

Trabajo Práctico Especial - 'Diseño e implementación de un servidor proxy para el protocolo SOCKSv5 [RFC1928]'

Alumnos:

De Simone, Franco

61100

Dizenhaus, Manuel

61101

Mattiussi, Agustín Hernán

61361

Sasso, Julián Martín

61535

Índice

1. Introducción	2
2. Descripción de los protocolos y aplicaciones detalladas	2
3. Problemas encontrados durante el diseño y la implementación	2
4. Limitaciones de la aplicación	3
5. Posibles extensiones	3
6. Conclusiones	3
7. Ejemplos de prueba	4
7.1. Concurrencia	4
7.2. Transparencia	5
7.3. Robustez	5
8. Guía de instalación	6
9. Instrucciones para la configuración	7
10. Ejemplos de configuración y monitoreo	7
11. Diseño del proyecto	9

1. Introducción

Para la materia "*Protocolos de Comunicación*" se requirió diseñar, analizar, e implementar un servidor proxy para el protocolo SOCKSv5, como también la creación, definición e implementación completa de un protocolo de monitoreo para el servidor.

El servidor debía cumplir con ciertos requerimientos funcionales, entre los cuales destacaban que debía poder atender a múltiples clientes en forma concurrente y simultánea, soporte para conexiones salientes a direcciones *IPv4*, *IPv6*, o usando un *Fully Qualified Domain Name*. Estas conexiones debían estar manejadas utilizando sockets en modo no bloqueante multiplexada.

A su vez, el protocolo diseñado por nosotros debía tener ciertas funcionalidades, como por ejemplo permitir manejo de usuarios (operaciones del estilo ABM), activar/desactivar el *sniffer* de credenciales para el protocolo de POP3, ver métricas del servidor como conexiones históricas, bytes transferidos, etc.

2. Descripción de los protocolos y aplicaciones detalladas

El trabajo consistió en la implementación de un servidor para el protocolo *SOCKSv5*, definido bajo los lineamientos del [RFC 1928]. Este servidor tenía requerimientos como que pueda soportar mas de 500 conexiones en forma concurrente y simultanea, soportar autenticación usuario/contraseña como establece el [RFC 1929].

Esta implementación contó con ciertas limitaciones respecto a la implementación completa del protocolo, tales como que no fue definida la autenticación mediante GSS API (cuya definición se puede encontrar en el [RFC 1961] , como tampoco se contempló la implementación de las opciones de *BIND* ni *UDP* como comandos para SOCKSv5.

Por otro lado, se implementó un protocolo de monitoreo denominado **SCALO_NET** cuya definición completa se puede encontrar en un archivo dentro del repositorio. Este protocolo contempla funcionalidades tales como manejo de usuarios (agregado, borrado, modificaciones), visualización de métricas (conexiones históricas tanto de requests hacia el servidor mediante socks como también peticiones de conexión via el protocolo de monitoreo, bytes transferidos, entre otros), activar/desactivar el *dissector* de contraseñas para el protocolo POP3, listar los usuarios que el *dissector* ha ido recolectando.

3. Problemas encontrados durante el diseño y la implementación

Al comenzar uno de las mayores dificultades con las que se encontró el equipo fue la comprensión del código y las herramientas provistas por la cátedra. Si bien estos archivos fueron de gran ayuda a lo largo del proyecto, primero se tuvo que investigar cómo funcionaban las diferentes implementaciones para lograr un óptimo uso de las mismas. Entre ellas, está el selector junto con los buffers para una primera implementación de un proxy TCP y los parsers para el desarrollo del hello y la autenticación de socks. Hubo que indagar como hacer un uso correcto de las intenciones del selector, como también manejar apropiadamente la maquina de estados.

Un dilema de diseño que se nos presentó fue la capacidad que debían tener los buffers para leer información. En la consigna se especifica que: "*Se espera que se maneje de forma eficiente los flujos de información (por ejemplo no cargar en memoria mensajes muy grandes, ser eficaz y eficiente en el intérprete de mensajes).*". Como grupo debatimos como buscar el balance entre tamaño suficiente para contener porciones significativas de la información (para realizar menor cantidad de system calls), pero que no sean excesivamente grandes y sobrecarguen a la memoria. Dado el

scope de este proyecto, y considerando que la cantidad máxima de conexiones se ve significativamente afectado por el uso de la *syscall* `select`, escogimos un tamaño de buffer de 2KB (2048B), que permite tener un tiempo de ejecución bajo y a su vez no sobrecargar a la memoria.

4. Limitaciones de la aplicación

Entrando en lo mencionado en la sección anterior, existe una limitación en cuanto a la implementación del trabajo que está relacionada al uso de la *system call* `select`. La misma cuenta con una particularidad, y es que el registro de los file descriptors que puede escuchar puede contener hasta un máximo de 1024 elementos. Del manual de `select`(`man select(2)`):

”POSIX allows an implementation to define an upper limit, advertised via the constant `FD_SETSIZE`, on the range of file descriptors that can be specified in a file descriptor set. The Linux kernel imposes no fixed limit, but the glibc implementation makes `fd_set` a fixed-size type, with `FD_SETSIZE` defined as **1024**, and the `FD_*()` macros operating according to that limit. To monitor file descriptors greater than 1023, use `poll(2)` instead.”

Otra limitación del trabajo fue el tamaño de los buffers de lectura y escritura. Como se aclara en la consigna, se pide que sea lo suficientemente grande para tener un tiempo de ejecución bajo manteniendo la eficiencia, pero que a su vez no sobrecargue la memoria con buffers gigantes.

5. Posibles extensiones

Por lo mencionado anteriormente, una posible extensión de este proyecto sería reemplazar el uso de la *syscall* `pselect()` por otro mecanismo como por ejemplo, la *syscall* `poll()` la cual es capaz de manejar un número mayor de file descriptors de forma concurrente.

Por otro lado, el registro de usuarios y registro de contraseñas podría dejar de ser volátil almacenando estos datos en algún archivo de forma tal de no perderlos cada vez que reiniciamos el servidor. Esto también aplica al almacenamiento de los usuarios y contraseñas recuperados por el *sniffer* de POP3.

Las extensiones de el servidor a nivel funcional todavía se pueden mejorar. De por si, podemos considerar la implementación de el método de autenticación mediante `GSS API`, como también la implementación de los comandos de socks `BIND` y `UDP`.

El protocolo de monitoreo también tiene aún un gran espacio de mejoría. Existen varias métricas históricas mas para considerar, como por ejemplo direcciones con mayor cantidad de pedidos, destinos con mas pedidos, métricas temporales como tiempo de existencia del servidor, pedido mas largo, pedido mas corto. Esto último también se puede aplicar a nivel tamaño del pedido.

6. Conclusiones

El trabajo consistía en realizar un proxy *SOCKSv5* que permitiera realizar pedidos de manera transparente. Luego de varias semanas de trabajo intensivo, creemos que logramos el objetivo, teniendo la posibilidad de navegar correctamente mediante un browser.

El funcionamiento del servidor cumple con el requerimiento de ser no bloqueante, esto fue validado en las pruebas, como también permite concurrencia de pedidos (aunque con un límite debido al diseño), pero esto puede ser modificado a futuro con el uso de pool.

En cuanto al protocolo de monitoreo, el mismo permite resolver cuestiones relacionadas al manejo de usuarios, obtener métricas históricas, habilitar y deshabilitar el *dissector* de contraseñas.

Como balance general, como grupo creemos que el trabajo alcanza los requerimientos pedidos tanto para el proxy como para el protocolo implementado. Para todos resultó un desafío enorme, e inclusive se podría considerar la (hasta el momento) tarea mas complicada que tuvimos que enfrentar como grupo dentro de la carrera, debido a que se trató tanto de comprender protocolos de manera completa para poder interpretar sus pedidos y devolverlos en el formato indicado, como también poder producir nuestro propio protocolo, definiendo las diferentes variables que podía tener el mismo.

7. Ejemplos de prueba

Para probar diferentes parámetros del trabajo práctico, sometimos al servidor a diversas pruebas para ver como reaccionaba.

7.1. Concurrencia

Si bien es difícil emular 1000 conexiones en simultaneo, lo que hicimos fue, mediante la consola, correr el siguiente comando:

```
$> for i in {1..1000}; do curl -x socks5://localhost
http://www.google.com > /dev/null ; done
```

lo que nos permitió generar 1000 conexiones (aunque no todas simultaneas dado que a medida que llegan se van resolviendo), y el servidor contestó de manera satisfactoria:

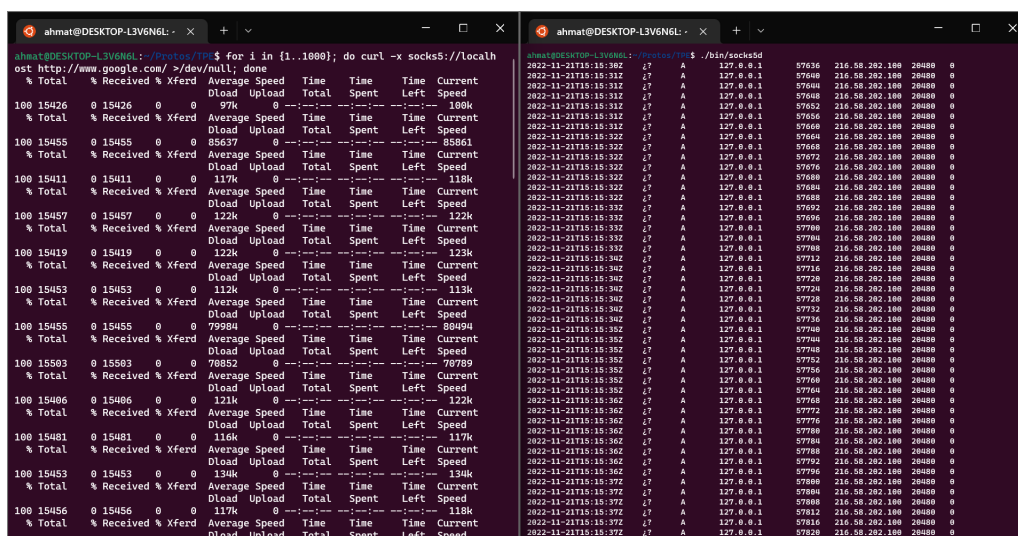


Figura 1: Procesando 1000 conexiones pseudoconcurrentes

El servidor logra manejar las 1000 conexiones entrantes que, al ir llegando, se van resolviendo, por lo que nunca se excede el límite del select mencionado dentro de las limitaciones. Sin embargo, es una buena medida para ver que

7.2. Transparencia

Nos pareció interesante mostrar la transparencia del uso del servidor para navegar por internet mediante un Web Browser clásico. Mozilla Firefox provee una opción sumamente cómoda para testear esto, permitiendo setear el proxy desde su configuración de manera nativa. Una vez configurado el servidor (logicamente debemos correrlo antes de comenzar a navegar), podemos acceder a distintos recursos de manera *transparente* para el usuario:

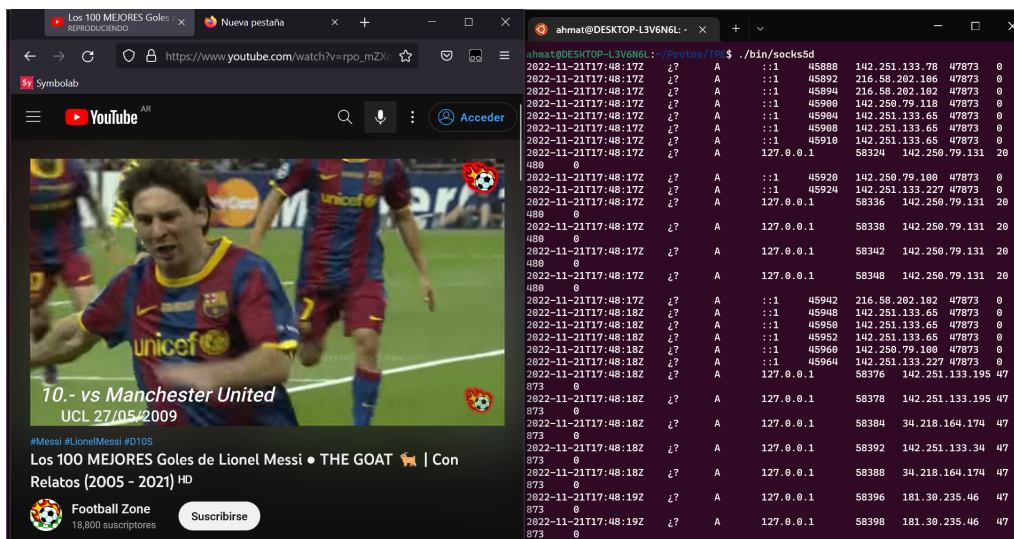


Figura 2: *YouTube* corriendo con el proxy procesando la información

7.3. Robustez

Otro aspecto a analizar es el volumen de descarga que puede tolerar el servidor, y como varía esto con el tiempo del buffer. Para esto, pusimos una prueba de rigor intentando traer una imagen de Ubuntu que se encontraba servida mediante un servidor *NGINX*. Esto permitía dejar de lado una posible especulación respecto a tiempos del ISP. Los resultados que obtuvimos fueron los siguientes:

```
manuel@manuel-HP-Laptop-15-bs0xx:~/Desktop$ curl -H "Host: localhost" localhost/ubuntu-22.04-desktop-amd64.iso | sha256sum
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100 3485M  100 3485M    0     0  179M      0  0:00:19  0:00:19  --:--:-- 183M
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07 -
```

Figura 3: Tiempo de ejecución sin el uso del proxy

```

manuel@manuel-HP-Laptop-15-bs0xx:~/Desktop/TPE-Socks5$ curl -x socks5://localhost -H "Host: localhost"
localhost/ubuntu-22.04-desktop-amd64.iso | sha256sum
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
             Dload  Upload   Total   Spent    Left   Speed
100 3485M  100 3485M    0     0  90.7M    0  0:00:38  0:00:38 --:--:--  94.0M
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07 -
manuel@manuel-HP-Laptop-15-bs0xx:~/Desktop/TPE-Socks5$ curl -x socks5://localhost -H "Host: localhost"
localhost/ubuntu-22.04-desktop-amd64.iso | sha256sum
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
             Dload  Upload   Total   Spent    Left   Speed
100 3485M  100 3485M    0     0  135M    0  0:00:25  0:00:25 --:--:--  137M
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07 -
manuel@manuel-HP-Laptop-15-bs0xx:~/Desktop/TPE-Socks5$ curl -x socks5://localhost -H "Host: localhost"
localhost/ubuntu-22.04-desktop-amd64.iso | sha256sum
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
             Dload  Upload   Total   Spent    Left   Speed
100 3485M  100 3485M    0     0  151M    0  0:00:23  0:00:23 --:--:--  152M
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07 -
manuel@manuel-HP-Laptop-15-bs0xx:~/Desktop/TPE-Socks5$ curl -x socks5://localhost -H "Host: localhost"
localhost/ubuntu-22.04-desktop-amd64.iso | sha256sum
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
             Dload  Upload   Total   Spent    Left   Speed
100 3485M  100 3485M    0     0  162M    0  0:00:21  0:00:21 --:--:--  159M
b85286d9855f549ed9895763519f6a295a7698fb9c5c5345811b3eefadfb6f07 -

```

Figura 4: Tiempo de ejecución con tamaños de buffer (en bytes) 512, 1024, 2048, y 4096, en este orden

La conclusión que observamos es que a mayor tamaño de buffer, en este caso, los tiempos se acortan, y la velocidad de transferencia promedio es mas alta. Esto denota el peso que tienen las system calls involucradas a la hora de enviar y recibir información en un nivel temporal. Sin embargo, los tiempos siempre se encuentran por arriba del curl realizado sin el proxy mediante.

Aclaración: La impresión del hash del archivo fue a modo de confirmación de que el archivo copiado es efectivamente el mismo que el original.

8. Guía de instalación

El proyecto requiere unicamente dos dependencias que suelen venir incluidas con cualquier versión de Linux para *WSL*, o logicamente la distro escogida:

- GCC
- Make

Luego, colocandonos en la raíz del proyecto (*./TPE-Socks5*) ejecutamos el comando `$> make all`, lo que va a generar los ejecutables:

- client
- socks5d

El primero corresponde a la conexión del cliente para el protocolo de monitoreo, mientras que la segunda corresponde con la inicialización del servidor. Su uso se encuentra descripto apropiadamente en el **README.md** que se encuentra en la raíz del repositorio.

9. Instrucciones para la configuración

El proyecto no requiere mayor configuración que los pasos de instalación. Una vez realizados los mismos, nos colocamos sobre la raíz del proyecto y ejecutamos:

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

para correr el servidor. De esta manera, uno ya puede realizar *curls* mediante el proxy y se deberían resolver efectivamente. Para ver diferentes opciones, referir al manual de uso que se encuentra en la raíz del proyecto, o también al **README.md**, ubicado en el mismo lugar.

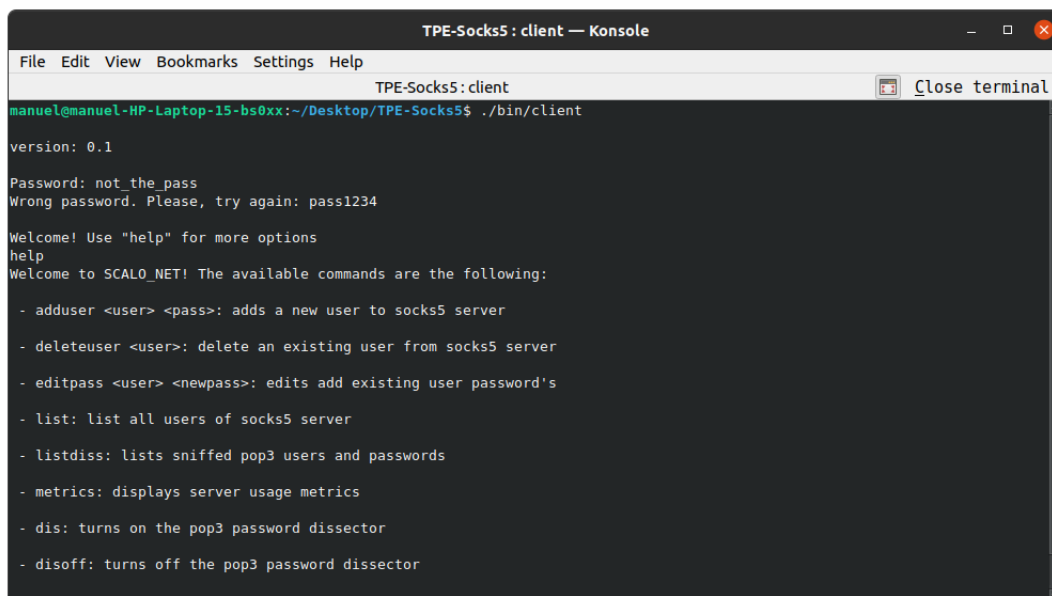
Para utilizar el protocolo de monitoreo, uno puede utilizar el otro ejecutable generado corriendo el comando:

```
$> .bin/client
```

Una vez ejecutado, se presentará la conexión con el servidor. Para ver comandos y opciones, referir al archivo **SCALO_NET.pdf**, que se encuentra en la raíz del proyecto

10. Ejemplos de configuración y monitoreo

El protocolo de monitoreo tiene en primera instancia una fase de autenticación, para el cuál se requiere una password para ingresar. La misma se valida contra la base de datos interna del servidor, y así proporciona acceso (o no): Como se puede observar en la imagen anterior, con el



```
TPE-Socks5: client — Konsole
File Edit View Bookmarks Settings Help
TPE-Socks5: client
manuel@manuel-HP-Laptop-15-bs0xx:~/Desktop/TPE-Socks5$ ./bin/client
version: 0.1
Password: not_the_pass
Wrong password. Please, try again: pass1234
Welcome! Use "help" for more options
help
Welcome to SCALO_NET! The available commands are the following:

- adduser <user> <pass>: adds a new user to socks5 server
- deleteuser <user>: delete an existing user from socks5 server
- editpass <user> <newpass>: edits add existing user password's
- list: list all users of socks5 server
- listdiss: lists sniffed pop3 users and passwords
- metrics: displays server usage metrics
- dis: turns on the pop3 password dissector
- disoff: turns off the pop3 password dissector
```

Figura 5: Negociación inicial entre cliente y servidor

comando **help** se puede acceder a un menú de ayuda con los diferentes comandos disponibles para ser utilizados. Vale aclarar que ante comandos que retornan correctamente, el servidor responde con un "OK!", y sirve un mensaje de error apropiado en caso que no haya sido exitoso.

Entre ellos se observan `adduser`, `editpass`, y `deleteuser`. Los 3 comandos son autoexplicativos, e interactúan con la base de datos del servidor de SOCKSv5. A su vez, se tiene un comando `list` que permite visualizar los usuarios actuales registrados dentro del servidor.

```
adduser user1 pass1
OK!
editpass user1 newpass
OK!
deleteuser user1
OK!
list
No users yet
adduser user1 pass1
OK!
list
Socks users:
user1
```

Figura 6: Uso de funciones que permiten operaciones básicas sobre usuarios de SOCKSv5

Si se trata de eliminar un usuario no existente, el servidor contempla este caso y anula la operación.

```
deleteuser user1
OK!
deleteuser user1
Error: user does not exist
```

Figura 7: Intento de borrado de un usuario no existente

También se puede interactuar con el disector de manera ilimitada, es decir, si uno ejecuta el comando `dis` n veces, y finalmente una vez `disoff`, el *dissector* se desactivará.

```
dis
OK!
disoff
OK!
dis
OK!
dis
OK!
disoff
OK!
disoff
OK!
```

Figura 8: Activar y desactivar el disector

También se puede acceder a diferentes métricas relacionadas a conexiones históricas y actuales, como también a bytes transferidos totales.

Aclaración: Se realizó un `curl` a `http://www.google.com` mediante el proxy para mostrar la transferencia de bytes y su guardado.

```
metrics
curr_socks;hist_socks;curr_control;hist_control;curr_total;hist_total;bytes_trnf
0;1;1;1;1;2;16346
```

Figura 9: Activar y desactivar el disector

11. Diseño del proyecto

Para el diseño del proyecto, se utilizaron diversas herramientas provistas por la cátedra: La maquina de estados, el sistema de buffers, el selector, el parseo de argumentos, fueron algunas de las herramientas que conformaron la base del trabajo.

La configuración básica del servidor comienza abriendo dos sockets pasivos tanto para IPv4 como IPv6, destinados a atender las conexiones entrantes. A su vez, se abren dos sockets (misma lógica) para atender las conexiones entrantes de el protocolo de monitoreo.

Cada conexión cuenta con su propia maquina de estados que sigue el momento de conexión en el que se encuentra, realizando un parseo byte a byte. Esto es posible debido a la abstracción que provee esta librería. Esto aplica tanto a las conexiones de Socksv5 quienes cuentan con sus parsers designados, como también para el protocolo de monitoreo.