



Instituto Tecnológico de Buenos Aires

Trabajo Práctico Especial

72.07 - Protocolos de Comunicación

Grupo 11

Integrantes:

Nacuzzi, Gianna Lucia - 64006

Squillari, Maria Agostina - 64047

Frutos, Pilar - 64225

Carrica, Florencia - 62040

Docentes:

Sebastian Kulesz

Thomas Mizrahi

Hugo Javier Stupenengo Faus

Índice

Descripción detallada de los protocolos y aplicaciones desarrolladas	2
Problemas encontrados durante el diseño y la implementación	2
Limitaciones de la aplicación	3
Posibles extensiones	3
Conclusiones	4
Ejemplos de prueba	4
Guía de instalación detallada y precisa	7
Instrucciones para la configuración	7
Ejemplos de configuración y monitoreo	8
Documento de diseño del proyecto	9

Descripción detallada de los protocolos y aplicaciones desarrolladas

En el presente trabajo se desarrolló un servidor proxy conforme a la especificación SOCKS versión 5, siguiendo el RFC 1928. El servidor implementa el flujo completo de negociación, autenticación y reenvío de conexiones. La autenticación se realiza exclusivamente mediante el método de usuario y contraseña. Esta decisión de diseño se fundamenta en la necesidad de asociar cada conexión a un usuario específico, requisito indispensable para la correcta operación del sistema de métricas.

Adicionalmente, se diseñó e implementó un protocolo propio, denominado METP, orientado a la gestión y monitoreo en tiempo real del servidor SOCKS5. METP es un protocolo basado en texto, inspirado en el estilo de los protocolos tipo RFC, y permite la interacción administrativa a través de un canal TCP independiente. Entre sus funcionalidades se incluyen la consulta de métricas de uso (conexiones históricas, conexiones activas, bytes transferidos), la gestión de usuarios (alta, baja, cambio de rol), la consulta de logs de acceso y la modificación dinámica de parámetros de configuración, como el tamaño de los buffers.

El protocolo METP fue especificado y documentado en un archivo de texto con formato similar a un RFC, incluido en la raíz del proyecto, donde se detallan los comandos soportados, los códigos de respuesta, los flujos de autenticación y los permisos requeridos para cada operación. La implementación de METP en el servidor garantiza que todas las operaciones administrativas y de monitoreo puedan realizarse en tiempo de ejecución, sin necesidad de reiniciar el servicio ni interrumpir las conexiones de los clientes SOCKS5.

Ambas aplicaciones, el servidor SOCKS5 y el cliente de administración METP, fueron desarrolladas en lenguaje C. El servidor soporta múltiples conexiones concurrentes mediante un modelo de I/O no bloqueante y un loop basado en select, lo que permite escalar a un gran número de clientes. El cliente METP, por su parte, permite a los administradores interactuar con el servidor de manera sencilla, autenticándose y ejecutando comandos de gestión o consulta de métricas según los permisos asignados.

En resumen, el sistema desarrollado no solo cumple con los requisitos funcionales de un proxy SOCKS5 seguro y eficiente, sino que también incorpora capacidades avanzadas de monitoreo y administración en tiempo real, facilitando la operación y el control del servicio en entornos de producción.

Problemas encontrados durante el diseño y la implementación

Se encontraron una variedad de problemas a lo largo del proyecto. En primer lugar, una vez que se logró que el servidor SOCKS acepte la autenticación de usuario contraseña se lograba la conexión de parte del cliente pero no se podía completar la conexión. Una vez ajustados los parámetros de selector y buffer se logró reutilizar la función `on_request_forward_write` para permitirle al cliente la conexión.

Además, se hallaron problemas al implementar el protocolo de métricas. El primer inconveniente surgió al intentar diseñar el protocolo ya que al principio se decidió ir por un

protocolo binario así se asemejaba más a SOCKS pero para poder ofrecer comodidad al usuario se optó por cambiar a un protocolo basado en texto lo cual hizo más fácil su implementación.

Limitaciones de la aplicación

Autenticación

La autenticación implementada en el servidor SOCKS5 se limita exclusivamente al método de usuario y contraseña, conforme al RFC 1929. No se encuentran soportados otros mecanismos de autenticación.

Protocolos soportados

El proxy implementa únicamente el reenvío de conexiones TCP, en línea con el comando CONNECT del protocolo SOCKS5. No se encuentra implementado el soporte para el comando UDP ASSOCIATE, por lo que no es posible utilizar el proxy para aplicaciones que requieran reenvío de tráfico UDP.

Gestión de usuarios en METP

El protocolo de administración y monitoreo METP permite la creación, eliminación y cambio de rol de usuarios, pero no contempla la edición de credenciales de usuarios existentes (por ejemplo, cambio de contraseña). Para modificar la contraseña de un usuario, es necesario eliminarlo y crearlo nuevamente con la nueva clave.

Escalabilidad y concurrencia

El servidor utiliza la llamada al sistema select() para la multiplexación de conexiones, lo que impone un límite máximo de 1024 file descriptors abiertos simultáneamente. Dado que cada conexión SOCKS5 completamente establecida utiliza dos file descriptors (uno para el cliente y otro para el destino), el número máximo de conexiones concurrentes se ve limitado en consecuencia. Este límite puede variar según el sistema operativo y la configuración del entorno de ejecución.

Métricas y logs

Las métricas y logs de acceso se mantienen en memoria y tienen un tamaño máximo predefinido. En caso de alcanzar el límite, las entradas más antiguas se sobrescriben.

Posibles extensiones

Por un lado, la implementación actual se limita al método de autenticación usuario/contraseña. Una extensión natural sería implementar los métodos de autenticación restantes definidos en el RFC 1928.

Por otro lado, se podría extender el diseño del protocolo METP en una versión posterior. Permitiendo otros cambios en ejecución y nuevas métricas o configuraciones. También se podría implementar métricas persistentes.

Otra posible extensión sería la implementación del comando UDP ASSOCIATE del protocolo SOCKS5, permitiendo el reenvío de tráfico UDP a través del proxy. Esto ampliará el rango de casos de uso del sistema.

Finalmente, podríamos extender la cantidad de conexiones simultáneas haciendo uso de epoll en lugar del select que es bastante limitado.

Conclusiones

Se concluye que se implementó un servidor a la altura de lo pedido, es no bloqueante y logrando conexiones de más de 500 clientes al mismo tiempo. Además, se logró diseñar y desarrollar un protocolo de monitoreo sencillo de usar.

Ejemplos de prueba

Se ve como usar el servidor. Se arranca el servidor con un usuario admin.

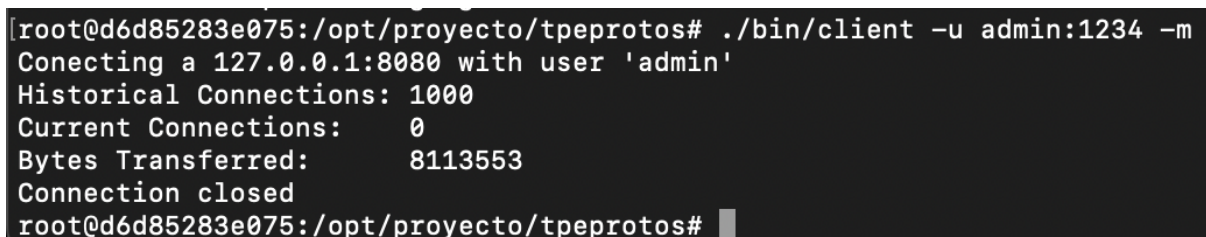
```
PROXY_CTRL_SUPERUSER=admin:1234 ./bin/socks5d
```

Se realiza una conexión a un servidor HTTP de prueba:

```
for i in {1..1000}; do curl -s -o /dev/null -x socks5h://admin:1234@localhost:1080 https://www.google.com & done; wait
```

Se puede observar como las conexiones fueron exitosas:

```
./bin/client -u admin:1234 -m
```



```
[root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -m
Conecting a 127.0.0.1:8080 with user 'admin'
Historical Connections: 1000
Current Connections:    0
Bytes Transferred:     8113553
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos#
```

Pruebas de estrés

Con el objetivo de evaluar el rendimiento, la escalabilidad y la disponibilidad del servidor proxy SOCKS5, se llevaron a cabo pruebas de estrés en un entorno controlado. Para ello, se ajustó previamente la configuración del sistema operativo cliente, incrementando el límite de descriptores de archivos abiertos mediante la instrucción `ulimit -n 4096`. Este cambio permitió establecer un mayor volumen de conexiones concurrentes, condición necesaria para ejecutar pruebas con hasta 500 conexiones simultáneas.

La generación de carga se realizó utilizando la herramienta curl, combinada con xargs, para simular múltiples clientes accediendo al proxy de forma concurrente. En particular, se ejecutaron 1000 solicitudes paralelas hacia el servidor utilizando el siguiente comando:

```
seq 1 1000 | xargs -n1 -P1000 curl \  
-x socks5h://localhost:1080 \  
-U admin:123 \  
-s -o /dev/null \  
http://google.com/ &
```

Cada una de estas instancias de curl estableció una conexión independiente, autenticándose con el usuario admin y la contraseña 123.

Vemos que luego de ejecutarlo tenemos:

```
./bin/client -u admin:123 -m  
Conecting a 127.0.0.1:8080 with user 'admin'  
Historical Connections: 938  
Current Connections: 713  
Bytes Transferred: 691152  
Connection closed
```

Lo que coincide con el establecido pues refleja exactamente los límites impuestos por nuestra implementación y el sistema operativo: cada sesión SOCKS5 registrada consume al menos un descriptor de archivo (y, una vez abierto el túnel remoto, llega a consumir dos), el servidor está dimensionado con `MAX_SELECTOR_FDS = 1024` y usa `select()`, que no puede manejar más de ese número de ficheros. Teniendo en cuenta que cada conexión ocupa dos fd, el resultado teórico es 512. Sin embargo, durante el arranque y la fase de negociación SOCKS5, muchas conexiones sólo ocupan un fds hasta que abran el remoto, lo que permite excederse ligeramente esa mitad teórica hasta las 713 conexiones que vemos.

Además de esta prueba de concurrencia, se implementó una segunda prueba más orientada a evaluar el comportamiento del sistema bajo carga continua. Para ello, se diseñó un script en Bash que ejecutó 500 solicitudes al proxy con breves intervalos de espera entre cada una. El script consiste en un bucle que, en cada iteración, lanzaba una nueva conexión utilizando curl, con una pausa de 0.1 segundos entre llamadas. Esto permitió simular una demanda sostenida a lo largo del tiempo. El script es el siguiente:

```
#!/bin/bash  
for i in $(seq 1 500); do  
  curl -x socks5h://localhost:1080 -U admin:123 http://google.com/ &  
  sleep 0.1  
done  
echo "Carga generada."
```

Durante la ejecución de ambas pruebas, se monitorearon en tiempo real las métricas internas del servidor mediante el protocolo METP, utilizando el cliente (`./bin/client -u admin:password123 -m`). Las métricas se pueden visualizar en frecuencia de 1 segundo usando el siguiente script mientras se corre el anterior:

```
#!/usr/bin/env bash
prev_bytes=0
prev_time=$(date +%s)

echo "Timestamp          Throughput (B/s)"
echo "-----"

while true; do
    now=$(date +%s)
    bytes=$(
        ./bin/client -u admin:123 -m 2>/dev/null \
        | grep 'Bytes Transferred' \
        | awk '{print $3}'
    )
    dt=$((now - prev_time))
    dbytes=$((bytes - prev_bytes))

    if [ $dt -gt 0 ]; then
        throughput=$((dbytes / dt))
    else
        throughput=0
    fi

    printf "%s    %10d\n" "$(date '+%Y-%m-%d %H:%M:%S')" "$throughput"

    prev_bytes=$bytes
    prev_time=$now
    sleep 2
done
```

Durante la prueba de carga continua, se midió el throughput del proxy registrando cada dos segundos el total de bytes transferidos y calculando la tasa instantánea. Al inicio el throughput fue nulo mientras se establecen las primeras conexiones; a los pocos segundos alcanzó ya 764 B/s y luego se estabilizó en torno a los 11 000–13 000 B/s. También se observó una breve caída a 7 878 B/s, coincidiendo con un pico de cierre de conexiones iniciales, aunque rápidamente volvió a valores superiores a 12 000 B/s. Estos resultados

muestran que, el proxy mantiene de forma sostenida un throughput aproximado de 12 000 B/s bajo un ritmo de 500 peticiones cada 0,1 s.

Por último, se supervisó el uso de recursos del proceso SOCKS5D a través de la herramienta htop. Durante toda la prueba, el uso de CPU se mantuvo cercano al 8%, mientras que el consumo de memoria se estabilizó en torno a los 140 MB. Estos valores reflejan una gestión eficiente de los recursos, lo cual demuestra que el servidor es capaz de operar bajo alta demanda con un nivel de consumo razonable, manteniendo su capacidad de respuesta y estabilidad.

Guía de instalación detallada y precisa.

- Correr desde la carpeta raíz del proyecto el comando make para compilar el servidor y el cliente.
- Los binarios se encontrarán en la subcarpeta bin con el nombre socks5d para los servidores socks5 y metp y client para el cliente de metp.
- Ejecutar los binarios con los flags deseados. Las opciones disponibles se pueden ver con el flag -h.

Instrucciones para la configuración

Para configurar y poner en funcionamiento el proxy SOCKS5 desarrollado en este proyecto, es necesario seguir los siguientes pasos:

1. Configuración inicial y ejecución del servidor

El servidor SOCKS5 se configura mediante los flags que se pasan al ejecutable al momento de su ejecución. Las opciones disponibles están documentadas tanto en la página de manual (socks5d.8) como en la ayuda accesible con el flag -h del binario (./bin/socks5d -h).

Es importante destacar que, por razones de seguridad, el servidor no crea usuarios administradores por defecto. Al menos un usuario con rol de administrador debe ser especificado al iniciar el servidor, mediante la variable de entorno PROXY_CTRL_SUPERUSER.

```
PROXY_CTRL_SUPERUSER=admin:1234 ./bin/socks5d
```

2. Gestión y configuración dinámica mediante METP

Una vez que el servidor está en ejecución, es posible acceder a funcionalidades avanzadas de monitoreo y administración a través del protocolo METP. Para facilitar la interacción con este protocolo, se provee un cliente de línea de comandos (./bin/client), cuya ayuda puede consultarse con el flag -h.

Para ejecutar comandos de administración, el usuario debe autenticarse con credenciales de administrador válidas. Si no se proporciona un usuario con permisos suficientes, el servidor denegará las operaciones de gestión y solo permitirá la consulta de métricas básicas.

3. Notas adicionales

Todos los parámetros de configuración y administración pueden consultarse en detalle en el archivo README.md y en la documentación del protocolo METP (src/client/protocol_mctp.txt).

Ejemplos de configuración y monitoreo

Se ve como usar el servidor. Se arranca el servidor con un usuario admin.

```
root@d6d85283e075:/opt/proyecto/tpeprotos# PROXY_CTRL_SUPERUSER=admin:1234 ./bin/socks5d
```

Se puede observar el uso de los comandos brindados en el cliente. Agregar usuarios usando -a, listar los usuarios con -l, eliminar usuarios con el uso de -d y por último imprimir los accesos y métricas con -g y -m.

```
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -a user user
Connecting a 127.0.0.1:8080 with user 'admin'
Operation successful
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -l
Connecting a 127.0.0.1:8080 with user 'admin'
admin admin
user user
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -d user
Connecting a 127.0.0.1:8080 with user 'admin'
Operation successful
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -l
Connecting a 127.0.0.1:8080 with user 'admin'
admin admin
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -g
Connecting a 127.0.0.1:8080 with user 'admin'
[2025-07-14T20:42:20Z] admin 127.0.0.1 google.com:80 846
[2025-07-14T20:42:21Z] admin 127.0.0.1 google.com:80 846
[2025-07-14T20:42:22Z] admin 127.0.0.1 google.com:80 846
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -m
Connecting a 127.0.0.1:8080 with user 'admin'
Historical Connections: 3
Current Connections: 0
Bytes Transferred: 2538
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos#
```

También se puede observar el uso de los roles y el error si no se está autorizado. Con -r se puede cambiar el rol de un usuario.

```
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -a user user
Connecting a 127.0.0.1:8080 with user 'admin'
Operation successful
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -l
Connecting a 127.0.0.1:8080 with user 'admin'
admin admin
user user
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -a gnacuzzi gnacuzzi
Connecting a 127.0.0.1:8080 with user 'admin'
Operation successful
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -l
Connecting a 127.0.0.1:8080 with user 'admin'
admin admin
user user
gnacuzzi user
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -r user admin
Connecting a 127.0.0.1:8080 with user 'admin'
Operation successful
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u admin:1234 -l
Connecting a 127.0.0.1:8080 with user 'admin'
admin admin
user admin
gnacuzzi user
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u user:user -g
Connecting a 127.0.0.1:8080 with user 'user'
[2025-07-14T20:42:20Z] admin 127.0.0.1 google.com:80 846
[2025-07-14T20:42:21Z] admin 127.0.0.1 google.com:80 846
[2025-07-14T20:42:22Z] admin 127.0.0.1 google.com:80 846
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos# ./bin/client -u gnacuzzi:gnacuzzi -g
Connecting a 127.0.0.1:8080 with user 'gnacuzzi'
Not authorized to execute command
Connection closed
root@d6d85283e075:/opt/proyecto/tpeprotos#
```

Documento de diseño del proyecto

Estructura de carpetas

- src/server/: Código fuente del servidor SOCKS5 y del protocolo METP.
- src/client/: Código fuente del cliente METP y documentación del protocolo.
- src/utls/: Utilidades compartidas (parsers, buffers, selector, etc.).
- bin/: Ejecutables generados tras la compilación.

Descripción de módulos en detalle

- src/server/socks5.c
 - Implementa handshake, autenticación y reenvío TCP.
 - Usa buffer.c para gestionar lecturas/escrituras fragmentadas.
- src/server/metp/metp.c
 - Implementa el flujo del servidor metp
- src/server/metp/metrics.c
 - Implementa el manejo de métricas.
- src/server/metp/users.c
 - Implementa el manejo de usuarios.
- src/utls
 - Aquí se encuentran todos los archivos brindados por la cátedra.
- src/client

- Implementa lo que sería una conexión al servidor metp como cliente, tiene la lógica para poder usar el protocolo de una forma sencilla y más rápida.