

INFORME TRABAJO PRÁCTICO ESPECIAL

DISEÑO E IMPLEMENTACIÓN DE UN PROXY SOCKS5

v1.0.0

Nombre	Apellido	Legajo	E-mail
Agustin	Galan	64098	aggalan@itba.edu.ar
Ramiro	Garcia	61133	rgarcia@itba.edu.ar
Federico	Viera	62022	fviera@itba.edu.ar
Ivo	Vilamowski	64210	ivilamowski@itba.edu.ar

1. Introducción	2
2. Descripción de los protocolos y aplicaciones desarrolladas	2
3. Problemas encontrados	6
4. Limitaciones de la aplicación	9
5. Posibles extensiones	9
6. Conclusiones	10
7. Ejemplos de prueba	10
8. Guía de instalación	16
9. Instrucciones para la configuración	16
10. Ejemplos de configuración y monitoreo	20
11. Documento del diseño del proyecto	22
	24

1. Introducción

El presente trabajo práctico consiste en el diseño e implementación de un servidor proxy para el protocolo **SOCKS v5 [RFC 1928]**, complementado de un protocolo de administración para dicho servidor, con capacidades de configuración y logueo.

2. Descripción de los protocolos y aplicaciones desarrolladas

Nuestro Protocolo

El protocolo presentado en este documento se ha diseñado para gestionar un proxy **SOCKS5** a través de paquetes **UDP**. La implementación se basa en un protocolo de gestión ligero y eficiente, que facilita tanto el control como la monitorización del proxy de manera efectiva. El protocolo es fácil de analizar y extender debido a su encabezado binario fijo y la posibilidad de agregar una carga útil opcional en formato ASCII.

Versión del Protocolo:

Como los protocolos vistos en clase incluimos una versión que nos ayuda a mantener el protocolo a lo largo del tiempo. Por el momento estamos utilizando la versión 1.

Estructura del Mensaje:

El formato del mensaje tiene un encabezado de 6 bytes seguido de una carga útil opcional que puede tener hasta 1024 bytes. Este enfoque nos ayuda con la rapidez que pretendemos para este simple protocolo, desde los mensajes y su procesamiento en ambas direcciones (cliente-servidor). La estructura del encabezado incluye los siguientes campos:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Versión	Método	Estado	Longitud		Reservado
Indica la versión del protocolo.	Representa el código de la operación que se está solicitando o respondiendo (ej., LOGIN, STATS).	Indica si el mensaje es una solicitud o una respuesta, además del resultado de la operación.	Indica la longitud de la carga útil en bytes (<i>en formato de orden de red</i>).		Un campo de 1 byte reservado para futuras extensiones.

Códigos de Método:

El protocolo soporta varios métodos que permiten la administración de usuarios y la supervisión del servidor SOCKS5. Los métodos permiten realizar tareas como la autenticación de usuarios, la obtención de estadísticas del proxy, la creación y eliminación de usuarios, y la gestión de registros de acceso. Los métodos definidos son los siguientes:

Código	Método	Definición
1	MGMT_LOGIN	Autenticación de usuario y contraseña.
2	MGMT_STATS	Recuperación de estadísticas del proxy.
3	MGMT_ADDUSER	Creación de un nuevo usuario.
4	MGMT_DELUSER	Eliminación de un usuario existente.
5	MGMT_LISTUSERS	Listado de todos los usuarios.
6	MGMT_SETAUTH	Habilitación/deshabilitación de la autenticación.
7	MGMT_DUMP	Volcado de los últimos N registros de acceso.
8	MGMT_SEARCHLOGS	Búsqueda de registros con un término de filtro.
9	MGMT_CLEARLOGS	Borrado de todos los registros de acceso.
10	MGMT_LOGOUT	Cierra la cesión del administrador.
11	MGMT_ADDADMIN	Creación de un nuevo usuario administrador
12	MGMT_DELADMIN	Eliminación de un usuario administrador
13	MGMT_LISTADMINS	Listado de todos los administradores
14	MGMT_SETBUFFER	Setea el nuevo tamaño de buffer

Códigos de Estado:

El protocolo utiliza un conjunto de códigos de estado para indicar el resultado de una solicitud:

Código	Método	Definición
0	MGMT_REQ	El paquete es una solicitud del cliente.
20	MGMT_OK_SIMPLE	La operación fue exitosa sin devolver datos adicionales.
21	MGMT_OK_WITH_DATA	La operación fue exitosa y la respuesta incluye datos.
40	MGMT_ERR_SYNTAX	El comando o los argumentos están malformados.
41	MGMT_ERR_AUTH	Las credenciales son incorrectas
42	MGMT_ERR_NOTFOUND	Método desconocido.
43	MGMT_ERR_NO_AUTH	No se está autenticado.
44	MGMT_NOT_SUPPORTED	La versión solicitada no es soportada
45	MGMT_ERR_LOGGED_IN	Ya se está logueado, no hace falta hacerlo de nuevo
46	MGMT_ERR_EXPIRED	La sesión actual ha caducado.
47	MGMT_ERR_BUSY	Otro administrador está logueado
50	MGMT_ERR_INTERNAL	Error interno del servidor.
51	MGMT_PAYLOAD_TOO_LONG	El payload supera el máximo esperado

Codificación de la Carga Útil:

La carga útil de cada datagrama puede incluir datos ASCII de hasta **1024 bytes**. La codificación ASCII es simple y permite que los argumentos o datos sean fácilmente legibles y procesables. En el caso de la operación **LOGIN**, la carga útil consistirá en un par de tokens (usuario y contraseña) separados por un espacio.

Consideraciones:

Además de esto optamos por un adminclient, que se encarga de darle una interfaz a todo lo antes documentado. Consideramos apropiado que por motivos de seguridad la aplicación cliente se encuentre limitada a soportar solo a un administrador concurrente, además de establecer un timeout de 30 segundos de inactividad, que en caso de activarse la sesión del mismo se cierra.

Servidor Proxy Socks5:

Arquitectura General

El servidor proxy utiliza una arquitectura orientada a eventos basada en un multiplexor de entrada/salida con selectores, lo que le permite manejar múltiples conexiones concurrentes de manera eficiente.

El servidor principal crea dos interfaces de red: una para escuchar conexiones TCP del protocolo SOCKS5 y otra para recibir comandos de administración por UDP.

Implementación del Protocolo SOCKS5

El proxy implementa el protocolo SOCKS5 semi completo mediante una máquina de estados que atraviesa las siguientes fases:

1. Fase de Negociación

En esta fase, el cliente y el servidor negocian el método de autenticación a utilizar. El servidor admite tanto autenticación sin credenciales (NO_AUTH) como autenticación con usuario y contraseña (USER_PASS), dando prioridad al método más seguro si está disponible.

2. Fase de Autenticación (Opcional)

Si se selecciona autenticación con usuario y contraseña, el cliente debe enviar sus credenciales, las cuales son validadas contra una lista de usuarios configurados. El proxy permite hasta 10 usuarios simultáneos.

3. Procesamiento de Solicitudes

El proxy analiza solicitudes del tipo CONNECT, con soporte para direcciones IPv4, IPv6 y nombres de dominio. En el caso de nombres de dominio, realiza la resolución DNS de forma asincrónica utilizando hilos separados.

4. Establecimiento de Conexión

El proxy establece conexiones con los servidores de destino, intentando múltiples direcciones en caso de que una resolución DNS devuelva varias opciones. Se realiza un mapeo detallado de errores del sistema a los códigos de respuesta definidos por el protocolo SOCKS5.

5. Relay de Datos

Una vez establecida la conexión, el proxy entra en modo de retransmisión, donde reenvía datos en ambas direcciones entre el cliente y el servidor de origen. Para ello, utiliza un sistema eficiente de buffers para gestionar el tráfico.

Gestión de Conexiones

Cada conexión SOCKS5 se representa mediante una estructura interna que mantiene el estado de la conexión, los buffers de entrada/salida, los analizadores del protocolo y la información de resolución DNS. El proxy gestiona correctamente la liberación de recursos, incluyendo el cierre de descriptores de archivo y la liberación de memoria.

3. Problemas encontrados

En primer lugar, al iniciar el trabajo práctico, lo primero que hicimos *-dado a la firme recomendación de la cátedra-* fue familiarizarnos con el código provisto por la misma, para comprender su funcionamiento y cementar una base de conocimientos de la cual construir el resto del trabajo práctico. Esto resultó ser un problema ya que, entre otras características del código, como fue la primera vez que veíamos una State Machine en la carrera, tuvimos dificultades para entender el funcionamiento del mismo. Sumado a esto, ya que decidimos leer el RFC 1928 mucho después del código, tuvimos dificultades al principio del trabajo práctico, ya que, no teníamos cómo debía comportarse el servidor, lo cual resultó en muchas funciones o estructuras que necesitaron cambios o refacciones a lo largo de la implementación de este.

La siguiente dificultad encontrada fue lograr la conexión a una dirección IPv4 y IPv6, que nos resultó el paso lógico previo a conexiones a FQDN. Al implementar esta funcionalidad protocolo, nos olvidamos de la posibilidad de que la conexión estuviese en proceso, lo cual, si no se analiza el estado del `errno`, una conexión en proceso (Osea, con `errno = EINPROGRESS`) podía confundirse con un error y se termina la conexión.

Una vez solucionado este tema, nos adentramos en el soporte para FQDN, donde encontramos varios problemas. Primero, al crear el thread para resolver DNS de forma asíncrona, al llamar a `selector_notify_block()` para notificar al selector principal se producía un segmentation fault que no se originaba en nuestro código directamente. Este fallo fue resuelto mediante una correcta sincronización entre threads: se protegió el

acceso a la lista `resolution_jobs` con `pthread_mutex_lock` y `pthread_mutex_unlock`, y se utilizó `pthread_kill` para notificar de forma segura al thread del selector.

Posteriormente, una vez resuelto ese tema, al fallar la conexión con una IP resuelta, queríamos que se intentara con todas las IPs devueltas por `getaddrinfo()`. Sin embargo, la máquina de estados quedaba trabada en el estado `REQUEST_CONNECTING` al intentar con la segunda IP. Esto se resolvió inicializando correctamente `current_addr` al finalizar la resolución DNS, y manejando adecuadamente el fallback: al fallar una conexión, `current_addr` se avanza a `ai_next` y se vuelve a invocar `request_create_connection()` con los nuevos datos. Además, se verificó que el estado `REQUEST_CONNECTING` esté correctamente configurado para manejar eventos de lectura, escritura y bloqueo, asegurando que la máquina de estados pueda avanzar correctamente ante fallos o éxitos de conexión.

El siguiente problema que luego de muchos dolores de cabeza pudimos resolver fue el paquete de respuesta del request socks. Al probar el comando curl con **FQDN**, en vez de una dirección de **IPv4** o **IPv6**, no logramos que funcione el protocolo de manera correcta. Al utilizar lo conocimientos aprendidos en la materia, con *Wireshark* logramos identificar que el problema era que el paquete de respuesta no estaba siendo formado de manera correcta y por ende, el cliente no respondía, ya que le estamos enviando un **DOMAINNAME** en vez de una dirección resuelta **IPv4** o **IPv6**.

El último problema destacable durante la implementación del protocolo fue que, al intentar utilizar nuestro Proxy en un navegador para evaluar cómo manejaba el tráfico, descubrimos que este no lograba establecer ninguna conexión, lo que nos llevó al descubrimiento de que en la respuesta de **IPv4** o **IPv6**, no especificamos los puertos (Utilizábamos puertos "Dummy"), lo cual hacía que los navegadores descarten nuestras respuestas por ser invalidadas.

4. Limitaciones de la aplicación

Nuestra implementación del Protocolo cuenta con las siguientes limitaciones:

- La aplicación cliente de administración cuenta con una **capacidad máxima de 100 usuarios registrados**. Si bien esta podría ser superior, no encontramos razón lógica por la cual se debería expandir esta capacidad.
- La aplicación cliente de administración posee la capacidad de atender **solo a un administrador en simultáneo** por motivos de seguridad. La implementación actual compromete la integridad de la autenticación en caso de varios usuarios, pero da espacio a mejoras en el futuro.
- La capacidad de conexiones concurrentes de nuestro servidor proxy socks5 se encuentra **limitada por la cantidad de file descriptors** capaz de soportar la implementación de selector (1024). Para evitar que nos quedemos sin file descriptors, optamos por rechazar las conexiones que superen las 500 concurrentes. Esto podría ser mejorado usando poll.

5. Posibles extensiones

Algunas funcionalidades a incorporar en futuras versiones del sistema con el objetivo de enriquecer su capacidad operativa y mejorar la experiencia del usuario y del administrador podrían ser:

- **Soporte para DNS over HTTPS (DoH):** Permitir resolver FQDN a través de servidores DoH para mayor privacidad y seguridad en la resolución de nombres.
- **Persistencia de métricas:** Actualmente, las métricas son volátiles. Se podría incorporar una base de datos embebida o persistencia en archivos para almacenar estadísticas históricas.
- **Monitoreo remoto con autenticación robusta:** Ampliar el protocolo de monitoreo para permitir la consulta remota autenticada.
- **Configuración mediante interfaz web:** Añadir una pequeña UI web para gestionar parámetros del servidor en tiempo real y visualizar métricas en dashboards gráficos.

- **Bloqueo de sitios específicos:** Pensando en un proxy que usaría una facultad como ITBA, se podría aplicar un filtro en cuanto a que sitios uno puede acceder y rechazar las peticiones de acceso a las mismas.
- **Sistema de permisos diferenciados por usuario:** Implementar distintos niveles de acceso a funcionalidades o sitios según el usuario autenticado.
- **Compatibilidad con UDP ASSOCIATE y BIND:** Extender la implementación para soportar conexiones UDP, completando el RFC.

6. Conclusiones

Durante el desarrollo de este Trabajo Práctico Especial se logró implementar un servidor SOCKS5 robusto y flexible, cumpliendo con los requisitos funcionales y no funcionales definidos por la cátedra. La arquitectura fue diseñada desde el inicio con modularidad y escalabilidad en mente, lo que permitió agregar componentes como métricas, monitoreo y reconfiguración en tiempo de ejecución sin comprometer la estabilidad del sistema.

El manejo de conexiones en modo no bloqueante multiplexado representó un desafío ya que no nos imaginamos un servidor atendiendo múltiples conexiones sin el uso de threads, pero se pudo alcanzar una alta eficiencia en la utilización de recursos del sistema.

El trabajo también nos permitió ver la importancia de algunos límites de la implementación actual, como la falta de persistencia de métricas, o la ausencia de una interfaz gráfica para la administración. Sin embargo, estas áreas representan una oportunidad clara para futuras extensiones del sistema.

En síntesis, se logró un sistema robusto y extensible, que nos permitió ver el funcionamiento de herramientas que utilizamos todos los días pero no nos percatamos de que están ni somos conscientes de cómo funcionan. La experiencia adquirida durante este trabajo resulta sumamente valiosa, especialmente en lo referente a diseño de protocolos, programación de bajo nivel y administración de recursos en servidores concurrentes.

7. Ejemplos de prueba

Nuestro protocolo, como indicado por el RFC-1928, permite el uso del mismo con FQDN, direcciones “literales” IPv4 y direcciones “literales” IPv6. También, con navegadores que lo permiten (i.e. Firefox), es posible configurar el proxy para manejar el tráfico web. En las siguientes figuras se presentan varios ejemplos de uso cubriendo estas posibilidades de funcionamiento dentro del protocolo:



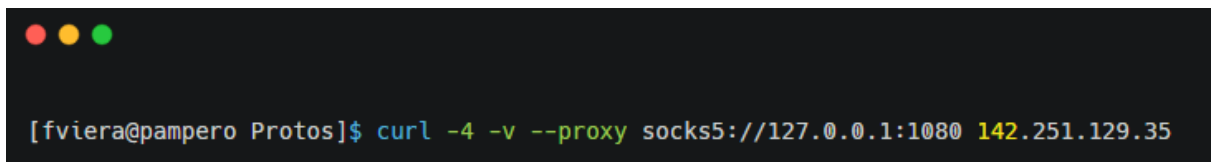
```
[fviera@pampero Protos]$ curl -4 -v --proxy socks5://127.0.0.1:1080 https://www.google.com/
```

Figura 7.1: Ejemplo de Prueba usando el FDQN de www.google.com



```
[fviera@pampero Protos]$ curl -6 -v --proxy socks5://127.0.0.1:1080 "https://[2606:4700:4700::1111]/"
```

Figura 7.2: Ejemplo de prueba utilizando una IPv6



```
[fviera@pampero Protos]$ curl -4 -v --proxy socks5://127.0.0.1:1080 142.251.129.35
```

Figura 73: Ejemplo de prueba utilizando una IPv4

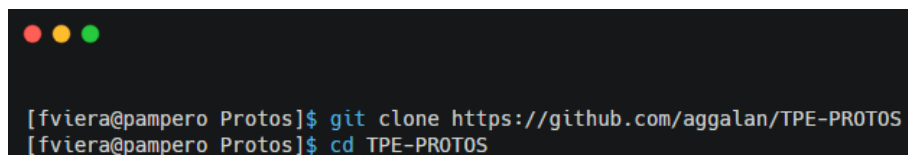
8. Guia de instalacion

8.1. Requisitos del Sistema

1. Sistema **POSIX** (probado en macOS y WSL)
2. gcc o clang con soporte **C99**
3. **make** y **pkg-config**
4. (Opcional) Librería Check para test unitarios

8.2. Instalación Paso a Paso

1. Clonar el Repositorio (o descomprimir el .zip)



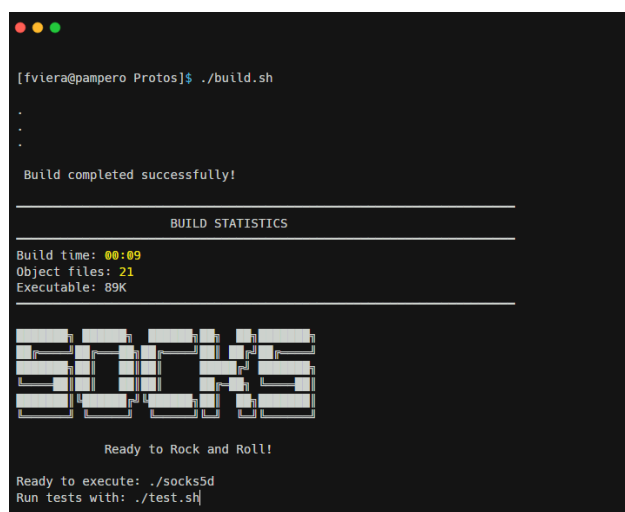
```
[fviera@pampero Protos]$ git clone https://github.com/aggalan/TPE-PROTOS
[fviera@pampero Protos]$ cd TPE-PROTOS
```

Figura 8.1: Consola simulada clonando el repositorio del Tp

2. Compilación Principal

2.1. Solución Personalizada

El proyecto utiliza un un Makefile que detecta automáticamente las dependencias y provee estadísticas del proceso de building:



```
[fviera@pampero Protos]$ ./build.sh
.
.
.
Build completed successfully!

BUILD STATISTICS
-----
Build time: 00:09
Object files: 21
Executable: 89K

██████████
██████████
██████████
██████████
██████████
██████████
██████████
██████████
██████████
██████████

Ready to Rock and Roll!

Ready to execute: ./socks5d
Run tests with: ./test.sh
```

Figura 8.2: Consola simulada ejecutando [./build.sh](#)

Este makefile es capaz de reconocer flags pertinentes a la compilación como `-a/-admin`, para compilar el cliente de administración y `-h/-help`, para que el usuario pueda ver las opciones de compilación diferentes (Cabe destacar, el script obligatoriamente hace una limpieza de ejecutables previo a la compilación, entonces, si uno quiere compilar el ejecutable principal y el client de administrador, deberá hacer al menos uno con make):



```
[fviera@pampero Protos]$ ./build.sh -a
.
.
.

Build completed successfully!

BUILD STATISTICS

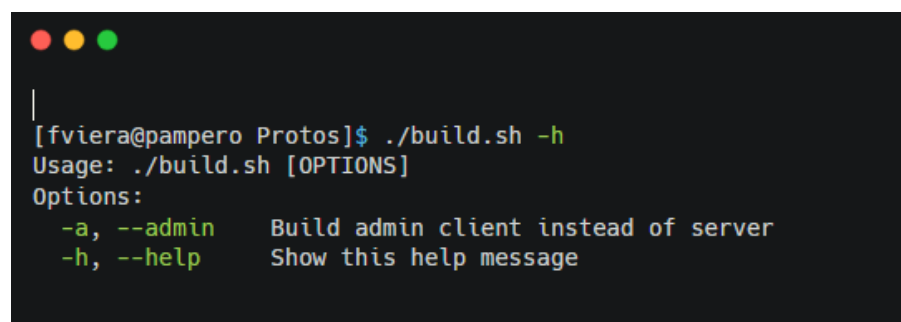
Build time: 00:09
Object files: 21
Executable: 85K
Build mode: Admin Client

SOCCER

Admin Client Ready!

Ready to execute: ./admin_client
Admin client built successfully! Build completed successfully!
```

Figura 8.2: Consola simulada ejecutando `./build.sh -a`




```
[fviera@pampero Protos]$ ./build.sh -h
Usage: ./build.sh [OPTIONS]
Options:
-a, --admin    Build admin client instead of server
-h, --help     Show this help message
```

Figura 8.3: Consola simulada ejecutando `./build.h -h`

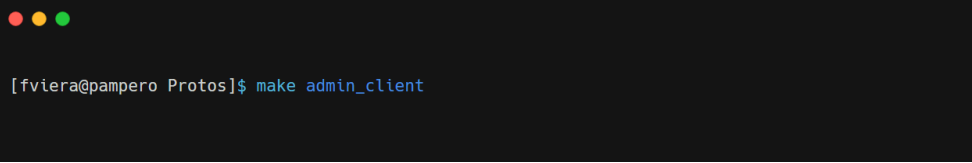
2.2. Solución Genérica

Alternativamente, el trabajo práctico puede ser compilado simplemente utilizando :



```
[fviera@pampero Protos]$ make
```

Figura 8.5: Consola simulada compilando

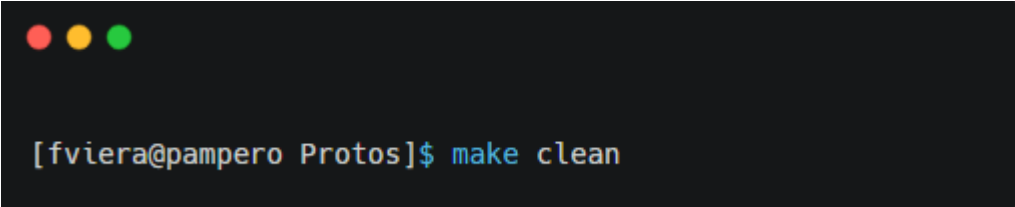


```
[fviera@pampero Protos]$ make admin_client
```

Figura 8.6: Consola simulada compilando en modo admin

8.3. Limpieza del Proyecto

A pesar de que nuestro archivo personalizado de **build.sh** hace una limpieza del proyecto previo a cada compilación, si el usuario desea limpiar el proyecto, basta con que corra la siguiente línea:

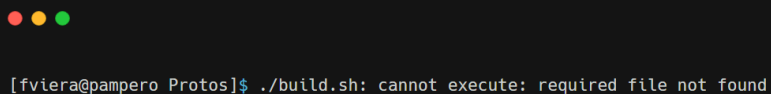


```
[fviera@pampero Protos]$ make clean
```

Figura 8.7: Consola simulando limpiando ejecutables del proyecto

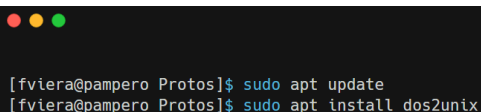
8.4. Troubleshooting

En casos extremos, nuestros scripts pueden no estar en formato DOS. En ese caso, el usuario recibirá un error como:

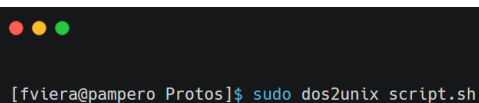
A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The prompt is [fviera@pampero Protos]\$ and the command is ./build.sh. The output is: cannot execute: required file not found.

```
[fviera@pampero Protos]$ ./build.sh: cannot execute: required file not found
```

Para esto, utilizamos dos2unix:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The prompt is [fviera@pampero Protos]\$ and the command is sudo apt update. The output is: [fviera@pampero Protos]\$ sudo apt install dos2unix.

```
[fviera@pampero Protos]$ sudo apt update
[fviera@pampero Protos]$ sudo apt install dos2unix
```

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The prompt is [fviera@pampero Protos]\$ and the command is sudo dos2unix script.sh.

```
[fviera@pampero Protos]$ sudo dos2unix script.sh
```

9. Instrucciones para la configuración

9.1. Configuración Proxy

El protocolo implementado permite varios comandos de línea para configurar el Proxy:

Flag	Descripción
-l <SOCKS addr>	Dirección en la que escuchará el proxy SOCKS. (default: 0.0.0.0)
-p <SOCKS port>	Puerto del proxy SOCKS. (default: 1080)
-L <conf addr>	Dirección del servicio de administración. (default: 127.0.0.1)
-P <conf port>	Puerto del servicio de administración. (default: 8080)
-u <name>:<pass>	Agrega usuarios al iniciar (máx. 10 usuarios).

9.2. Configuración Administrador

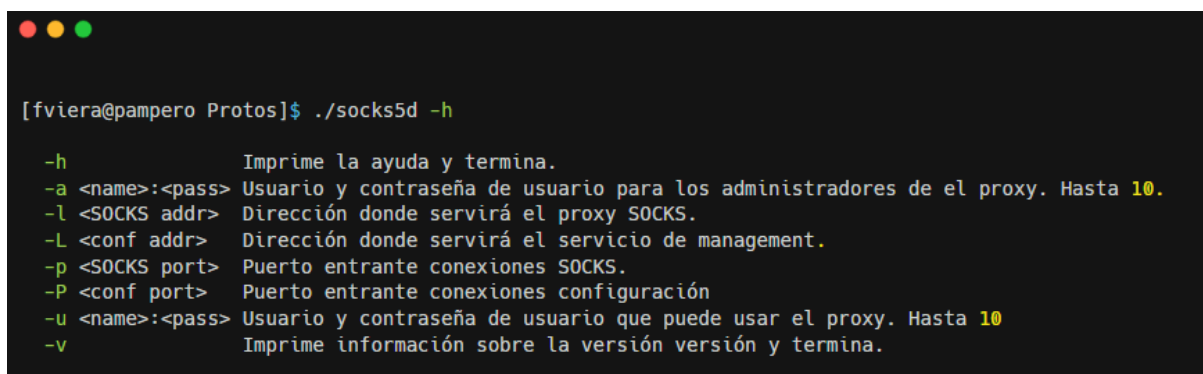
También, permite varios comandos de línea para configurar el Admin Client:

Comando	Descripción
adduser <u> <p>	Agrega un nuevo usuario con contraseña
deluser <u>	Elimina un usuario existente
listusers	Muestra todos los usuarios configurados actualmente
addadmin <u> <p>	Agrega un nuevo admin con contraseña
listadmins	Muestra todos los admins configurados actualmente

10. Ejemplos de configuración y monitoreo

El protocolo implementado tiene capacidades de configuración y monitoreo. Incluyendo la autenticación de usuarios, métricas en tiempo real, logging detallado y un cliente de administrador interactivo. El sistema, como mencionado anteriormente, está diseñado para poder soportar hasta 500 conexiones concurrentes e incluye pruebas exhaustivas para validar el rendimiento y la configuración.

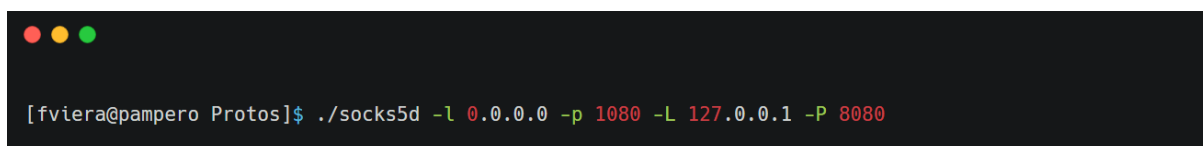
10.1. Help



```
[fviera@pampero Protos]$ ./socks5d -h

-h           Imprime la ayuda y termina.
-a <name>:<pass> Usuario y contraseña de usuario para los administradores de el proxy. Hasta 10.
-l <SOCKS addr> Dirección donde servirá el proxy SOCKS.
-L <conf addr> Dirección donde servirá el servicio de management.
-p <SOCKS port> Puerto entrante conexiones SOCKS.
-P <conf port> Puerto entrante conexiones configuración
-u <name>:<pass> Usuario y contraseña de usuario que puede usar el proxy. Hasta 10
-v           Imprime información sobre la versión versión y termina.
```

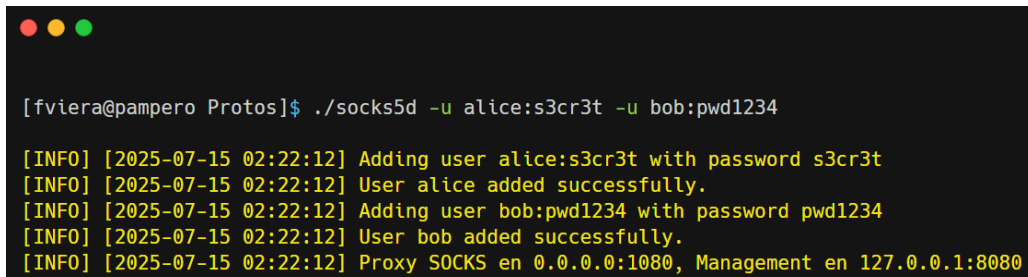
Figura 10.1



```
[fviera@pampero Protos]$ ./socks5d -l 0.0.0.0 -p 1080 -L 127.0.0.1 -P 8080
```

Figura 10.2: Ejemplo de uso de Flags

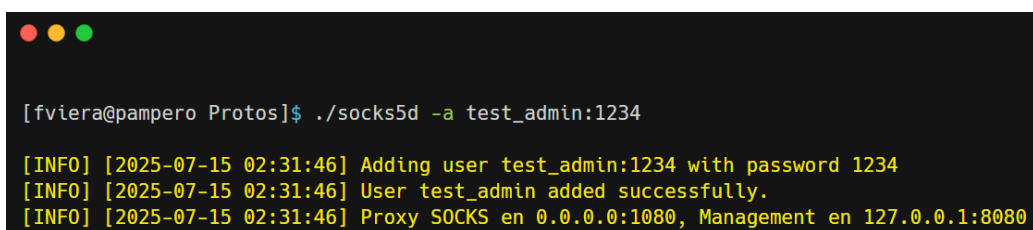
10.2. Add User



```
[fviera@pampero Protos]$ ./socks5d -u alice:s3cr3t -u bob:pwd1234

[INFO] [2025-07-15 02:22:12] Adding user alice:s3cr3t with password s3cr3t
[INFO] [2025-07-15 02:22:12] User alice added successfully.
[INFO] [2025-07-15 02:22:12] Adding user bob:pwd1234 with password pwd1234
[INFO] [2025-07-15 02:22:12] User bob added successfully.
[INFO] [2025-07-15 02:22:12] Proxy SOCKS en 0.0.0.0:1080, Management en 127.0.0.1:8080
```

Figura 10.3

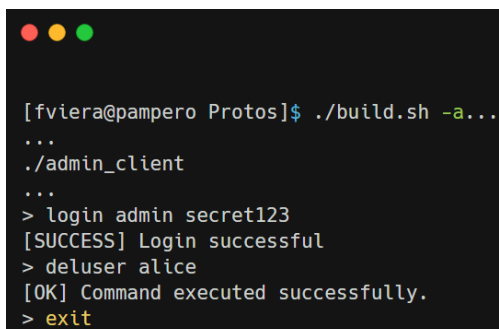


```
[fviera@pampero Protos]$ ./socks5d -a test_admin:1234

[INFO] [2025-07-15 02:31:46] Adding user test_admin:1234 with password 1234
[INFO] [2025-07-15 02:31:46] User test_admin added successfully.
[INFO] [2025-07-15 02:31:46] Proxy SOCKS en 0.0.0.0:1080, Management en 127.0.0.1:8080
```

Figura 10.4

10.3. Delete User



```
[fviera@pampero Protos]$ ./build.sh -a...
...
./admin_client
...
> login admin secret123
[SUCCESS] Login successful
> deluser alice
[OK] Command executed successfully.
> exit
```

Figura 10.5

10.4. Stress Test y Monitoreo

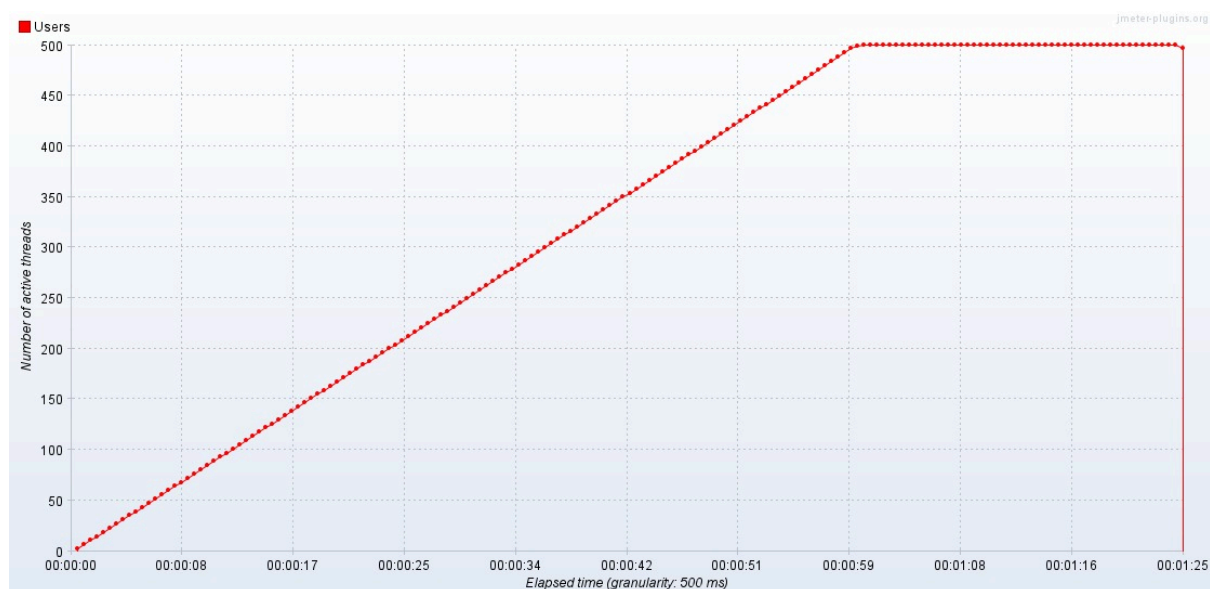


Figura 10.6: Cantidad de usuarios concurrentes en función del tiempo

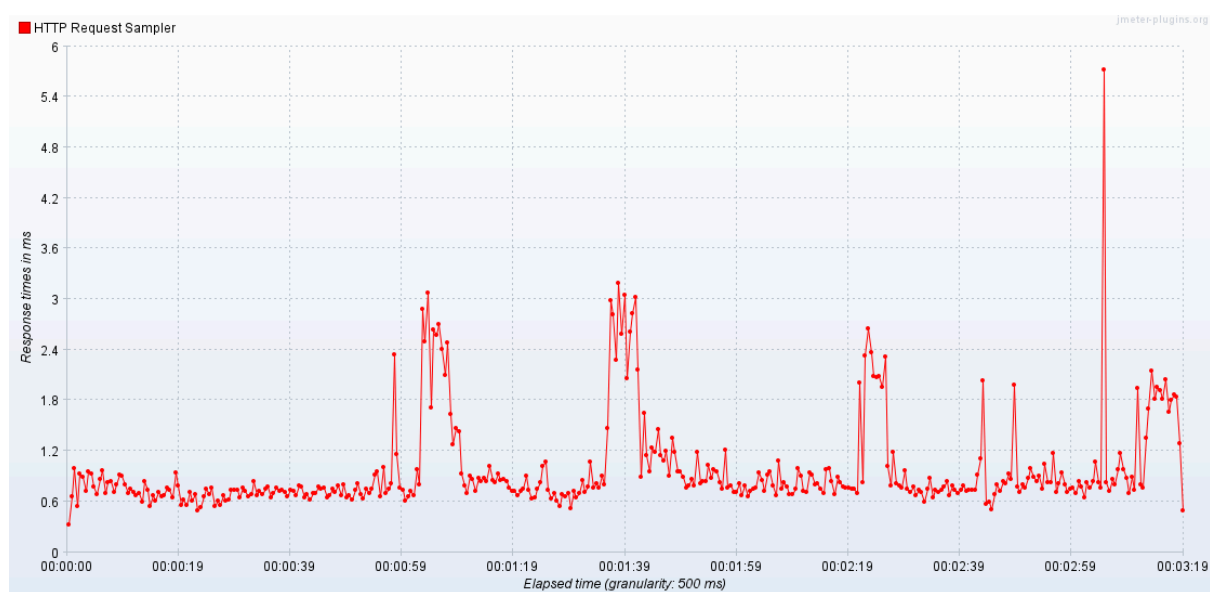


Figura 10.7: Response Time (con Proxy)

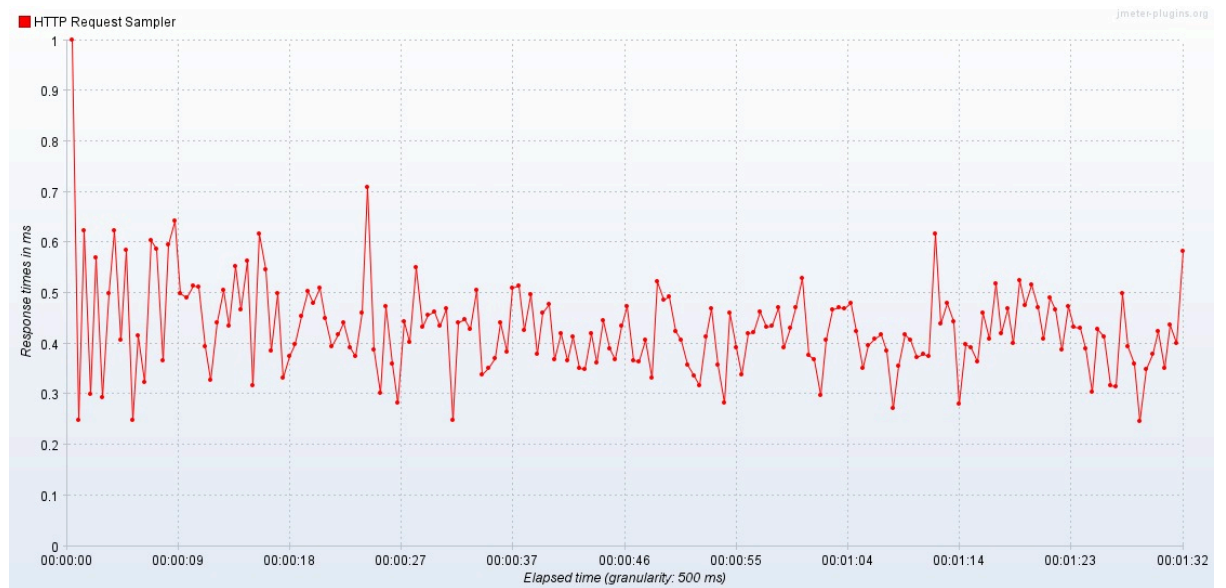


Figura 10.8: Response Time Sin (Proxy)

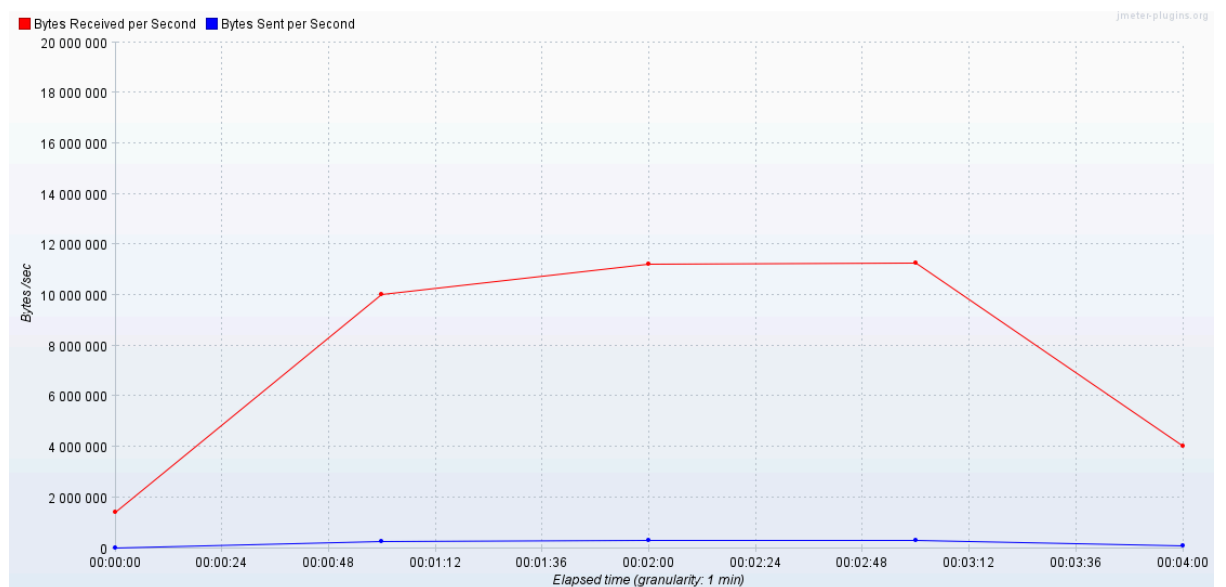


Figura 10.9: Bytes Throughput Over Time(Proxy)

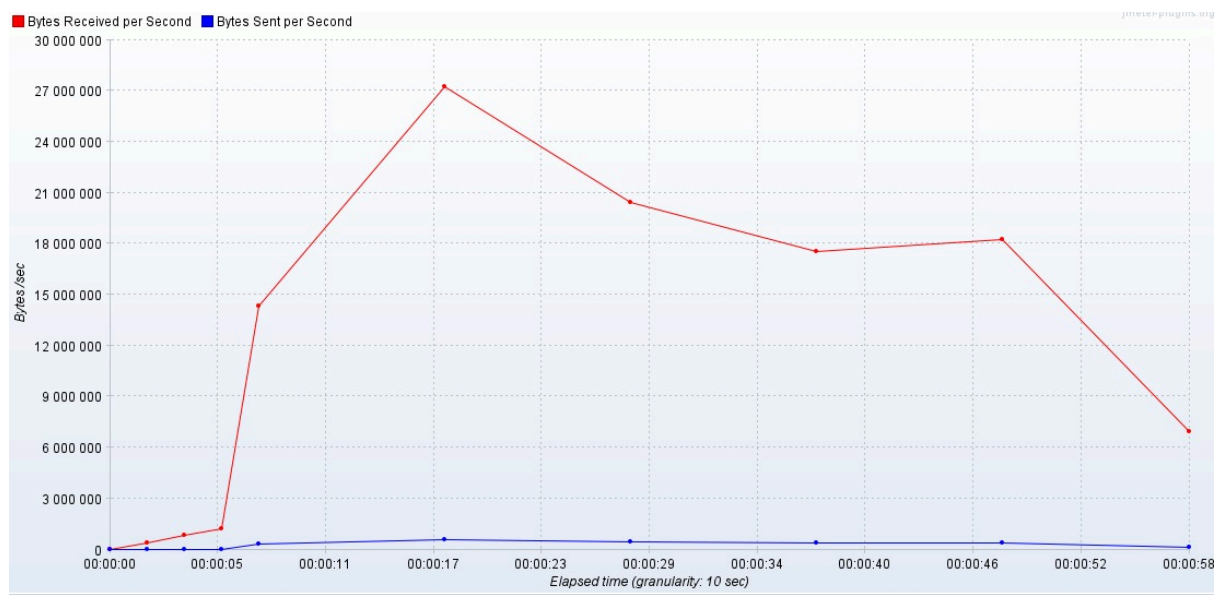


Figura 10.10: Bytes Throughput Over Time(Sin Proxy)

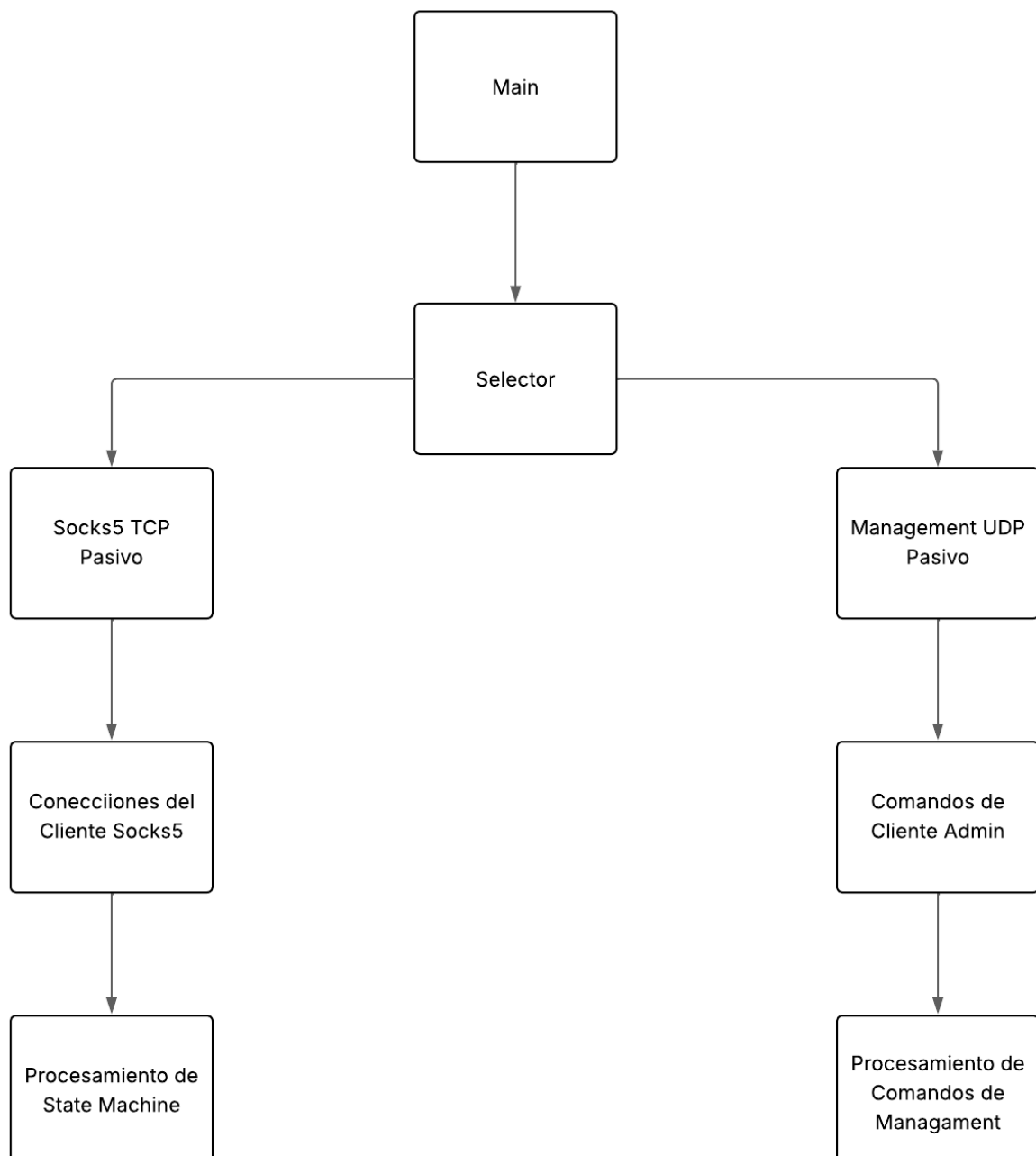
11. Documento del diseño del proyecto

11.1. Arquitectura Basada en Eventos con Selector

El núcleo del sistema está construido sobre un multiplexor de eventos personalizado que permite manejar operaciones de entrada/salida (I/O) sobre file descriptors dentro de un solo thread de ejecución no bloqueante. Este selector abstrae la implementación final.

La aplicación principal configura tanto la interfaz SOCKS como la de administración dentro del mismo proceso.

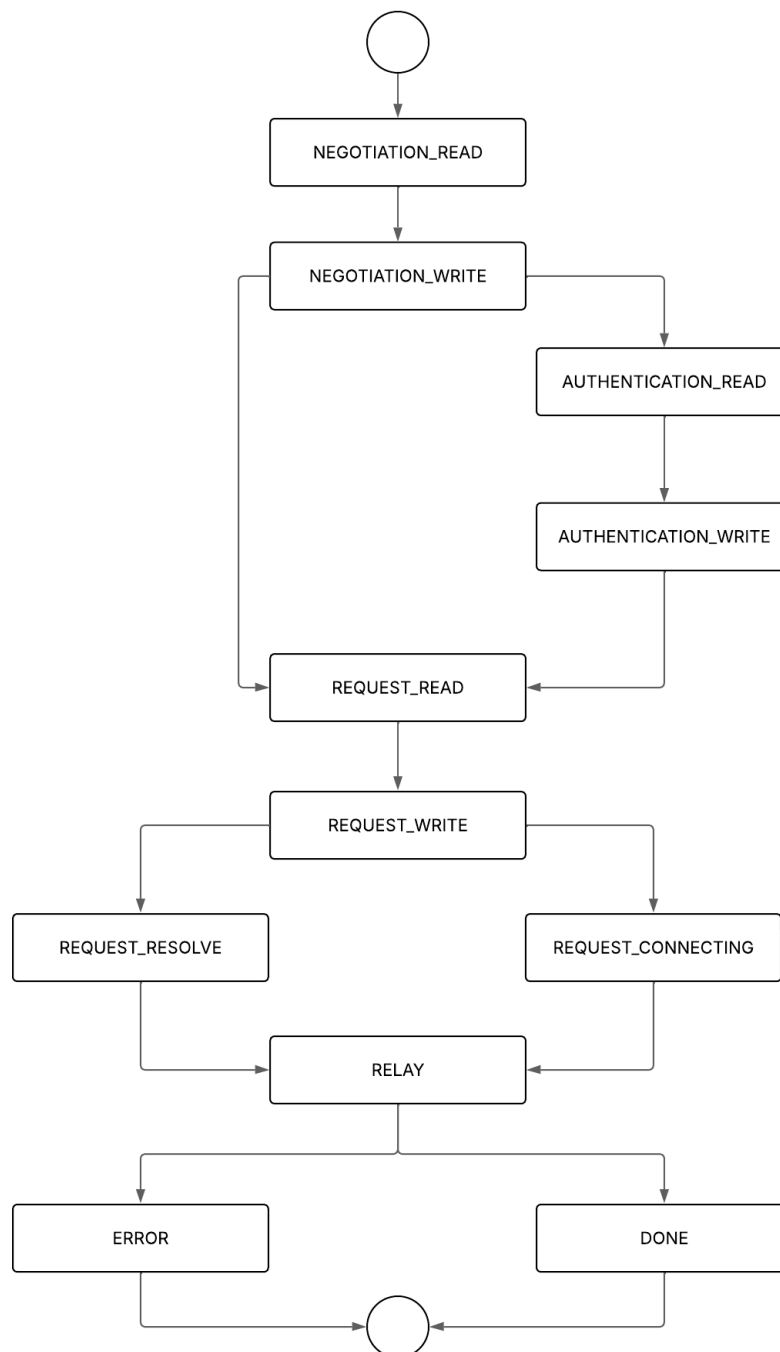
Se crean dos sockets, un socket TCP para conexiones SOCKS5 y otro para el admin client. Podemos observar esto en el siguiente diagrama:



11.2. Implementación de Máquina de Estados

La implementación del protocolo SOCKS5 se basa en una máquina de estados finita donde los eventos que provocan transiciones provienen del selector de eventos. Los estados posibles incluyen: **NEGOTIATION_READ**, **NEGOTIATION_WRITE**, **AUTHENTICATION_READ**, **AUTHENTICATION_WRITE**, **REQUEST_READ**, **REQUEST_WRITE**, **REQUEST_CONNECTING**, **REQUEST_RESOLVE**, **RELAY**, **DONE**, **ERROR**, **ORIGIN_CONNECT** y **ORIGIN_CONNECT_WRITE**

Cada estado define manejadores específicos para: *on_arrival*, *on_departure*, *on_read_ready*, *on_write_ready* y *on_block_reads*



11.3. Interfaz de Gestión

El sistema expone un servicio de gestión mediante un puerto UDP independiente, utilizando un protocolo binario personalizado. Este protocolo incluye los siguientes códigos de método:

- *MGMT_LOGIN, MGMT_LOGOUT*
- *MGMT_STATS*
- *MGMT_ADDUSER, MGMT_DELUSER, MGMT_LISTUSERS*
- *MGMT_SETAUTH*
- *MGMT_DUMP, MGMT_SEARCHLOGS, MGMT_CLEARLOGS*

El protocolo utiliza un header estructurado con campos de versión, método, estado, longitud y reservado.

11.4. Diseño Modular

El sistema sigue un diseño modular con separación clara de responsabilidades entre componentes:

- **Autenticación:** Validación de credenciales de usuario
- **Métricas:** Seguimiento de conexiones históricas, bytes transferidos y conexiones activas
- **Buffers:** Gestión eficiente de la entrada/salida
- **Logs:** Registro de accesos y salida de depuración
- **Utilidades de red:** Manejo de conexiones
- **Parsers:** Interpretación y validación de mensajes del protocolo