Trabajo Práctico Especial

Protocolos de Comunicación

Grupo 16

Juan Ignacio Cantarella, 64509
Máximo Daniel Carranza, 64538
Lola Díaz Varela, 62792
Lucas Di Candia, 63212

ITBA
Instituto Tecnológico de Buenos Aires

1. Introducción

El Trabajo Práctico Especial de la materia *Protocolos de Comunicación* consistió en la implementación completa de un servidor proxy que cumple con el estándar SOCKS versión 5, definido en el RFC 1928, incorporando a su vez el mecanismo de autenticación usuario/contraseña especificado en el RFC 1929. Además del servidor, la consigna requirió el diseño e implementación de un protocolo de aplicación propio destinado al monitoreo y la configuración en tiempo de ejecución del sistema, así como el desarrollo de un cliente capaz de interactuar con dicho protocolo.

A lo largo de este informe se detalla el proceso de diseño, desarrollo e integración de todos estos componentes, haciendo foco en las decisiones de arquitectura, el manejo de sockets no bloqueantes con multiplexación, y los mecanismos necesarios para garantizar robustez, concurrencia, eficiencia y escalabilidad. También se presentan las métricas recolectadas, los mecanismos de logging de accesos, las herramientas creadas para administrar usuarios y parámetros operativos, y las pruebas de estrés utilizadas para evaluar el comportamiento del sistema bajo carga.

2. Protocolos y aplicaciones desarrolladas

a. Servidor proxy SOCKS5

El núcleo del proyecto es el servidor SOCKS5. Este servidor implementa una máquina de estados finitos (FSM) para gestionar el ciclo de vida de cada conexión, asegurando que ninguna operación de red bloquee la ejecución del servidor. Los estados implementados cubren las fases de negociación de versión (HELLO), autenticación (AUTH), solicitud de conexión (REQUEST) y transferencia de datos (COPY).

El servidor soporta el comando CONNECT, permitiendo a los clientes establecer conexiones TCP a través del proxy. Se ha puesto especial énfasis en el manejo correcto de buffers y en la robustez ante errores.

Estructura del Mensaje de Solicitud SOCKS5

VER	CMD	RSV	ATYP	DST.ADDR	DST.PORT
Versión del protocolo. Debe ser siempre 5 para SOCKS	Representa el código de la operación que se está solicitando o respondiendo	Reservado. Debe establecerse siempre en 0x00. Se guarda para uso futuro del protocolo.	Tipo de dirección.	Dirección de destino. Es la dirección del servidor final al que el cliente quiere conectarse.	Puerto de destino. El puerto del servidor final, en orden de red (Big Endian).

b. Protocolo de administración S5ADMIN

Para cumplir con los requerimientos de monitoreo y configuración en tiempo de ejecución, se diseñó e implementó el protocolo de administración S5ADMIN. A diferencia de la propuesta original sobre TCP, este protocolo opera sobre UDP (predeterminado en el puerto 8080) para minimizar la sobrecarga y ofrecer un mecanismo de control ligero y desacoplado del flujo principal de datos.

El protocolo es basado en texto y soporta los siguientes comandos:

- STATS: consulta de métricas en tiempo real (bytes transferidos, conexiones históricas/actuales, tasa de éxito de autenticación).
- USERS: listado de los usuarios registrados actualmente en el servidor.
- ADD <user>:<pass>: alta de nuevos usuarios sin necesidad de reiniciar el servicio.
- DEL <user>: baja de usuario existente.
- PING: chequea si el servidor está “vivo”.

Se desarrolló un cliente de administración (client.c) que facilita la interacción con el servidor mediante una interfaz de línea de comandos, permitiendo a los administradores ejecutar estas operaciones de manera sencilla.

3. Problemas encontrados durante el diseño y la implementación

Durante el desarrollo, el equipo enfrentó varios desafíos técnicos:

- Manejo de dual-stack (IPv4/IPv6): configurar correctamente los sockets para aceptar conexiones tanto IPv4 como IPv6 en el mismo puerto requirió el uso de la opción `IPV6_V6ONLY` desactivada en sistemas que lo soportan, o la creación de dos sockets separados. Finalmente se optó por una estrategia híbrida que intenta primero un socket dual-stack y hace fallback a dos sockets si es necesario.
- Complejidad de la máquina de estados: dividir la lógica del protocolo en estados discretos y manejar las transiciones correctamente, especialmente durante la fase de autenticación y conexión al servidor de origen, añadió una complejidad significativa al código en comparación con un modelo secuencial/bloqueante.

4. Limitaciones de la aplicación

- Métricas volátiles: las métricas recolectadas se mantienen en memoria y se pierden al reiniciar el proceso.
- Comandos soportados: solo se soporta el comando `CONNECT`. `BIND` y `UDP ASSOCIATE` no están implementados.

5. Posibles extensiones

Para futuras versiones del proyecto, se proponen las siguientes mejoras:

- Resolución DNS asíncrona: utilizar una librería como un pool de hilos para evitar bloqueos durante la resolución de nombres.
- Persistencia de métricas: integrar un sistema de base de datos ligera (como SQLite) para persistir métricas históricas entre reinicios.
- Soporte completo SOCKS5: implementar los comandos BIND y UDP ASSOCIATE para soportar protocolos más complejos como FTP activo o streaming UDP.
- Optimización de buffers: implementar un manejo dinámico del tamaño de los buffers para optimizar el uso de memoria bajo carga extrema.
- Optimización de comandos de management: actualmente, los comandos de listar usuarios, agregar un usuario y borrar un usuario funcionan en O(n) siendo n la cantidad de usuarios de nuestro sistema. Podría implementarse una estructura de datos como un árbol AVL donde las keys son los nombres de usuario, lo que permitiría realizar las operaciones ya mencionadas en O(log n).

6. Conclusiones

El proyecto SOCKSv5-TPE ha cumplido satisfactoriamente con todos los objetivos planteados, entregando un servidor proxy funcional, concurrente y administrable. La incorporación del protocolo S5ADMIN y el sistema de logging eleva la calidad del producto final, permitiendo su operación y auditoría en entornos reales. A través de este desarrollo, el equipo ha consolidado conocimientos avanzados sobre programación de redes en C, multiplexación de E/S y diseño de protocolos de capa de aplicación.

7. Ejemplos de prueba

a. Velocidad en función del buffer_size

Para elegir el tamaño óptimo del buffer del proxy SOCKSv5, armamos un banco de pruebas reproducible. El script **scripts/benchmark_buffer.sh** compila el servidor con distintos valores de BUFFER_SIZE, genera archivos de 1 MB a 128 MB, los sirve por HTTP local y mide descargas directas y a través del proxy con curl, promediando múltiples corridas. Los resultados quedan en *buffer_benchmark.csv*. Luego, **scripts/plot_buffer_benchmark.py** toma ese CSV y genera gráficos de throughput vs. tamaño de buffer (por tamaño de archivo y promedio general).

Los gráficos muestran el comportamiento esperado: la velocidad mejora rápido al aumentar el buffer desde valores muy pequeños, pero a partir de ~128 KB–256 KB el rendimiento se estabiliza y las ganancias marginales son despreciables. De algún modo, se ve reflejada la ley de los rendimientos decrecientes. Por eso fijamos el BUFFER_SIZE por defecto en 128 KB: es lo bastante grande para evitar cuellos de botella en las transferencias y lo bastante chico para no desperdiciar memoria por conexión.

Esta medición elimina ruido de red (usa HTTP local) y permite comparar de forma objetiva el impacto de la configuración interna del servidor sobre el throughput.

Otro factor a tener en cuenta es que antes de la preentrega, teníamos un ciclo de select ineficiente que hacía select -> read -> select -> write. Gracias a los comentarios hechos en la preentrega del trabajo, hemos optimizado el flujo del selector a select -> read -> write -> select. En la figura 1, puede observarse el gráfico de buffer size vs. throughput con el ciclo optimizado. En la figura 2, puede observarse lo mismo con el flujo viejo de selector. Resulta más evidente con el nuevo gráfico que el buffer size de 128 KiB resulta óptimo para nuestro servidor.

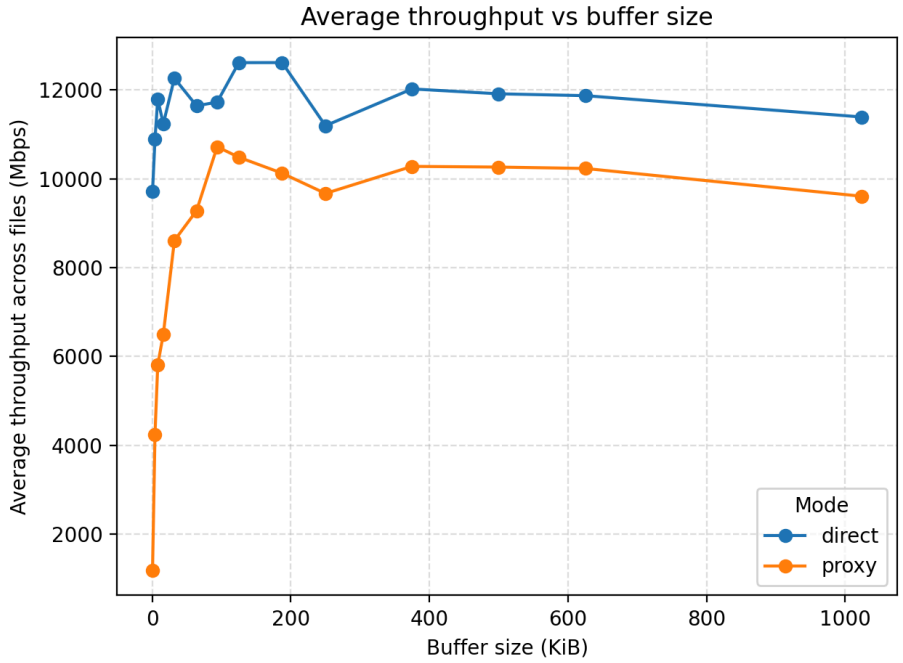


Figura 1: luego de la pre-entrega

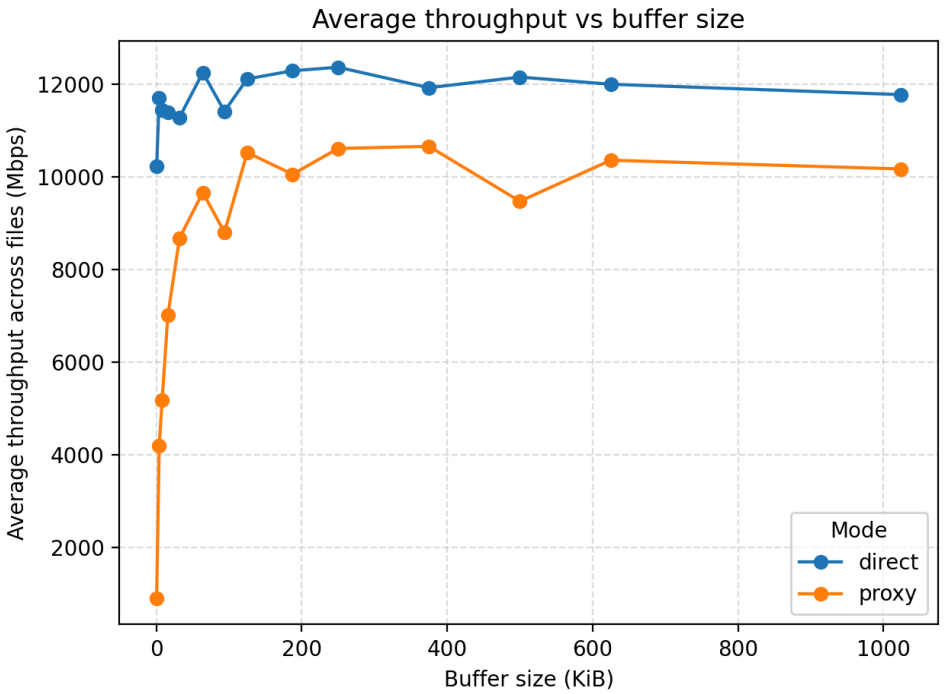


Figura 2: antes de la corrección de la pre-entrega

b. Stress test de usuarios

Utilizando herramientas de generación de carga, se verificó que el servidor es capaz de manejar más de 500 conexiones concurrentes simultáneas sin degradación

funcional, cumpliendo con el requerimiento de concurrencia. La arquitectura basada en `select`/`poll` permite escalar eficientemente hasta el límite de file descriptors del sistema operativo.

```
$> python3 ./tests/concurrency_test.py
```

c. Navegación utilizando el proxy

Se configuró un navegador web Firefox para utilizar el proxy SOCKSv5 en 127.0.0.1:1080. Se verificó la correcta carga de sitios web complejos (YouTube, Google, sitios con HTTPS). En el navegador Firefox, se accedió a *about:preferences* y se configuraron los network settings de forma tal de usar nuestro servidor proxy levantado en 127.0.0.1:1080. Para testear bien que efectivamente se estuviera usando nuestro servidor, una vez matado el mismo con Ctrl+C, el navegador dejó de funcionar. A su vez, en el access.log que se creó al matar al servidor, pudimos ver todas las request que hicimos en el navegador durante esa sesión.

8. Guía de instalación

Requisitos previos: Sistema operativo Linux/Unix, compilador GCC, `make` y `python3` (para tests).
Pasos:
1. Clonar el repositorio o descomprimir el código fuente.
2. Navegar al directorio raíz del proyecto (`SOCKSv5-TPE`).
3. Ejecutar el comando:

```
$> make clean all
```

4. El ejecutable se generará en **build/bin/socks5d**.

```
$> make run
```

```
$> ./build/bin/socks5d
```

9. Instrucciones para la configuración

a. Configuración del Servidor

El servidor se configura principalmente mediante argumentos de línea de comandos al iniciarlo. No utiliza un archivo de configuración persistente.

Opciones disponibles:

- **-l <ip>**: Dirección IP donde escuchar (default: 0.0.0.0).
- **-p** <port>: Puerto TCP donde escuchar (default: 1080).
- **-u** <user>:<pass>: Registra un usuario y contraseña para autenticación. Puede usarse múltiples veces (hasta 10 usuarios).
- **-v**: Muestra la versión y sale.
- **-h**: Muestra ayuda.

b. Configuración del cliente de administración

El cliente se encuentra en client.c. No requiere instalación, solo permisos de ejecución. Por defecto intenta conectar a 127.0.0.1:8080. Estas variables pueden sobreescribirse cambiando los “define” en el código del cliente.

10. Ejemplos de configuración y monitoreo

a. Comandos generales

Iniciar el servidor escuchando en todas las interfaces, puerto 1080, con el usuario "foo" y contraseña "bar":

```
$> ./build/bin/socks5d -u foo:bar
```

Iniciar en un puerto específico y en background:

```
$> ./build/bin/socks5d -l 127.0.0.1 -p 8888 -u admin:secret > server.log 2>&1 &
```

b. Comandos de monitoreo y gestión

Para interactuar con el servidor, utilice el script cliente incluido:
Obtener estadísticas:

```
$> ./build/bin/client STATS
```

Salida esperada: Conexiones históricas, actuales, bytes transferidos, etc.
Agregar un nuevo usuario en tiempo de ejecución:

```
$> ./build/bin/client ADD alice:secret123
```

Verificar usuarios registrados:

```
$> ./build/bin/client USERS
```

Custom Host/Port:

```
$> ./build/bin/client -L 127.0.0.1 -P 8080 STATS
```

c. Comandos de configuración

La configuración es estática tras el inicio. Para cambiar usuarios o puertos, debe reiniciar el proceso con nuevos argumentos.

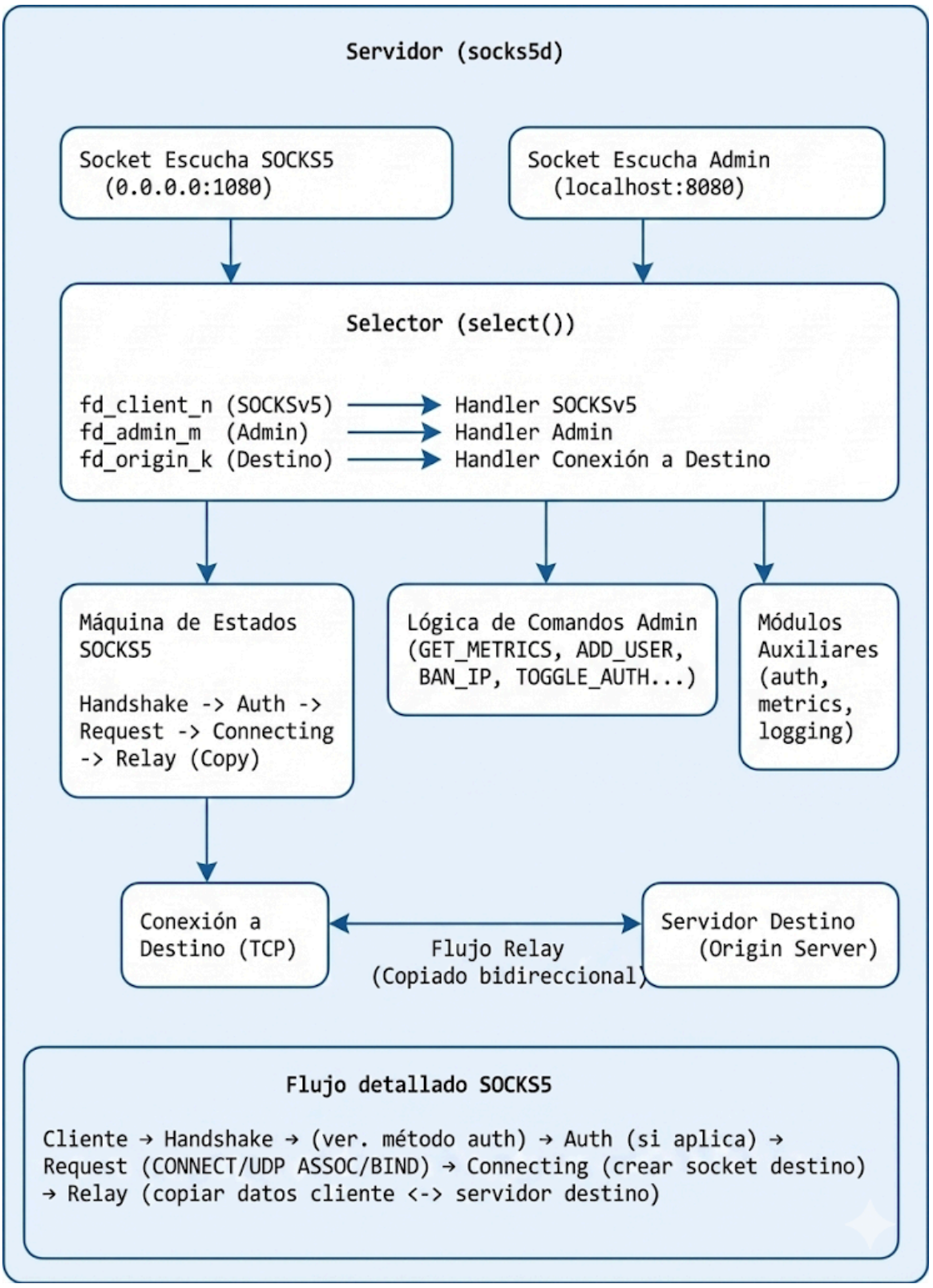
11. Documento de diseño del proyecto

a. Componentes principales

- Selector: Abstracción del bucle de eventos.
- Socks5Nio: Módulo core que integra el selector con la lógica del proxy.
- Management (S5ADMIN): Módulo encargado de procesar paquetes UDP con comandos de administración, interactuando directamente con las estructuras de métricas y usuarios del servidor.
- Logger: Subsistema centralizado para la emisión de logs de acceso y operativos.
- STM: Motor genérico de máquinas de estado.

b. Flujo de un conexión SOCKSv5

1. Aceptación: El servidor acepta una nueva conexión TCP y crea una estructura `socks5`.
2. HELLO: El servidor lee la versión y métodos de autenticación del cliente. Selecciona `USERNAME/PASSWORD` (0x02) si hay usuarios definidos, o `NO AUTH` (0x00).
3. AUTH: Si se requiere, el servidor valida las credenciales enviadas por el cliente.
4. REQUEST: El servidor lee el comando (CONNECT) y la dirección de destino (IPv4, IPv6 o Dominio).
5. RESOLUCIÓN Y CONEXIÓN: Si es un dominio, se resuelve la IP. Luego, el servidor inicia una conexión no bloqueante hacia el destino.
6. COPY: Una vez establecida la conexión con el destino, el servidor entra en un bucle de copia bidireccional, moviendo bytes entre el cliente y el servidor de origen hasta que uno de los dos cierre la conexión.


Servidor (socks5d)
Socket Escucha SOCKS5 (0.0.0.0:1080)
Socket Escucha Admin (localhost:8080)
Selector (select())
fd_client_n (SOCKSv5) → Handler SOCKSv5
fd_admin_m (Admin) → Handler Admin
fd_origin_k (Destino) → Handler Conexión a Destino
Máquina de Estados SOCKS5
Handshake -> Auth -> Request -> Connecting -> Relay (Copy)
Lógica de Comandos Admin (GET_METRICS, ADD_USER, BAN_IP, TOGGLE_AUTH...)
Módulos Auxiliares (auth, metrics, logging)
Conexión a Destino (TCP)
Flujo Relay (Copiado bidireccional)
Servidor Destino (Origin Server)
Flujo detallado SOCKS5
Cliente → Handshake → (ver. método auth) → Auth (si aplica) → Request (CONNECT/UDP ASSOC/BIND) → Connecting (crear socket destino) → Relay (copiar datos cliente <-> servidor destino)