Protocolos de Comunicación

Trabajo Práctico Especial

2025 2Q

Informe

Grupo 15

Integrantes:

- Alexis Herrera Vegas, 64045
- Tomas Jeronimo Esquivel, 64756
- Andrés Cortese, 64612
- Sebastian Caules, 64331

December 11, 2025

Contents

1 Descripción de protocolos y aplicaciones

1.1 Servidor Proxy SOCKS5

El servidor implementa el protocolo SOCKS5 (RFC 1928) con soporte para autenticación usuario/contraseña (RFC 1929). Acepta conexiones de clientes en un puerto configurable (default: 1080) y establece túneles TCP hacia destinos especificados por IPv4, IPv6 o FQDN.

1.2 Protocolo de Administración

Se desarrolló un protocolo binario propietario para administración remota del servidor. Opera en un puerto configurable (default: 8080, configurable con `-P`) y dirección configurable (default: 127.0.0.1, configurable con `-L`). Requiere autenticación previa y distingue entre usuarios con rol "admin" y "user".

1.2.1 Comandos disponibles

Comandos para todos los usuarios:

- **GET_METRICS**: Obtiene métricas del servidor (conexiones totales, actuales, bytes transferidos, uptime)
- **LIST_USERS**: Lista todos los usuarios registrados con sus estadísticas
- **LIST_CONNECTIONS**: Muestra el registro de las últimas 1000 conexiones

Comandos solo para administradores:

- **ADD_USER**: Crea un nuevo usuario con rol "user" por defecto
- **DEL_USER**: Marca un usuario como inactivo
- **CHANGE_PASSWORD**: Modifica la contraseña de un usuario
- **CHANGE_ROLE**: Cambia el rol entre "admin" y "user"

2 Problemas encontrados durante el diseño y la implementación

Una dificultad clave fue descomponer SOCKS5 en fases bien delimitadas (handshake, autenticación RFC 1929 y request) sobre I/O no bloqueante. Esto obligó a diseñar una máquina de estados explícita y parsers que procesan byte a byte las variantes de dirección (IPv4, IPv6 o dominio) y los códigos de respuesta. El manejo de longitudes variables en el request (dominio con length-prefixed) requirió controles de límites y buffers estáticos para evitar lecturas fuera de rango o respuestas mal formadas.

Otra complejidad surgió al combinar el modelo no bloqueante con servicios auxiliares. La resolución DNS se delega a un hilo con comunicación por pipe hacia el selector principal;

sincronizar las notificaciones sin bloquear el lazo de select() demandó coordinar señales y registro de file descriptors auxiliares. Del mismo modo, la gestión en memoria de usuarios y métricas exigió proteger estructuras compartidas con mutex, ya que tanto el flujo SOCKS como el canal de administración pueden acceder y actualizar contadores o credenciales en paralelo.

3 Limitaciones de la aplicación

3.1 Usuarios volátiles

La aplicación gestiona los usuarios únicamente en memoria durante el tiempo de ejecución. Como consecuencia, cada vez que el servidor se reinicia, toda la información vinculada a los usuarios se pierde. Esto afecta tanto a los perfiles creados como a cualquier modificación realizada durante la sesión de ejecución, requiriendo reconstruir la información desde cero tras cada reinicio.

3.2 Máximo de 100 usuarios registrados

Se estableció un límite de hasta 100 usuarios registrados en la base de datos en memoria (`MAX_USERS_DB = 100`). Adicionalmente, solo se pueden agregar hasta 10 usuarios desde la línea de comandos al iniciar el servidor (`MAX_USERS = 10`). Esta restricción condiciona la escalabilidad de la solución y podría requerir ajustes en escenarios donde se proyecte una cantidad mayor de usuarios.

3.3 Logs limitados a las últimas 1000 conexiones

La aplicación mantiene un registro circular acotado a las 1000 conexiones más recientes (`MAX_CONNECTION_LOG = 1000`). Una vez alcanzado este límite, las entradas más antiguas se sobrescriben para permitir el almacenamiento de nuevas conexiones. Esta política de rotación puede dificultar ciertos análisis históricos o diagnósticos que dependan de información previa al límite establecido.

3.4 Falta de persistencia de métricas

Las métricas generadas por el servidor solo se conservan durante la ejecución actual. Tras un reinicio, todos los datos relativos a monitoreo, estadísticas o cualquier tipo de métrica interna se reinician, lo que impide mantener un historial persistente que permita evaluaciones del comportamiento del sistema en diferentes ejecuciones.

3.5 Ausencia de cifrado de credenciales

Las credenciales de los usuarios (nombre de usuario y contraseña) se transmiten sin ningún mecanismo de cifrado ni protección adicional en el protocolo SOCKS5. Esto implica que la información sensible circula en texto plano, lo cual representa un riesgo de seguridad considerable, especialmente en entornos donde se requiera protección contra interceptación o accesos no autorizados.

4 Posibles extensiones

4.1 Persistencia y respaldo

Guardar usuarios, contraseñas, métricas y registros de conexiones en almacenamiento duradero (por ejemplo, archivos o una base embebida) para conservar el estado tras reinicios y permitir respaldo o restauración cuando sea necesario.

4.2 Seguridad de credenciales

Incorporar TLS en el canal de administración y en el proxy SOCKS5, y almacenar contraseñas utilizando hashing robusto. Esto evitaría el envío en texto plano y reduciría el riesgo ante compromisos de memoria o ataques en la red.

4.3 Protocolos adicionales

Implementar los comandos SOCKS5 BIND y UDP ASSOCIATE para ampliar el soporte más allá del comando CONNECT actualmente disponible.

4.4 Control de acceso y límites

Agregar listas de control de acceso basadas en IP, FQDN o puerto, junto con mecanismos de limitación de tasa por usuario o cliente. Incluir bloqueos temporales ante múltiples intentos de autenticación fallidos y cuotas sobre conexiones o ancho de banda utilizado.

4.5 Configuración ampliada

Permitir configurar el host y puerto del servicio de administración mediante argumentos de línea de comandos (`-L` y `-P`, ya implementados), junto con otras opciones (por ejemplo, límites de usuarios o tamaño del archivo de registros), con posibilidad de recarga en caliente sin reiniciar el servidor.

4.6 Observabilidad persistente

Exportar métricas a un endpoint consumible en formatos estándar, como Prometheus, y ofrecer paginación en el cliente de administración para recorrer el historial completo de conexiones sin truncamiento.

5 Conclusiones

El desarrollo del trabajo práctico, aunque demandante, resultó una instancia valiosa para poner en práctica los contenidos vistos a lo largo del cuatrimestre. La implementación nos permitió trabajar con una arquitectura cliente-servidor concreta basada en SOCKS5 y profundizar en cada una de sus fases operativas, incluyendo el establecimiento de la conexión, la autenticación y el procesamiento de las solicitudes de los clientes. Además, la incorporación del protocolo de administración nos llevó a analizar cómo ampliar un

sistema ya existente para sumar capacidades de control y supervisión. A lo largo del trabajo, obtuvimos nuevos aprendizajes como la programación con sockets, la construcción de parsers orientados a estados y la administración de múltiples conexiones mediante un selector no bloqueante.

6 Ejemplos de prueba

Se desarrollaron tests de estrés en C para evaluar el rendimiento del servidor bajo diferentes condiciones de carga. Los tests implementados fueron:

6.1 Test de Máximas Conexiones Concurrentes

Este test abre conexiones SOCKS5 completas (con handshake, autenticación y CONNECT) sin cerrarlas, incrementando hasta que el servidor o el sistema operativo rechace nuevas conexiones.

Resultado: El servidor alcanzó 508 conexiones simultáneas antes de alcanzar el límite del sistema operativo ("Too many open files").

6.2 Test de Throughput

Mide cómo varía el throughput (requests/segundo) con diferentes cantidades de conexiones concurrentes, usando un ramp-up gradual de 10ms entre inicios de threads y máximo 50 threads concurrentes para evitar saturación artificial.

Resultados:

Conexiones	Tiempo (s)	Throughput (req/s)	Exitosas	Tasa Éxito
10	0.17	59.81	10	100%
25	0.37	68.37	25	100%
50	0.65	76.87	50	100%
75	0.95	79.29	75	100%
100	1.24	80.36	100	100%
150	1.86	80.82	150	100%
200	2.50	80.05	200	100%
250	3.06	81.73	250	100%
300	3.64	82.31	300	100%
400	4.89	81.85	400	100%
500	6.06	82.53	500	100%

Table 1: Resultados del test de throughput

Observaciones: El throughput se estabiliza en aproximadamente 80-82 req/s a partir de 75 conexiones concurrentes, manteniendo 100% de tasa de éxito en todas las pruebas hasta 500 conexiones. Destaca especialmente el rendimiento con 500 conexiones alcanzando 82.53 req/s, lo que indica que el servidor mantiene excelente performance bajo alta carga.

6.3 Test de Latencia

Ejecuta 100 conexiones secuenciales midiendo el tiempo desde el socket connect hasta completar el handshake SOCKS5, autenticación y CONNECT request.

Resultados:

- Latencia promedio: 19.88 ms
- Latencia mediana: 18.67 ms
- Percentil 95 (P95): 27.50 ms
- Percentil 99 (P99): 46.09 ms
- Desviación estándar: 4.82 ms
- Tasa de éxito: 100%

Conclusión: La baja latencia promedio (menor a 20 ms) y la poca variabilidad (desv. est. menor a 5 ms) indican que el servidor procesa requests de manera eficiente y consistente.

7 Guía de compilación

Para compilar el proyecto es necesario contar con `gcc` y `Make` previamente instalados. Desde la raíz del proyecto, ejecute:

```
make clean
make all
```

Estos comandos generarán dos binarios en el directorio actual:

- `socks5d` — Servidor proxy SOCKS5
- `admin-client` — Cliente de administración

7.1 Ejecución del servidor

El servidor SOCKS5 se ejecuta desde la raíz del proyecto mediante el siguiente comando:

```
./socks5d [OPCIONES]
```

7.2 Opciones de línea de comandos

- `-h`: Mostrar ayuda y terminar
- `-l <dirección>`: Dirección de escucha del proxy SOCKS (por defecto: 0.0.0.0). Para habilitar modo dual-stack IPv6 utilice `::`
- `-L <dirección>`: Dirección del servicio de administración (por defecto: 127.0.0.1)

- `-p <puerto>`: Puerto SOCKS5 (por defecto: 1080)
- `-P <puerto>`: Puerto del servicio de administración (por defecto: 8080)
- `-u <nombre>:<contraseña>`: Agregar usuario en línea de comandos (máximo 10)
- `-v`: Mostrar versión y terminar

8 Instrucciones para la configuración

Antes de utilizar el servidor SOCKS5 y el cliente de administración, es necesario realizar una configuración mínima basada en los parámetros proporcionados por la aplicación. El servidor se ejecuta especificando la dirección de escucha y el puerto del servicio mediante las opciones de línea de comandos. Por defecto, escucha en 0.0.0.0 y utiliza el puerto 1080 para el tráfico SOCKS5, aunque es posible habilitar modo dual-stack asignando la dirección `::`.

El servidor expone un puerto configurable para el protocolo de administración (por defecto 127.0.0.1:8080, configurable con `-L` y `-P`). Al iniciar, se crea automáticamente un usuario administrador con credenciales por defecto (`admin` / `1234`), que deberá utilizarse para las tareas iniciales de gestión. Es posible agregar hasta 10 usuarios adicionales desde la línea de comandos usando la opción `-u`.

El cliente de administración requiere especificar, como mínimo, el usuario y la contraseña para autenticarse contra el servidor. Opcionalmente es posible redefinir la dirección y el puerto del servicio de administración mediante los flags `-l` y `-p`. Las funciones disponibles dependen del rol del usuario autenticado: los administradores pueden gestionar usuarios, roles y contraseñas, mientras que los usuarios estándar sólo pueden consultar información.

9 Ejemplos de configuración y monitoreo

9.1 Iniciar el servidor

Para iniciar el servidor con parámetros explícitos en IPv4:

```
./socks5d -l 0.0.0.0 -p 1080 -P 8080
```

Si se desea ejecutar en modo dual-stack con soporte para IPv6:

```
./socks5d -l :: -p 1080
```

Para agregar usuarios desde la línea de comandos:

```
./socks5d -u admin:1234 -u alice:secret -u bob:pass123
```

9.2 Monitoreo y administración

Para consultar las métricas del servidor:

```
./admin-client -u admin -P 1234 metrics
```

Para obtener la lista de usuarios registrados:

```
./admin-client -u admin -P 1234 users
```

Para visualizar las últimas conexiones registradas:

```
./admin-client -u admin -P 1234 conns
```

9.3 Gestión de usuarios

Para agregar un nuevo usuario:

```
./admin-client -u admin -P 1234 add juan password123
```

Para cambiar el rol de un usuario:

```
./admin-client -u admin -P 1234 change-role juan admin
```

9.4 Verificación del proxy

Para verificar el funcionamiento del proxy SOCKS5 mediante curl:

```
curl --proxy socks5://admin:1234@localhost:1080 http://google.com
```

Para resolución DNS remota:

```
curl --proxy socks5h://admin:1234@localhost:1080 http://google.com
```

10 Diseño del proyecto

El diseño se basa en una arquitectura cliente-servidor que implementa SOCKS5 con autenticación RFC 1929 y el comando CONNECT. El servidor (`socks5d`) actúa como proxy TCP, procesa handshake, autenticación y solicitud en una máquina de estados y redirige tráfico al destino. Usa un selector propio sobre `select()` con descriptores no bloqueantes, evitando hilos por conexión y permitiendo múltiples clientes en paralelo. Las métricas globales (conexiones totales/activas y bytes) se actualizan a lo largo del flujo.

La implementación se organiza en módulos separados: `socks5/` (handshake, auth, request y copia), `auth/` (validación de credenciales), `users/` (usuarios en memoria con roles admin/usuario y log circular de 1000 conexiones), `metrics/` (contadores), `dns/` (resolución asíncrona en un hilo con notificación por pipe al selector) y `utils/` (selector, buffers y máquina de estados). El parser de solicitudes por estado valida versión, comando y tipo de dirección (IPv4, IPv6 o dominio) y construye la respuesta según el código SOCKS5.

Se desarrolló un protocolo de administración propio en un socket TCP con puerto y dirección configurables (default: 127.0.0.1:8080) que requiere autenticación y expone comandos para métricas, usuarios y registro de conexiones, además de altas, bajas y cambios de rol/contraseña. Un cliente de administración (`admin-client`) en C consume este protocolo y ofrece interfaz CLI.

11 Decisiones de implementación

11.1 Almacenamiento de usuarios y log de accesos

Las credenciales y roles se guardan en un vector estático en memoria (`MAX_USERS_DB=100`), protegido por mutex. Al iniciar se crea el admin por defecto (`admin/1234`) y se cargan hasta 10 usuarios adicionales desde la línea de comandos (`MAX_USERS=10`). No hay persistencia: todo se pierde al reiniciar. El registro de conexiones también es estático y circular (`MAX_CONNECTION_LOG=1000`), lo que simplifica la gestión y evita alocaciones dinámicas durante la operación.

11.2 Alcance de comandos SOCKS5

En la fase de request se implementa únicamente el comando CONNECT. BIND y UDP ASSOCIATE no se procesan; cualquier comando distinto se rechaza con el código de error correspondiente. Esto reduce la complejidad de estados y manejo de sockets, manteniendo el caso de uso principal (túnel TCP) cubierto.

11.3 Tamaños fijos de buffers y datos

Se emplean buffers de tamaño fijo para tráfico (`BUFFER_SIZE=8192` bytes para cada dirección) y límites estáticos en campos (`MAX_USERNAME=256`, `MAX_PASSWORD=256`). Esta elección evita `malloc/free` en el camino de datos y facilita controlar errores de lectura/escritura parcial, a costa de rigidez ante credenciales o registros más largos.

11.4 Multiplexación no bloqueante y DNS asíncrono

Todo el I/O del proxy y del canal de administración se maneja con un selector propio sobre `select()` y descriptores no bloqueantes. El selector se inicializa con capacidad para 1024 conexiones (`selector_new(1024)`). Para no bloquear en resoluciones DNS, se usa un hilo trabajador que procesa una cola acotada (`MAX_QUEUE_SIZE=100`) y devuelve resultados por un pipe registrado en el selector; así el loop principal sigue atendiendo clientes.

11.5 Alcance y seguridad del canal de administración

El servicio de administración escucha en dirección y puerto configurables (default: 127.0.0.1:8080, configurable con `-L` y `-P`) y requiere autenticación antes de cualquier comando. Los mensajes son binarios con cabeceras de versión, comando y longitud; los comandos de mutación (`add/del/change-password/change-role`) exigen rol admin. Mantenerlo en localhost por defecto reduce la exposición.

11.6 Sincronización y concurrencia

Todas las estructuras compartidas (usuarios, métricas, log de conexiones) están protegidas por mutex. El servidor SOCKS5 y el canal de administración operan sobre el mismo selector, permitiendo gestión concurrente sin bloqueos. La cola del resolver DNS también

está protegida por mutex y condition variable para coordinación entre el thread worker y el thread principal.