUP] Session cleanup completed
DEBUG: src/server/socks5.c:306, [SOCKS5_HANDLE_CLOSE] Close handler complete for fd=5

```

Figura 2.1. Señal de thread DNS no capturada.

En la interfaz gráfica de la aplicación cliente, tomó lugar un vaivén respecto a qué librería utilizar. Originalmente se iba a dar uso de *Dialog*, pero en cierto punto del desarrollo se decidió migrar a *Whiptail*, ya que ofrecía más simpleza a la hora de crear las interfaces y se consideró más estética. Sin embargo, se encontró un problema con la nueva librería: desde alrededor de 2019, la función de scroll dejó de funcionar, por lo que mostrar ventanas extensas resultaba una imposibilidad. Así, se volvió a *Dialog*.

Otra problemática en la aplicación cliente fue la compatibilidad con *Pampero*; *Dialog* no está entre las librerías allí instaladas. De esta forma, se implementó tanto una interfaz gráfica como una algo más rudimentaria en la consola, pudiéndose alternar entre las opciones con la flag `--console` (siendo *Dialog* la opción por defecto). Para solucionar esta dicotomía, se creó una librería para cada tipo de UI (`ui_dialog` y `ui_console`) y un adaptador (`ui_adapter`). Una función de este último toma la siguiente forma:

```

void ui_show_message(const char *title, const char *message) {
    if(ui_mode){
        ui_console_show_message(title, message);
    } else {
        ui_dialog_show_message(title, message);
    }
}

```

Así, el cliente únicamente llama a las funciones del adaptador, inicializándolo con un valor booleano `ui_mode` según la elección del usuario. Mediante esta implementación -una suerte de inyección de dependencias- los nombres de las funciones de UI llamadas a lo largo del proyecto son independientes de la interfaz

elegida, quedando encapsulado el comportamiento dual y siendo de relativa sencillez añadir nuevos modos de visualizar la información.

3. Limitaciones

En términos de escalabilidad, una de las limitaciones más evidentes es la ineficiencia de las resoluciones DNS, puesto que cada conexión debe crear un *thread* separado, lo que podría resultar ineficiente o incluso insostenible ante muchas resoluciones simultáneas. Se podría utilizar un *pool* de *threads* para reutilizar un número fijo de hilos y así limitar el consumo de recursos, o bien *cachear* las resoluciones DNS frecuentes para evitar consultas repetidas y reducir la latencia, mejorando el rendimiento general del servidor.

Por otro lado, como ya fue mencionado, la introducción de *timeouts* a las conexiones produce un *overhead* adicional en el procesamiento de eventos, lo cual se ve exacerbado al incrementar la cantidad de conexiones concurrentes. Para mejorar esta ineficiencia se investigó la posibilidad de implementar un *Wheel Timer*¹ que utilice una estructura similar a una tabla de *hash* circular para manejar los *timeouts* expirados, reduciendo la complejidad del manejo de *timeouts* a $O(1)$ en contraposición a la búsqueda lineal implementada actualmente.

Adicionalmente, se observa que -a causa de la implementación de los usuarios- los pares usuario/contraseña son fáciles de capturar, puesto a que se transportan en texto plano mediante el protocolo *CalSetting*.

4. Posibles extensiones

En primera instancia, se observan como posibles extensiones las soluciones propuestas en la sección anterior para las limitaciones más urgentes del proyecto. Estas implementaciones deberían priorizarse en casos de uso donde se espere una gran concurrencia.

Respecto a la información volátil del sistema, se considera idóneo implementar mecanismos para persistir la información del servidor a lo largo del tiempo, incluso al reiniciar el mismo. En primer lugar, se podría implementar la persistencia de usuarios y configuraciones de *buffer* y *timeouts*, escribiendo a un archivo específico para tal fin (*socks5.conf* por ejemplo). Los registros de acceso (y las métricas), por otro lado, también podrían ser guardados en un archivo, como se pensó originalmente. Podrían también utilizarse otros *threads* para guardar periódicamente los registros del *array* actual en archivos de disco, o incluso podría utilizarse nuevos procesos creados mediante *fork* si se espera que los *logs* en disco solo se actualicen periódicamente. Aunque esto último introduciría nuevos problemas de sincronización, resultaría útil para lograr un monitoreo más extensivo.

También sobre los registros, una extensión relativamente sencilla es la posibilidad de permitir que los usuarios no administradores puedan acceder al cliente

¹ [Hashed Wheel Timers - Frank Rosner \(DEV\)](#)

de métricas, o incluso a otro cliente especializado, según el cual tengan acceso a información relevante (registros de acceso propios, métricas y estado del servidor), sin acceder a información sensible del servidor.

Para extender la funcionalidad del proxy, en caso de necesitarse, una posible extensión es limitar el acceso a ciertas IPs, concepto siempre útil para un administrador de red.

En último lugar, en cuanto a lo que abarca el protocolo SOCKS5 y los comandos aceptados, solo se implementó el comando CONNECT (X'01) pero no se implementaron los comandos BIND (X'02) ni UDP ASSOCIATE (X'03) debido a que el alcance del proyecto no lo requería. La ausencia de estos comandos no impacta en la gran mayoría de usos típicos (navegación web, HTTPS y aplicaciones TCP estándar), pero dada su complejidad podría resultar interesante su incorporación para lograr una plena implementación del protocolo SOCKS5v.

5. Ejemplos de prueba

En primer lugar, se buscó testear la integridad de los datos transmitidos a través del server. Se utilizó un script (`test_data_integrity_progressive.sh`) que, a través de un servidor local, sirviera archivos creados con `/dev/urandom` de tamaños progresivamente mayores, comparando luego los hash sha256 de cada uno para verificar su contenido. A continuación pueden verse tanto el resultado de una ejecución de este test como un gráfico que promedia los datos de 3 ejecuciones. Es posible observar la línea de tendencia lineal del tiempo de descarga, lo cual era esperado para el test ejecutado.

```

root@526024388fd8:~# ./script/tests/test_data_integrity.sh
==> SOCKS5 Progressive Size Testing ==
Testing different file sizes to identify reliability patterns...

==> Testing Small (1MB) ==
Creating 1MB file...
Downloading through SOCKS5 proxy...
#####
Speed: 58338488 bytes/sec
Time: 0.017974s
Size: 1048576 bytes
✓ SUCCESS: Complete transfer (1048576 bytes) in 0s
✓ INTEGRITY: Hashes match - data integrity verified
Result: PASSED

==> Testing Medium-Small (5MB) ==
Creating 5MB file...
Downloading through SOCKS5 proxy...
#####
Speed: 103743395 bytes/sec
Time: 0.050537s
Size: 5242880 bytes
✓ SUCCESS: Complete transfer (5242880 bytes) in 0s
✓ INTEGRITY: Hashes match - data integrity verified
Result: PASSED

==> Testing Medium (10MB) ==
Creating 10MB file...
Downloading through SOCKS5 proxy...
#####
Speed: 151322769 bytes/sec
Time: 0.069294s
Size: 10485760 bytes
✓ SUCCESS: Complete transfer (10485760 bytes) in 0s
✓ INTEGRITY: Hashes match - data integrity verified
Result: PASSED

==> Testing Medium-Large (25MB) ==
Creating 25MB file...
Downloading through SOCKS5 proxy...
#####
Speed: 139366390 bytes/sec
Time: 0.188097s
Size: 26214400 bytes
✓ SUCCESS: Complete transfer (26214400 bytes) in 0s
✓ INTEGRITY: Hashes match - data integrity verified
Result: PASSED

==> Testing Large (50MB) ==
Creating 50MB file...
Downloading through SOCKS5 proxy...
#####
Speed: 129989834 bytes/sec
Time: 0.403330s
Size: 52428800 bytes
✓ SUCCESS: Complete transfer (52428800 bytes) in 0s
✓ INTEGRITY: Hashes match - data integrity verified
Result: PASSED

==> Testing Very Large (75MB) ==
Creating 75MB file...
Downloading through SOCKS5 proxy...
#####
Speed: 120962926 bytes/sec
Time: 0.650143s
Size: 78643200 bytes
✓ SUCCESS: Complete transfer (78643200 bytes) in 1s
✓ INTEGRITY: Hashes match - data integrity verified
Result: PASSED

==> Testing Extra Large (100MB) ==
Creating 100MB file...
Downloading through SOCKS5 proxy...
#####
Speed: 182574417 bytes/sec
Time: 0.574328s
Size: 104857600 bytes
✓ SUCCESS: Complete transfer (104857600 bytes) in 0s
✓ INTEGRITY: Hashes match - data integrity verified
Result: PASSED

==> Final Summary ==
Size progression results:
1MB: PASS
5MB: PASS
10MB: PASS
25MB: PASS
50MB: PASS
75MB: PASS
100MB: PASS

==> Analysis ==
✓ All tests passed - proxy appears to be working correctly

Cleaning up test files...

```

Figuras 5.1 y 5.2 - Resultados de test_data_integrity_progressive.sh.

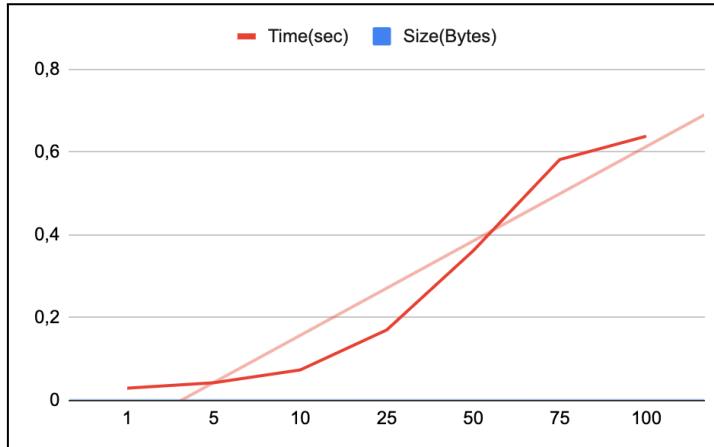


Figura 5.3 - Gráfico de tiempo en función del tamaño de archivo.

Luego se buscó comparar los tiempos de descarga de archivos grandes, en comparación con los tiempos de ejecución sin utilizar el proxy. Para esto se utilizó [nginx](#) para servir un archivo de 1GB, y se compararon los tiempos de realizar una petición con la herramienta *curl* con el proxy y sin. Se encontró que, al utilizar el *buffer default* de 4KB, se tardaba con el proxy más de 7 segundos (figura 5.4). Este rendimiento se consideró inaceptable, por lo que se decidió subir el tamaño del *buffer default* a 32KB. Así, se observó una mejora sustancial de la *performance* (figuras 5.5 y 5.6), y no se encontraron problemas de memoria al utilizar el proxy con el nuevo tamaño de *buffer*, por lo que se decidió mantenerlo. Adicionalmente, al poder cambiarse en tiempo de ejecución, esto aporta flexibilidad al administrador para poder modificarlo acorde a las

necesidades de los usuarios. El rendimiento sin la utilización del proxy puede verse en la figura 5.7.

```
anegre@anegre-vm:~$ time curl --socks5-hostname localhost:1080 http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload Total Spent   Left  Speed
100 1024M 100 1024M    0     0  139M      0  0:00:07  0:00:07  --- 139M

real    0m7,374s
user    0m0,323s
sys     0m0,986s
anegre@anegre-vm:~$ time curl --socks5-hostname localhost:1080 http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload Total Spent   Left  Speed
100 1024M 100 1024M    0     0  135M      0  0:00:07  0:00:07  --- 133M

real    0m7,552s
user    0m0,285s
sys     0m1,148s
anegre@anegre-vm:~$ time curl --socks5-hostname localhost:1080 http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload Total Spent   Left  Speed
100 1024M 100 1024M    0     0  136M      0  0:00:07  0:00:07  --- 140M

real    0m7,560s
user    0m0,327s
sys     0m1,207s
```

Figura 5.4 - Rendimiento con buffer de 4KB.

```
anegre@anegre-vm:~$ time curl --socks5-hostname localhost:1080 http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload Total Spent   Left  Speed
100 1024M 100 1024M    0     0  496M      0  0:00:02  0:00:02  --- 496M

real    0m2,081s
user    0m0,097s
sys     0m0,682s
anegre@anegre-vm:~$ time curl --socks5-hostname localhost:1080 http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload Total Spent   Left  Speed
100 1024M 100 1024M    0     0  490M      0  0:00:02  0:00:02  --- 490M

real    0m2,104s
user    0m0,135s
sys     0m0,617s
anegre@anegre-vm:~$
```

Figura 5.5 - Rendimiento con buffer de 16KB.

```
anegre@anegre-vm:~$ time curl --socks5-hostname localhost:1080 http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload Total Spent   Left  Speed
100 1024M 100 1024M    0     0  549M      0  0:00:01  0:00:01  --- 549M

real    0m1,884s
user    0m0,279s
sys     0m0,482s
anegre@anegre-vm:~$ time curl --socks5-hostname localhost:1080 http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload Total Spent   Left  Speed
100 1024M 100 1024M    0     0  540M      0  0:00:01  0:00:01  --- 540M

real    0m1,928s
user    0m0,167s
sys     0m0,613s
anegre@anegre-vm:~$ time curl --socks5-hostname localhost:1080 http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload Total Spent   Left  Speed
100 1024M 100 1024M    0     0  547M      0  0:00:01  0:00:01  --- 547M

real    0m1,890s
user    0m0,118s
sys     0m0,650s
```

Figura 5.6 - Rendimiento con buffer de 32KB.

```
anegre@anegre-vm:~$ time curl http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload   Total  Spent   Left  Speed
100 1024M 100 1024M    0     0 1827M      0 --::-- --::-- 1848M

real    0m0,577s
user    0m0,083s
sys     0m0,319s
anegre@anegre-vm:~$ time curl http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload   Total  Spent   Left  Speed
100 1024M 100 1024M    0     0 2219M      0 --::-- --::-- 2226M

real    0m0,498s
user    0m0,068s
sys     0m0,314s
anegre@anegre-vm:~$ time curl http://barcito:80/largefile.bin -o /dev/null
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
                                         Dload  Upload   Total  Spent   Left  Speed
100 1024M 100 1024M    0     0 2316M      0 --::-- --::-- 2321M

real    0m0,458s
user    0m0,073s
sys     0m0,276s
```

Figura 5.7 - Rendimiento sin la utilización del proxy.

Para testear la simultaneidad de conexiones y los timeouts, se utilizó [ncat](#) para conectarse a otra instancia. A continuación, se intercambió información entre ambas instancias, y luego se dejó al cliente en modo inactivo. Las siguientes figuras muestran los registros del servidor evidenciando el intercambio, y la subsecuente eliminación por timeout expirado. Cabe destacar que desde la terminal del cliente se observa como el servidor cierra su lado de la conexión.

```
INFO: src/server/socks5.c:187, -----
INFO: src/server/socks5.c:188, [HANDLE_CONNECTION] Accepted new client: fd=6 from ::1:59060
DEBUG: src/server/socks5.c:361, [HELLO_READ] Entered hello_read
DEBUG: src/server/socks5.c:383, [HELLO_READ] Read 3 bytes from client.
DEBUG: src/server/socks5.c:403, [HELLO_READ] Received version: 0x05, nmethods: 1
DEBUG: src/server/socks5.c:428, [HELLO_READ] Have complete hello message (3 bytes)
DEBUG: src/server/socks5.c:439, [HELLO_READ] Client method 0: 0x00
DEBUG: src/server/socks5.c:460, [HELLO_READ] Selected NO_AUTH method
DEBUG: src/server/socks5.c:472, [HELLO_READ] Switching to write mode to send response
INFO: src/server/socks5.c:537, [WRITE_TO_CLIENT] Sent 2/2 bytes to client.
DEBUG: src/server/socks5.c:564, [WRITE_TO_CLIENT] Hello complete, transitioning to REQUEST_READ
DEBUG: src/server/socks5.c:710, [REQUEST_READ] Entered request_read.
DEBUG: src/server/socks5.c:744, [REQUEST_READ] Read 16 bytes from client.
DEBUG: src/server/socks5.c:758, [REQUEST_READ] Header: VER=0x05 CMD=0x01 RSV=0x00 ATYP=0x03
DEBUG: src/server/socks5.c:936, [REQUEST_READ] Parsed domain name localhost and port 80.
DEBUG: src/server/socks5.c:972, [REQUEST_RESOLVE] Starting DNS resolution for localhost
DEBUG: src/server/socks5.c:1028, [DNS_THREAD] Starting DNS resolution for localhost:80
DEBUG: src/server/socks5.c:1045, [DNS_THREAD] DNS resolution succeeded for localhost
INFO: src/server/socks5.c:1679, [DNS_RESOLVE] Resolved addresses for domain 'localhost':
INFO: src/server/socks5.c:1694, [DNS_RESOLVE] [1] IPv6: ::1:80
INFO: src/server/socks5.c:1694, [DNS_RESOLVE] [2] IPv4: 127.0.0.1:80
INFO: src/server/socks5.c:1698, [DNS_RESOLVE] Total addresses resolved: 2
DEBUG: src/server/libs(selector.c:538, [SELECTOR_NOTIFY] Notifying selector for fd=6
DEBUG: src/server/libs(selector.c:568, [SELECTOR_NOTIFY] EventFD notification sent successfully
DEBUG: src/server/socks5.c:1049, [DNS_THREAD] Selector notification sent for fd=6
DEBUG: src/server/libs(selector.c:696, [SELECTOR_SELECT] DNS notification received via eventfd (value=1)
DEBUG: src/server/socks5.c:309, [HANDLE_BLOCK] DNS completion handler called for fd=6
DEBUG: src/server/socks5.c:316, [HANDLE_BLOCK] Session state: 6, DNS failed: NO
DEBUG: src/server/socks5.c:339, [HANDLE_BLOCK] DNS resolution completed for fd=6
INFO: src/server/socks5.c:1059, [REQUEST_CONNECT] Attempting to connect to destination.
INFO: src/server/socks5.c:1099, [REQUEST_CONNECT] Attempting connection to IPv6: ::1:80
DEBUG: src/server/socks5.c:1136, [REQUEST_CONNECT] Connection in progress...
DEBUG: src/server/socks5.c:1169, [REMOTE_CONNECT_COMPLETE] Connection completed successfully
INFO: src/server/socks5.c:1210, [CONNECT_SUCCESS] Connection successful
INFO: src/server/libs/logger.c:77, [ACCESS] anonymous@::1:59060 -> localhost:80 (status=0x00)
DEBUG: src/server/socks5.c:1784, [ACCESS_LOG] anonymous@::1:59060 -> localhost:80 (status: 0x00)
DEBUG: src/server/socks5.c:1273, [ALLOCATE_BUFFERS] Remote buffers allocated successfully
DEBUG: src/server/socks5.c:952, [REQUEST_WRITE] Sending SOCKS5 success response to client
INFO: src/server/socks5.c:537, [WRITE_TO_CLIENT] Sent 22/22 bytes to client.
DEBUG: src/server/socks5.c:575, [WRITE_TO_CLIENT] Request complete, transitioning to RELAY
INFO: src/server/socks5.c:957, [REQUEST_WRITE] SOCKS5 handshake complete - connection established
DEBUG: src/server/socks5.c:965, [REQUEST_WRITE] Relay mode activated
```

Figura 5.8 - Negociación inicial y resolución DNS del remoto.

```
[INFO: src/server/socks5.c:1557] [REQUEST_WRITE] socks5->remote->complete - connection established
DEBUG: src/server/socks5.c:965, [REQUEST_WRITE] Relay mode activated
DEBUG: src/server/socks5.c:1332, [RELAY_DATA] Relay client to remote
DEBUG: src/server/socks5.c:1412, [RELAY] Relayed data client->remote
DEBUG: src/server/socks5.c:1575, [SOCKS5_REMOTE_READ] Resetting timeout to 1752574898
DEBUG: src/server/socks5.c:1576, [SOCKS5_REMOTE_READ] Relay remote to client
DEBUG: src/server/socks5.c:1484, [RELAY] Relayed data remote->client
DEBUG: src/server/socks5.c:1332, [RELAY_DATA] Relay client to remote
DEBUG: src/server/socks5.c:1412, [RELAY] Relayed data client->remote
INFO: src/server/libs(selector.c:537, [SELECTOR_SELECT]) Idle connection timeout for fd=6
DEBUG: src/server/socks5.c:1656, [HANDLE_ERROR] Handling error state for fd=6
DEBUG: src/server/socks5.c:1649, [SEND SOCKS5_ERROR] Prepared error response with code: 0x06
DEBUG: src/server/socks5.c:1664, [HANDLE_ERROR] Switching to STATE_ERROR_WRITE to send error response
INFO: src/server/socks5.c:537, [WRITE_TO_CLIENT] Sent 10/10 bytes to client.
DEBUG: src/server/socks5.c:549, [WRITE_TO_CLIENT] All data sent. Shutting down socket.
DEBUG: src/server/socks5.c:258, [SOCKS5_HANDLE_CLOSE] *** CLOSE HANDLER CALLED *** for fd=6
DEBUG: src/server/socks5.c:277, [SOCKS5_HANDLE_CLOSE] Client fd=6 closed
DEBUG: src/server/socks5.c:292, [SOCKS5_HANDLE_CLOSE] Closing remaining remote fd=7
DEBUG: src/server/socks5.c:258, [SOCKS5_HANDLE_CLOSE] *** CLOSE HANDLER CALLED *** for fd=7
DEBUG: src/server/socks5.c:266, [SOCKS5_HANDLE_CLOSE] Session already being cleaned up, skipping
DEBUG: src/server/socks5.c:299, [SOCKS5_HANDLE_CLOSE] Both fds closed, cleaning up session
DEBUG: src/server/socks5.c:258, [SOCKS5_HANDLE_CLOSE] *** CLOSE HANDLER CALLED *** for fd=6
DEBUG: src/server/socks5.c:266, [SOCKS5_HANDLE_CLOSE] Session already being cleaned up, skipping
DEBUG: src/server/socks5.c:1715, [CLEANUP] Starting session cleanup for client_fd=-1, remote_fd=-1
DEBUG: src/server/socks5.c:1722, [CLEANUP] Freeing domain name: localhost
DEBUG: src/server/socks5.c:1774, [CLEANUP] Session cleanup completed
DEBUG: src/server/socks5.c:304, [SOCKS5_HANDLE_CLOSE] Close handler complete for fd=6
```

Figura 5.9 – Relay de información y subsecuente timeout.

```
anegres-MacBook-Pro:TPE-Protos-72.07 anegre$ docker exec -it protos-container /bin/bash
root@5d1c8375c5b9:~# ncat -l localhost 80
Hello from client!
Hello from remote !
will stay idle now...
```

Figura 5.10 – Perspectiva del remoto.

```
root@5d1c8375c5b9:~# ncat --proxy localhost:1080 --proxy-type socks5 localhost 80
Hello from client!
Hello from remote !
will stay idle now...

hi
Ncat: Broken pipe.
root@5d1c8375c5b9:~#
```

Figura 5.11 – Perspectiva del cliente.

A continuación se buscó testear la concurrencia del servidor. Se lanzaron numerosas peticiones a dominios varios utilizando *curl* para observar el comportamiento del servidor². Si bien inicialmente se observó una alta concurrencia exitosa (más de 500 conexiones simultáneas al lanzar 1000 *curl requests* a la vez), al optimizar la limpieza de recursos y el funcionamiento general del servidor, se observó que la concurrencia bajó drásticamente. Si bien esto implica una mejora en la performance del proxy, significó mayor dificultad para testear la concurrencia del mismo. Los resultados se evidencian en los registros del servidor y el resultado del test.

² El script utilizado fue *test_many_curl.sh*.

```
INFO: src/server/socks5.c:1361, [RELAY] Client closed connection
^CINFO: src/server/main.c:248, Received signal 2, shutting down server
INFO: src/server/libs/metrics.c:53, [METRICS] Server shutdown - Final statistics:
INFO: src/server/libs/metrics.c:54, [METRICS] Total connections: 1000
INFO: src/server/libs/metrics.c:55, [METRICS] Max concurrent: 30
INFO: src/server/libs/metrics.c:56, [METRICS] Bytes received: 868151
INFO: src/server/libs/metrics.c:57, [METRICS] Bytes sent: 857951
INFO: src/server/libs/metrics.c:58, [METRICS] Total bytes transferred: 1726102
INFO: src/server/libs/metrics.c:59, [METRICS] Total errors: 0
INFO: src/server/libs/metrics.c:60, [METRICS] Network errors: 0
INFO: src/server/libs/metrics.c:61, [METRICS] Protocol errors: 0
INFO: src/server/libs/metrics.c:62, [METRICS] Auth errors: 0
INFO: src/server/libs/metrics.c:63, [METRICS] System errors: 0
INFO: src/server/libs/metrics.c:64, [METRICS] Timeout errors: 0
INFO: src/server/libs/metrics.c:65, [METRICS] Memory errors: 0
INFO: src/server/libs/metrics.c:66, [METRICS] Other errors: 0
INFO: src/server/libs/metrics.c:67, [METRICS] Server uptime: 143 seconds
INFO: src/server/libs/metrics.c:68, [METRICS] Metrics system cleaned up
```

Figura 5.12 - Registros del servidor al correr test_many_curl.sh.

```
Launching request 1000 to http://amazon.com
All 1000 concurrent requests completed WITH PROXY.
Successful connections: 1000
Failed connections: 0
[05:26:02.478851] "...
```

Figura 5.13 - Resultados de la prueba.

Como “prueba de fuego” se configuró Firefox para redirigir su tráfico a través del servidor. Se utilizó en varias ocasiones, visitando varias páginas en simultáneo sin ningún problema. Si fue notable, sin embargo, el incremento de velocidad al pasar del buffer de 4KB al de 32.

6. Conclusiones

El desarrollo del proxy SOCKS5 y su protocolo de monitoreo permitió poner en práctica los conceptos de multiplexación y programación de sockets vistos durante el transcurso de la materia, a la vez que evidenció la dificultad del diseño de protocolos y lógica adecuados a los requerimientos de un servidor de las características solicitadas. Se logró implementar un servidor SOCKS5 conforme a los RFCs pertinentes y acorde a los objetivos propuestos, pudiendo implementar además funcionalidades adicionales. El presente trabajo, sin embargo, deja una base sólida sobre la cual pueden agregarse nuevas *features*, como persistencia de datos, mayor soporte de comandos, y optimización de recursos. En conclusión, queda evidenciada la importancia de una planificación cuidadosa para la solución de problemas de concurrencia, poniendo especial énfasis en la gestión adecuada de recursos y la anticipación de desafíos de escalabilidad.

7. Guía de instalación

7.1. Compilación

Puede compilarse el trabajo en un entorno Linux con *make* y *gcc* utilizando el comando *make* en el directorio root del proyecto. Adicionalmente, pueden limpiarse los binarios con *make clean*.

Se provee, además, una imagen minimalista de Docker para compilar el trabajo y ejecutarlo. La misma puede utilizarse ejecutando los scripts provistos en el directorio

“script”: en orden, deben correrse build.sh, run.sh e install.sh (dentro del contenedor). Para ejecutar el proyecto sin Docker, basta con ejecutar install.sh.

7.2. Ejecución

La compilación dará por resultado los binarios correspondientes al servidor (socks5d) y al programa de administración (client) que utiliza el protocolo de monitoreo implementado. Ambos se encontrarán en el directorio bin.

7.2.1. Servidor

Para ejecutar el servidor puede utilizar el comando:

```
./bin/socks5d
```

Los administradores deben ser creados junto al comando para iniciar el servidor mediante -a user:password. Opcionalmente, pueden añadirse usuarios con la flag -u.

```
./bin/socks5d -a admin:admin -u user1:user1 -u user1:user1
```

Opcionalmente, se puede especificar un puerto para Socks5 con -p y uno para management con -P de la siguiente manera:

```
./bin/socks5d -a admin:admin -p 1090 -P 8090
```

De forma similar, también se puede cambiar la dirección de Socks5 y management con -l y -L. Además, es posible obtener el detalle de los argumentos aceptados con -h.

Se agregaron las flags -g/ --log <LOG_LEVEL> para manejar el nivel de logging del servidor. Además, se agregó la flag -s (silent) para deshabilitar el logging.

7.2.2. Cliente

Para ejecutar el cliente puede utilizar el comando:

```
./bin/client
```

Opcionalmente, se puede especificar un host y puerto de la siguiente manera:

```
./bin/client -h localhost -p 9090
```

Por defecto se intentará ejecutar la UI gráfica de Dialog. Para utilizar la UI de la consola, puede utilizar la flag --console. Además, es posible obtener el detalle de los argumentos aceptados con --help.

8. Instrucciones para la configuración

Para configurar el tamaño del *buffer* o el tiempo para *timeout*, se deben seguir los pasos a continuación:

1. Iniciar el servidor y cliente como se indicó previamente, asegurándose de crear al menos un administrador.
2. Autenticarse en el cliente con un usuario y contraseña de administrador.
3. Ingresar a “Manage settings”.
4. Seleccionar “Change buffer size” o “Change timeout”.
5. Ingresar un nuevo valor válido y confirmar el cambio.

Para verificar que los cambios hayan tomado lugar, es posible consultar ambos valores en “Show configurations”.

9. Ejemplos de configuración y monitoreo

Una vez iniciada la sesión en el cliente, el administrador se encontrará con el menú principal. En el mismo, puede decidir ver las métricas, los logs del servidor, cambiar las configuraciones o salir del cliente. A continuación se presentan imágenes de cada una de las opciones, entrando recursivamente en sus respectivos menús, si los hay.

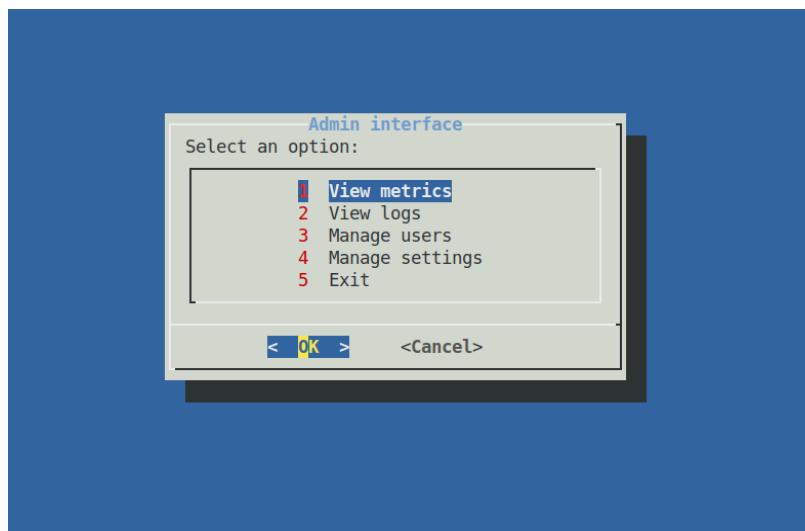


Figura 9.1 - Menú principal de la aplicación cliente.

La primera opción -la vista de métricas- permite vislumbrar el estado del servidor, la cantidad de conexiones actual, las máximas conexiones concurrentes, los bytes que entraron y salieron, el tiempo que el servidor lleva activo y una variedad de errores.

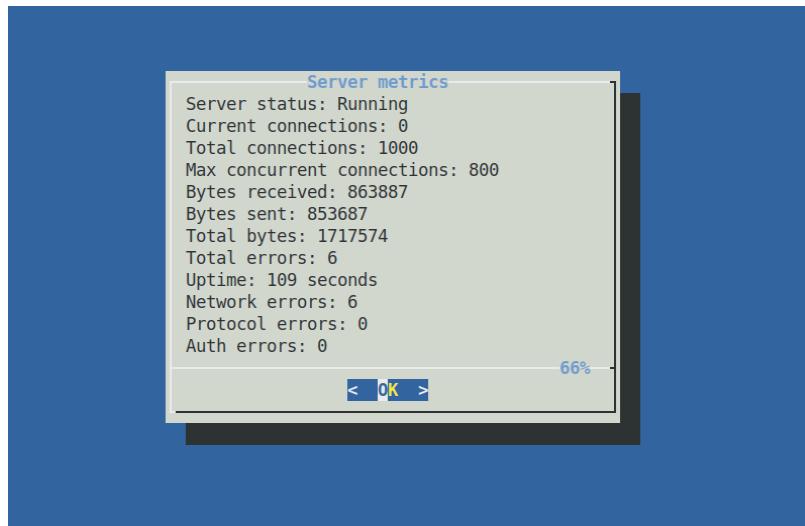


Figura 9.2 - Vista de métricas.

La segunda opción lleva a los logs. Los mismos están paginados, a razón de 10 por página, y cada uno muestra la fecha, usuario (siendo *anonymous* de no autenticarse), tipo de registro (siempre es 'A'), puerto e IP de origen, destino, puerto de destino y el status code de la acción.

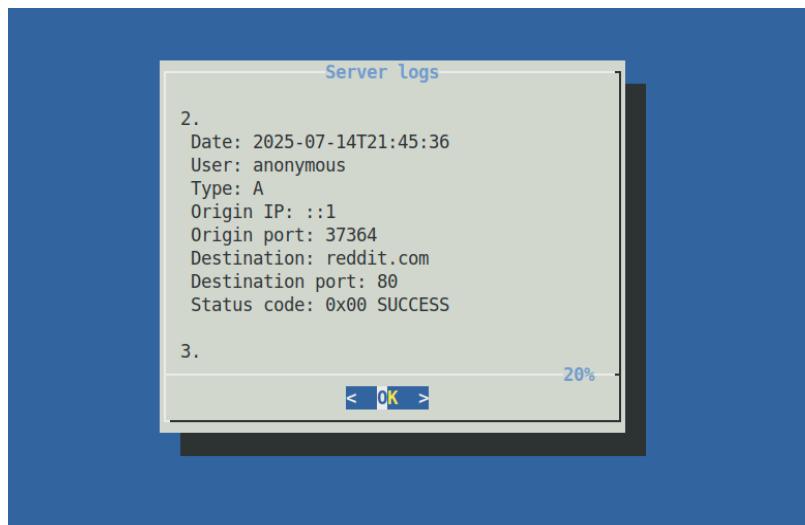


Figura 9.3 - Vista de logs.

La tercera opción -"Manage users"- lleva a un nuevo menú, en el cual es posible listar los usuarios del servidor y agregar o remover un usuario.

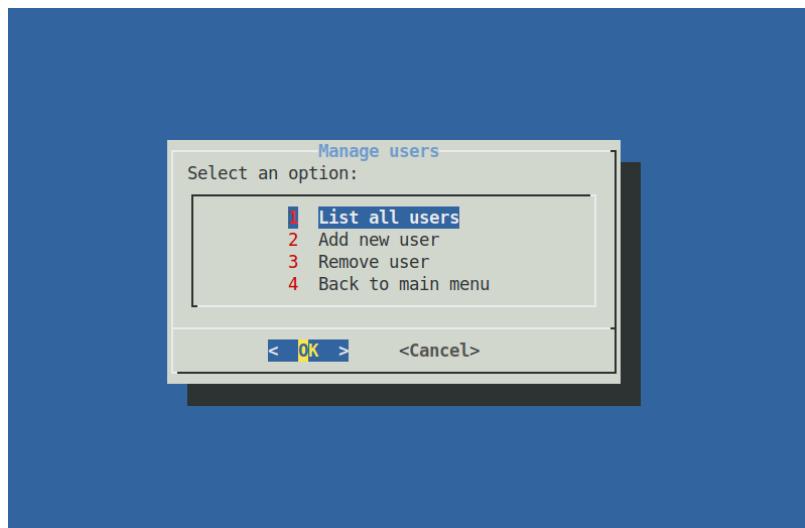


Figura 9.4 - Menú de usuarios.

Al igual que los *logs*, la lista de usuarios se encuentra paginada. Además del nombre, se muestra el tipo de usuario, pudiendo el mismo ser “Administrator” o “User”.

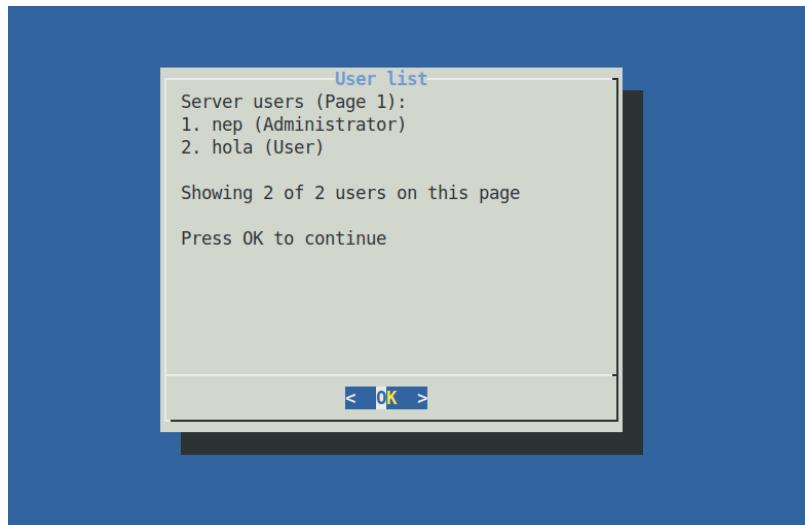


Figura 9.5 - Lista de usuarios.

Para agregar un usuario, debe ingresarse un nombre (que no puede ser repetido) y una contraseña. Esta última deberá ser ingresada nuevamente para confirmarla, y tiene que contener al menos cuatro caracteres, una letra y un número.

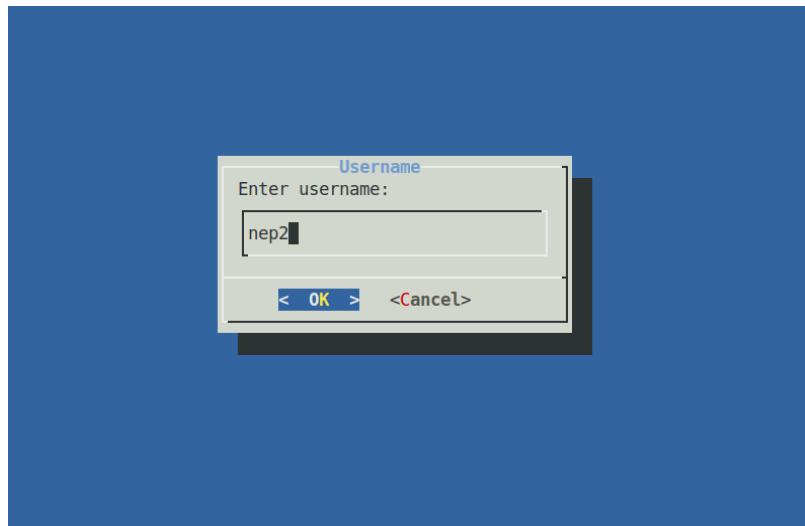


Figura 9.6 - Agregar usuario.

Para eliminar un usuario se cuenta con un cuadro de entrada de texto, el cual posteriormente le notificará al administrador si la operación fue exitosa. De fallar, tiene la capacidad de indicar que fue porque el usuario era inexistente.

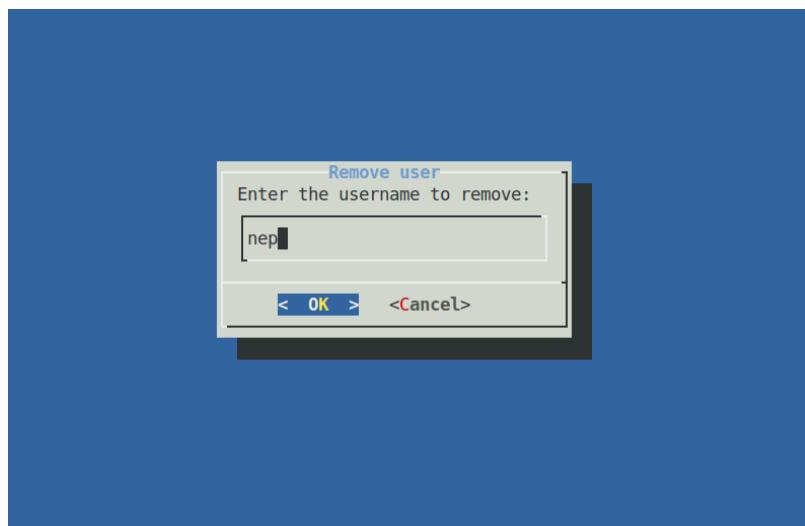


Figura 9.7 - Eliminar usuario.

Pasando a las configuraciones, se cuenta con un nuevo menú. Este ofrece la posibilidad de mostrar las configuraciones actuales, de cambiar el tamaño del *buffer* y de cambiar el tiempo para *timeout*.

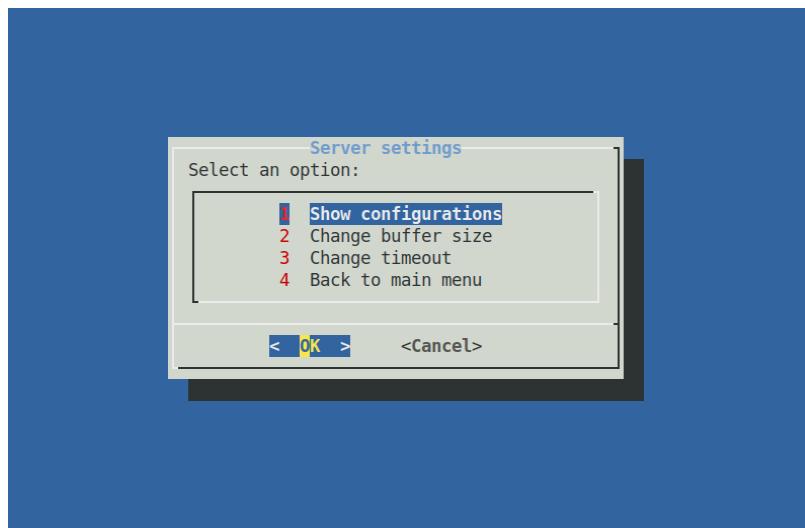


Figura 9.8 - Menú de configuraciones.

Para la vista de configuraciones, el administrador puede visualizar la dirección del servidor y el puerto para el servicio de *management*, junto al tamaño de *buffer* y el tiempo para *timeout*.

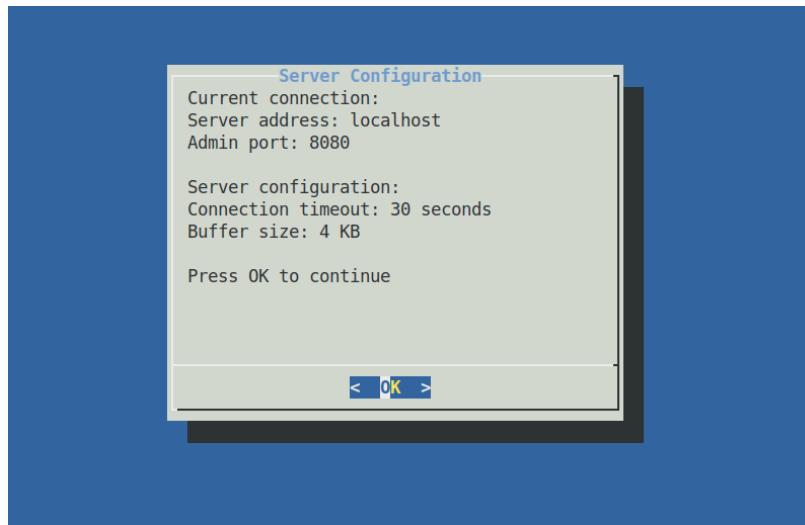


Figura 9.9 - Vista de configuraciones.

Para ingresar un nuevo tamaño de *buffer*, basta con completarlo y confirmarlo. El nuevo valor debe estar entre 1 y 64 kilobytes, pero activamente se le recomienda al usuario que ingrese uno mayor a 4KB.



Figura 9.10 - Configuración: tamaño de buffer.

Por último, para el timeout el mínimo es 1 segundo y el máximo 60. No se cuentan con recomendaciones para el usuario.

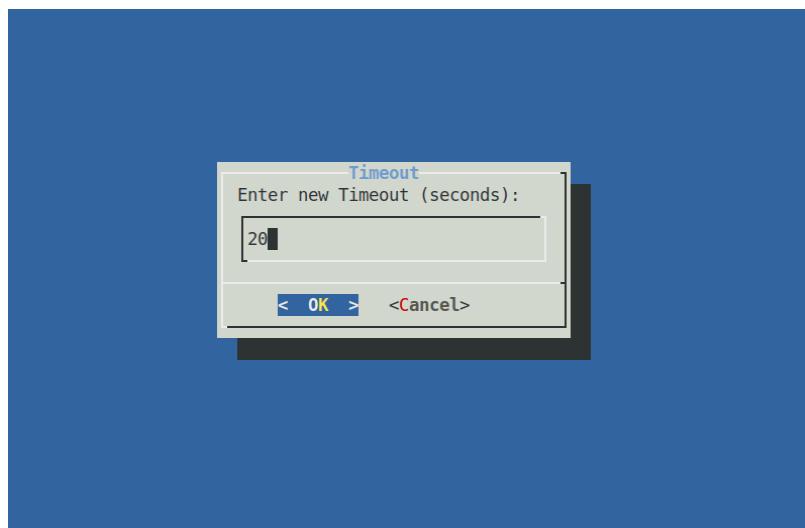


Figura 9.11 - Configuración: tiempo para timeout.

Habiendo finalizado con las funciones ofrecidas por el cliente, a continuación se presentan dos ventanas adicionales. La que se encuentra a continuación es la asociada a paginación, permitiendo moverse a la página anterior o siguiente.

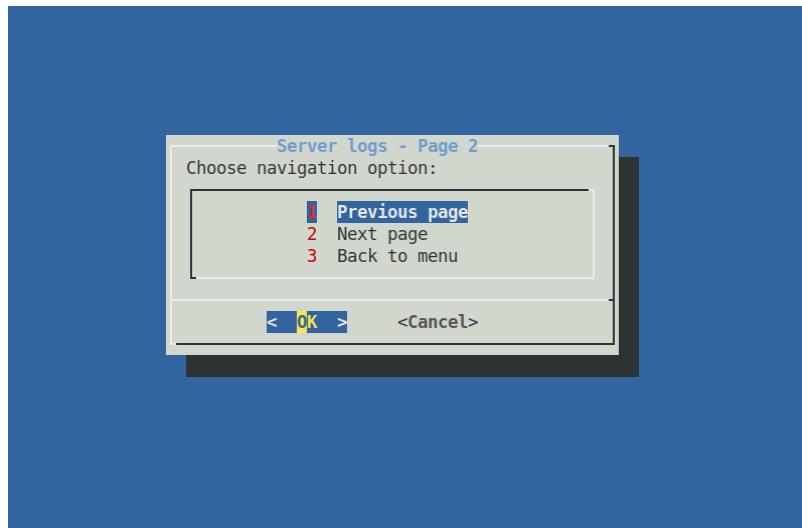


Figura 9.12 - Menú de paginación.

Además, como ya se mencionó anteriormente, el administrador tiene la posibilidad de reconectarse si se pierde la conexión con el servidor, volviendo a la ventana en la que se encontraba antes del fallo.

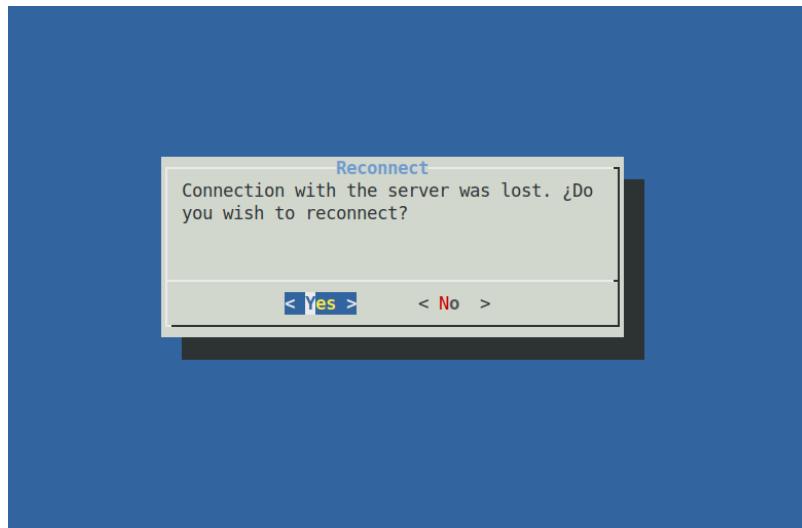


Figura 9.13 - Reconexión al servidor.

Cabe destacar que todos los menús y ventanas presentadas siguen la misma lógica y funcionan de la misma manera en la UI de consola, como se puede apreciar en las siguientes imágenes de ejemplo.

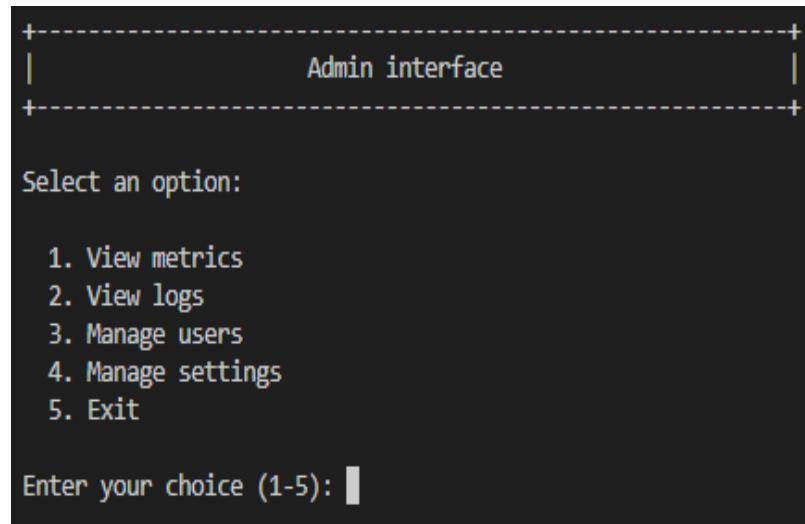


Figura 9.14 - Menú principal de la aplicación cliente en modo consola.

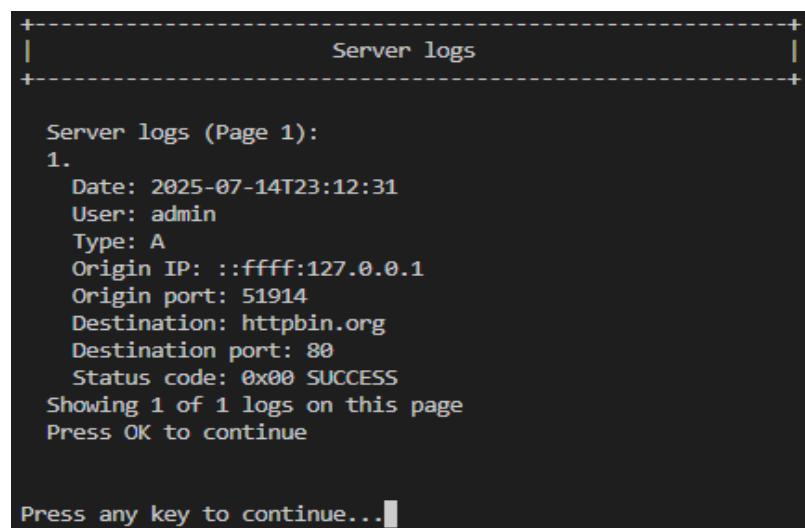


Figura 9.15 - Vista de logs en modo consola.

10. Documentos de diseño

10.1. Servidor

En el servidor, se utilizó una variedad de librerías provistas por la cátedra; selector.h, netutils.h, logger.h, buffer.h y args.h. Como es de esperarse en un proyecto de este estilo, casi todas sufrieron cambios a lo largo del desarrollo, habiendo sido cubiertos previamente los más relevantes.

En cuanto al protocolo SOCKS, se creó la librería socks5.h, que centraliza las estructuras de datos y estados necesarios para manejar las conexiones cliente-servidor de forma no bloqueante. Se optó por un enfoque basado en una máquina de estados finitos, motivado tanto por el hecho de que operaciones como recv, send o connect son potencialmente bloqueantes como también porque la cátedra proveía un selector que permite multiplexar múltiples descriptores de archivos. Así, se separó la división del procesamiento de cada conexión en etapas bien definidas. Los estados incluyen desde la

lectura y escritura del saludo inicial (STATE_HELLO_READ, STATE_HELLO_WRITE), pasando por autenticación (STATE_AUTH_READ, STATE_AUTH_WRITE), resolución DNS (STATE_REQUEST_RESOLVE), conexión al destino (STATE_REQUEST_CONNECT) y *relay* de datos (STATE_RELAY), incluyendo estados terminales como STATE_DONE, STATE_ERROR y STATE_ERROR_WRITE.

Vale aclarar que STATE_ERROR se utiliza exclusivamente para manejar los errores en la etapa de *request/reply*, mientras que STATE_ERROR_WRITE fue introducido para permitir una gestión más general de errores en cualquier etapa de la conexión que requiera notificar el error al cliente y proceder con un cierre ordenado. Este cierre se realiza directamente mediante una llamada a selector_unregister_fd, sin utilizar shutdown() ni esperar el cierre explícito por parte del cliente. Aunque el [RFC 1928](#) explicita que, en casos como la selección del método 0xFF el cliente debe cerrar la conexión, se decidió no depender de dicho comportamiento para evitar cuelgues o la retención de recursos en casos de clientes poco conformes.

Para encapsular la información asociada a una única sesión cliente, se diseñó la estructura client_session, que contiene tanto el estado actual de la sesión como los descriptores de archivo (client_fd, remote_fd), buffers de lectura/escritura para ambos extremos de la conexión, datos del protocolo (connection_data) y metadatos relevantes como tiempo de inicio, timeout, autenticación y flags de error.

Respecto a management y el protocolo CalSetting, se implementaron las librerías management.h y management_commands.h. La primera contiene las estructuras necesarias, y maneja la conexión con los clientes del protocolo de métricas, mientras que la segunda permitió modularizar los comandos aceptados por el protocolo, proveyendo las funciones específicas para cada uno de los pedidos de la interfaz de administrador, generando cada una de las estructuras que se le enviarán al cliente.

Para capturar métricas, se creó el apropiadamente llamado metrics.h. Son sus funciones las que se llaman en momentos pertinentes de la implementación del servidor SOCKS. La estructura creada para este propósito se exhibe a continuación, contando con todos los campos ya explicados con anterioridad.

```
typedef struct {
    uint64_t total_connections;
    uint32_t concurrent_connections;
    uint32_t max_concurrent_connections;
    uint64_t bytes_transferred_in;
    uint64_t bytes_transferred_out;
    uint64_t total_bytes_transferred;
    uint32_t error_counts[ERROR_TYPE_OTHER + 1];
    uint32_t total_errors;
    time_t start_time;
} server_metrics;
```

Para las configuraciones, se crearon config.h y config.c. Su propósito es proveer los valores tanto de las configuraciones modificables por el administrador como las inamovibles.

Adicionalmente, se implementó una librería de utilidades: socks5_utils.h. La misma se encarga de mapear errores varios a los específicos de SOCKS y de crear y almacenar bloques de información de direcciones IPv4 e IPv6.

10.2. Cliente

En cuanto al cliente, lib_client.h se encarga de realizar propiamente cada una de las peticiones al servidor, y de leer y devolver las respuestas en un formato conveniente para su presentación.

También se cuentan con librerías específicas para las UIs (ui_console.h y ui_dialog.h), junto al adaptador ui_adapter.h ya desarrollado en la sección de problemas encontrados.

Adicionalmente, se creó una librería de paginación (pagination.h), la cual se utilizó para mostrar la lista de usuarios y de logs. Esto es debido a que se consideró que -particularmente en los logs, dado que el número máximo de usuarios es limitado- era irracional y más bien complejo mostrar -potencialmente- centenares de entradas a la vez, y más aún en una interfaz gráfica.

Otra librería que se creó para el cliente es validation.h, y es la encargada de validar usuarios, contraseñas, tamaños de buffer, timeouts o cualquier otro tipo de dato que pueda ingresar el usuario. En algunos casos particulares -como el que respecta al tamaño del buffer, por ejemplo- se le advierte al usuario si el valor que ingresó es poco conveniente, teniendo la opción de recapacitar o de seguir adelante con el cambio.

Por último, se cuenta con client_constants.h, librería que ofrece tamaños de buffer predeterminados, mínimos, máximos y otras constantes relevantes al cliente.