



72.07

Protocolos de comunicación

1º Cuatrimestre del 2022

Trabajo Práctico Especial “Servidor Proxy Socks5”

Integrantes:

- | | |
|--------------------|-------|
| - Ezequiel Pérez | 61475 |
| - Nicolás Birsa | 61482 |
| - Gonzalo Baliarda | 61490 |
| - Valentín Ye Li | 61011 |

Grupo: 13

Índice

1. Introducción	3
2. Descripción de los protocolos y aplicaciones desarrolladas	3
2.1 Servidor proxy SOCKSv5	3
2.2 Protocolo de monitoreo de servidor (versión 1)	5
2.3 Cliente del protocolo de monitoreo	7
3. Problemas encontrados	8
3.1 Diseño	8
3.2 Implementación	8
4. Limitaciones	9
4.1 Cantidad de clientes concurrentes	9
4.2 Tamaño de buffers	10
4.3 Generales	11
5. Posibles extensiones	12
6. Conclusiones	12
7. Ejemplos de prueba	13
7.1 Bindings del servidor	13
7.2 Integridad y velocidad de descarga de archivos	13
7.3 Desconexión repentina	14
7.4 Errores en origin servers	15
7.5 Uso con browser (Mozilla Firefox)	16
7.6 Uso del password disector	17
7.7 Prueba de estrés	18
8. Instalación	22
9. Instrucciones de configuración	23
10. Ejemplos de configuración y monitoreo	25
10.1 Cantidad de conexiones concurrentes e históricas del server	25
10.2 Cantidad de bytes transferidos	27
10.3 Añadir un usuario del proxy e imprimir la lista de usuarios del proxy	28
11. Diseño del proyecto	29
12. Anexo	30

1. Introducción

El objetivo principal de este trabajo práctico especial fue lograr la exitosa implementación de un servidor proxy para el protocolo **SOCKSv5**, aplicando los conocimientos adquiridos a lo largo de la materia como programación de aplicaciones cliente/servidor con sockets, comprensión de estándares de la industria o la capacidad de diseñar protocolos de aplicación.

Dicho servidor cumple con una serie de requerimientos funcionales y no funcionales, además de ser compatible con los RFCs correspondientes a cada funcionalidad para así poder ser usado con cualquier cliente “compliant” de las mismas, como *curl* o *netcat*.

2. Descripción de los protocolos y aplicaciones desarrolladas

2.1 Servidor proxy SOCKSv5

El servidor proxy desarrollado implementa las características del protocolo **SOCKSv5** descritas en el [RFC1928], además de cumplir con una serie de requerimientos. A saber:

1. Poder atender a, por lo menos, 500 clientes en forma concurrente y simultánea.

El servidor es capaz de atender hasta 509 clientes en simultáneo. Dicho número luego será explicado y detallado en la sección de Limitaciones.

2. Soportar autenticación usuario / contraseña [RFC1929].

La misma es opcional dentro del servidor y se activa al agregar un usuario, ya sea al iniciar el servidor o luego mediante el cliente de monitoreo*. Así se puede permitir a los usuarios navegar de forma anónima o sólo si fueron “registrados” dentro del server.

****Esta acción solo se puede realizar previo al primer pedido al proxy, ya que una vez completado dicho pedido por un usuario anónimo la autenticación se desactiva de manera permanente. Lo mismo ocurre si se realiza con un usuario registrado; activándose de manera permanente.***

3. Soportar de mínima conexiones salientes a servicios TCP a direcciones IPv4, IPv6, o utilizando FQDN que resuelvan cualquiera de estos tipos de direcciones.

Dentro del servidor se utiliza un *thread* o hilo encargado de la resolución de nombres que trabaja en forma paralela al servidor; mientras el servidor realiza las conexiones pertinentes a los clientes, el hilo mencionado resuelve los FQDNs, dejándolos a disposición del server para cuando éste necesite realizarles una petición.

4. Ser robusto en cuanto a las opciones de conexión

El hilo utilizado devuelve una lista de IPs correspondientes al FQDN enviado, la cual se itera hasta poder realizar una conexión exitosa y obtener el recurso. Caso contrario se arrojará un error de *TTL exceeded* como indica el protocolo **SOCKSv5**.

5. Reportar los fallos a los clientes usando toda la potencia del protocolo

Además del ya mencionado *TTL exceeded*, en caso de errores al intentar la petición, el servidor informa al cliente mediante los códigos de estatus definidos en el *RFC1928* correspondiente al protocolo. Algunos de ellos son "*Host unreachable*" en caso de no poder resolver el *FQDN* enviado o "*Connection refused by destination host*" en caso de sí poder resolverlo pero no conectarse.

6. Implementar mecanismos que permitan recolectar métricas que ayuden a monitorear la operación del sistema

El servidor implementa un Protocolo de Monitoreo que permite a un cliente del mismo obtener información como la cantidad de conexiones históricas y concurrentes, además de la cantidad de bytes transferidos. Estas y otras métricas luego serán detalladas en la sección de dicho protocolo, pero cabe destacar que son **volátiles**, por lo que se reseteará su valor si se reinicia el servidor.

7. Implementar mecanismos que permitan a los usuarios cambiar la configuración del servidor en tiempo de ejecución.

Al igual que en la sección anterior, estos cambios incluyen el encendido/apagado de un disector de contraseñas *POP3* y el añadido de usuarios al server y serán detallados luego, pero al tener métricas **volátiles** dentro del servidor, estos cambios tendrán efecto sólo hasta que se decida reiniciar el mismo.

8. Implementar un registro de acceso que permita a un administrador entender los accesos de cada uno de los usuarios

Este registro se presenta en forma de Logs, los cuales luego de completar un pedido se imprimen en pantalla, detallando la fecha y hora del pedido, el usuario que lo hizo (en caso de que no sea anónimo) y dirección de origen y de destino, entre otros.

9. Monitorear el tráfico y generar un registro de credenciales de acceso de forma similar a ettercap.

Dentro de los Logs ya mencionados, en el caso de que un usuario se quiera autenticar en el servidor usando el protocolo *POP3*, el *password disector* implementado (activado por defecto) permite obtener las credenciales de dicho usuario.

Por otro lado, dentro de los aspectos no funcionales, el servidor usa sockets no bloqueantes para comunicarse con los clientes y realizar pedidos, además de que sus comandos se adaptan al formato provisto por la cátedra. El resto de requerimientos no funcionales se detallarán en mayor profundidad a lo largo del informe.

2.2 Protocolo de monitoreo de servidor (versión 1)

El protocolo implementado para monitoreo en el servidor es un protocolo **binario**, montado sobre **TCP** y **no orientado a conexión**, mediante el cual se puede cambiar la configuración del server en tiempo de ejecución mediante una serie de *requests* y obtener feedback sobre el estado posterior del servidor con las *responses* devueltas. Dichos cambios sólo pueden ser realizados por usuarios **admin** previamente registrados en el servidor con un par **<nombre, token>**, donde el token será un string de 16 caracteres ASCII usado en cada *request* como el método de autenticación de la misma.

VER	TOKEN	MÉTODO	TARGET	DLEN	DATA
1	16	1	1	2	1 to 65535

Figura 1. Formato de una *request*, con cantidad de bytes por campo.

Siguiendo lo dicho anteriormente, en las requests además de un token de autenticación se debe enviar la versión del protocolo usada (actualmente solo soporta la versión 1) y el **método, target, dlen y data** a enviar. Estos campos describen:

→ Método

Acción a realizar en el servidor. Puede ser un **GET**, con el que se obtiene información de un parámetro del servidor, o un **CONFIG**, con el que se cambian parámetros del servidor.

→ Target

Su valor depende del método, y define el tipo de parámetro a obtener/modificar en el servidor. En el caso de un método **GET**, se puede obtener:

- ❖ Cantidad de conexiones históricas
- ❖ Cantidad de conexiones concurrentes
- ❖ Cantidad de bytes transferidos
- ❖ Listado de usuarios del proxy*
- ❖ Listado de admins del servidor

Por otro lado, mediante un **CONFIG** se puede modificar:

- ❖ Encender/Apagar el *password disector* POP3
- ❖ Agregar un usuario del proxy*
- ❖ Agregar un usuario admin
- ❖ Borrar un usuario del proxy*
- ❖ Borrar un usuario admin

***Los usuarios del proxy son usuarios regulares registrados para la navegación con autenticación, mientras que los usuarios admin son aquellos capaces de obtener/modificar parámetros del servidor mediante el protocolo de monitoreo.**

→ *Dlen*

Este campo indica la longitud de la data enviada, y sirve para determinar el final del paquete cuando se lo está leyendo. Si la longitud de la data fuera constante, no haría falta este campo, pero al poder ingresar, por ejemplo, nombres de usuario y contraseñas de longitud variable, es necesario su uso.

→ *Data*

Como su nombre lo indica, son los datos que, dado un método y target específicos, brindan la información necesaria para efectuar dicha acción. En el caso de un método **GET**, no es necesario enviar datos en ningún método, aunque se recomienda enviar un **0** al servidor como data. En cambio, todos los métodos **CONFIG** necesitan un campo data para ejecutarse, teniendo los parámetros a modificar en el servidor como estado próximo del password disector (ON/OFF), credenciales de usuario proxy/admin a agregar o el nombre de usuario a borrar.

STATUS	DLEN	DATA
1	2	1 to 65535

Figura 2. Formato de una *response*, con cantidad de bytes por campo.

Por último, tenemos las *responses* del protocolo, que incluyen los campos de **status**, que indica si se concretó la petición enviada o el tipo de error ocurrido en caso de fallar, **dlen** que tiene el mismo propósito que en la **request**, y data, que en el caso de éxito contiene los datos solicitados al servidor, como el número de conexiones concurrentes/históricas, o la lista de usuarios proxy/admin.

El resto de detalles sobre el protocolo se pueden consultar en el **RFC** adjunto, que detalla a fondo la implementación del mismo, acompañado de ejemplos en *requests* y *responses*.

2.3 Cliente del protocolo de monitoreo

Por último, también se desarrolló un cliente bloqueante que implementa el protocolo antes visto, con una serie de comandos que permiten realizar todas las acciones de obtener/configurar parámetros disponibles en el servidor. Se puede consultar el formato del comando del cliente y sus parámetros corriendo './client -h', como se muestra en la figura a continuación.

```
vyeli@DESKTOP-9FDMFLV:~/proto/TPE/socksv5-protocol$ ./client -h
Usage: ./client [OPTIONS]... TOKEN [DESTINATION] [PORT]
Options:
-h             imprime los comandos del programa.
-c             imprime la cantidad de conexiones concurrentes del server.
-C            imprime la cantidad de conexiones históricas del server.
-b            imprime la cantidad de bytes transferidos del server.
-a            imprime una lista con los usuarios del proxy.
-A            imprime una lista con los usuarios administradores.
-n            enciende el password disector en el server.
-N            apaga el password disector en el server.
-u<user:pass> agrega un usuario del proxy con el nombre y contraseña indicados.
-U<user:token> agrega un usuario administrador con el nombre y token indicados.
-d<user>      borra el usuario del proxy con el nombre indicado.
-D<user>      borra el usuario administrador con el nombre indicado.
-v           imprime la versión del programa.
```

Figura 3. Formato del comando ./client y sus parámetros.

Las opciones con solo texto son **obligatorias**, como el token, mientras que las que están entre corchetes son **opcionales**. En caso de no especificar ip destino o puerto, se tomarán los valores default correspondientes de 127.0.0.1 y 8080.

La principal limitación del cliente es que sólo puede realizar **un pedido por conexión**, ya que luego de recibir una *response* del servidor, éste corta la conexión impidiendo que el cliente pueda enviar otra request y debiendo crear un nuevo socket para hacerla. Este problema es inherente de la naturaleza del protocolo por no ser orientado a conexión.

Sin embargo, el cliente sí permite realizar múltiples pedidos en una ejecución, concatenando las opciones en el orden que se deseen ejecutar. Así, si se corre el comando:

>> ./client -u juan:juan -d juan <token>

El estado final del servidor será el mismo al inicial. Lo mismo ocurrirá si se pide listar usuarios del proxy y luego admins, mostrando una lista abajo de la otra, y así con los demás comandos. También se puede conectar a direcciones IP versión 4 y 6, ya que el servidor puede estar corriendo en ambas.

Por otro lado, aprovechando el conocimiento sobre las *requests* enviadas, el cliente interpreta las *responses* recibidas e imprime un mensaje sobre el estado de la acción realizada. Así si, por ejemplo, se añade un usuario, a pesar de que el servidor no responde con los datos del usuario agregado, el cliente sí sabe quién se agregó y puede informarlo.

3. Problemas encontrados

3.1 Diseño

En cuanto al diseño se tuvieron problemas más que nada con las implementaciones base para el proyecto provistas por la cátedra, en las cuales:

- ❖ En los parches, habían archivos con los que no se contaba inicialmente (por ej: un *readme* y un *makefile*), por lo que fallaban al intentar aplicarlos. Para solucionarlo, se limpió el parche y funcionó correctamente.
- ❖ Se tuvieron problemas para entender inicialmente los códigos de las implementaciones base, sobre todo por su extensión y complejidad. Sin embargo, una vez se pudo comprender el funcionamiento de los mismos, resultaba fácil usarlos ya que son muy completos en cuanto a funcionalidad.
- ❖ Dentro de los problemas mencionados en el ítem anterior, destaca el comprender el código de la máquina de estado y su relación con el selector.

Luego, fuera de la implementación base, también se tuvieron dificultades a la hora de idear la heurística para el disector POP3, que luego de una serie de cambios resultó en una serie de pasos a seguir. A saber:

1. Leer lo primero que escribe el *origin server*. Si no empieza la comunicación con un "+OK", no sniffear más los paquetes de esa conexión, de ningún lado.
2. Si el *origin server* inició con un "+OK", empieza a sniffear todo lo que pone el cliente, intentando formar la secuencia "USER\n" -> "PASS\n".
3. Si logra matchear la secuencia anterior, se pone a sniffear lo próximo que ponga el *origin server*. Si escribe un "+OK", loguea el usuario y la pass encontrada, y si no, vuelve al paso 2.

Por último, un leve problema radica en que a pesar de que el servidor en su diseño es no bloqueante, como las llamadas a la función usada para la resolución de nombres en el hilo es bloqueante (*getaddrinfo*), entonces el server puede "bloquearse" esperando la resolución de nombres para realizar la conexión al *origin server*.

3.2 Implementación

En este apartado se tuvieron problemas sobre todo a nivel de manejo de memoria de C, ya que:

- ❖ Al poder estar manejando los enteros en *little endian* localmente y tener que pasarlos a *network order* (*big endian*), en algunos casos no bastaba usar la librería de funciones *htons*, *ntohs*, etc, como en los parsers, donde se tenía que leer byte a byte y rearmar el número para luego pasarlo al host order, lo que en algunos casos generó mayor confusión de lo esperado.

- ❖ Para mandar los datos desde el cliente al servidor inicialmente se utilizó un struct con los campos ordenados y de tamaño según se esperaba en una request. Sin embargo, ya que internamente en memoria se realizaba un *padding* de algunos campos, esto hacía que se mandaran más bytes de la cuenta, resultando en mayores problemas que sólo pudieron ser identificados al analizar los paquetes enviados con wireshark. Dado esto, se decidió usar un buffer para enviar los datos y colocar los bytes "a mano" en cada parte del buffer emulando la struct usada.

Por otra parte, inicialmente se diseñaron los parámetros de cada cliente en el server como variables globales, pero una vez implementado surgió el problema de que varios clientes compartían una misma variable. Por ello, se decidió mover dichas variables a estructuras de datos privadas de cada cliente.

4. Limitaciones

4.1 Cantidad de clientes concurrentes

La principal limitación en el desarrollo del servidor viene dada por la cantidad máxima de usuarios que se puede manejar de manera concurrente, ya que para manejar las conexiones entrantes al servidor se utiliza la función *pselect*, la cual permite escuchar en un máximo de **1024 file descriptors** a la vez. Luego este número se ve limitado debido a las necesidades de implementación del servidor, que incluyen:

- ❖ **2 File descriptors** para imprimir mensajes por STDOUT y STDERR
- ❖ **4 File descriptors** para sockets pasivos IPv4 e IPv6, en caso que se corra el servidor en modo default.

Esto nos deja con 1018 *File descriptors* disponibles para asignar a los clientes, donde los clientes del proxy necesitan 2 *File descriptors* (uno para conectarse al servidor y otro que abre el servidor para la request al origin server) y los clientes del servidor de monitoreo necesitan un único *File descriptor* para conectarse a éste, encontramos que el máximo soportado por el servidor es de:

- ❖ **509 clientes proxy** (sin clientes de monitoreo)
- ❖ **1018 clientes de monitoreo** (sin clientes proxy)

4.2 Tamaño de buffers

Por otro lado, otra limitación importante existente es la del tamaño del buffer usado, ya que el servidor debe guardar la información de la request antes de enviarla al usuario. Por un lado, el buffer de escritura al cliente debe tener un tamaño de al menos 10 caracteres para armar una respuesta acorde al protocolo SOCKSv5. Luego, para determinar un tamaño óptimo del buffer acorde a nuestras necesidades, se realizaron 2 pruebas:

- ❖ Test de tiempo de ejecución al descargar un archivo.
- ❖ Test de tamaño de buffer realmente usado.

Dichas pruebas se realizaron a tamaños de buffer potencias de 2 partiendo desde 128 hasta 8192, realizando 5 mediciones de tiempos por cada tamaño de buffer a fines de reducir los errores de medición y normalizar el valor obtenido. En cuanto al tamaño de buffer usado nos limitamos a observar en las mediciones si al guardar datos en el buffer, éste se llenaba por completo o sólo parte de él, dejando espacio inutilizado y desaprovechado.

En la primera prueba se utilizó un archivo de 10MB, y se descargó utilizando una conexión por cable ethernet con velocidad de 150Mb. Así, volcando los valores promedio de demora en la descarga del archivo en función del tamaño del buffer, se puede obtener un gráfico como el siguiente:

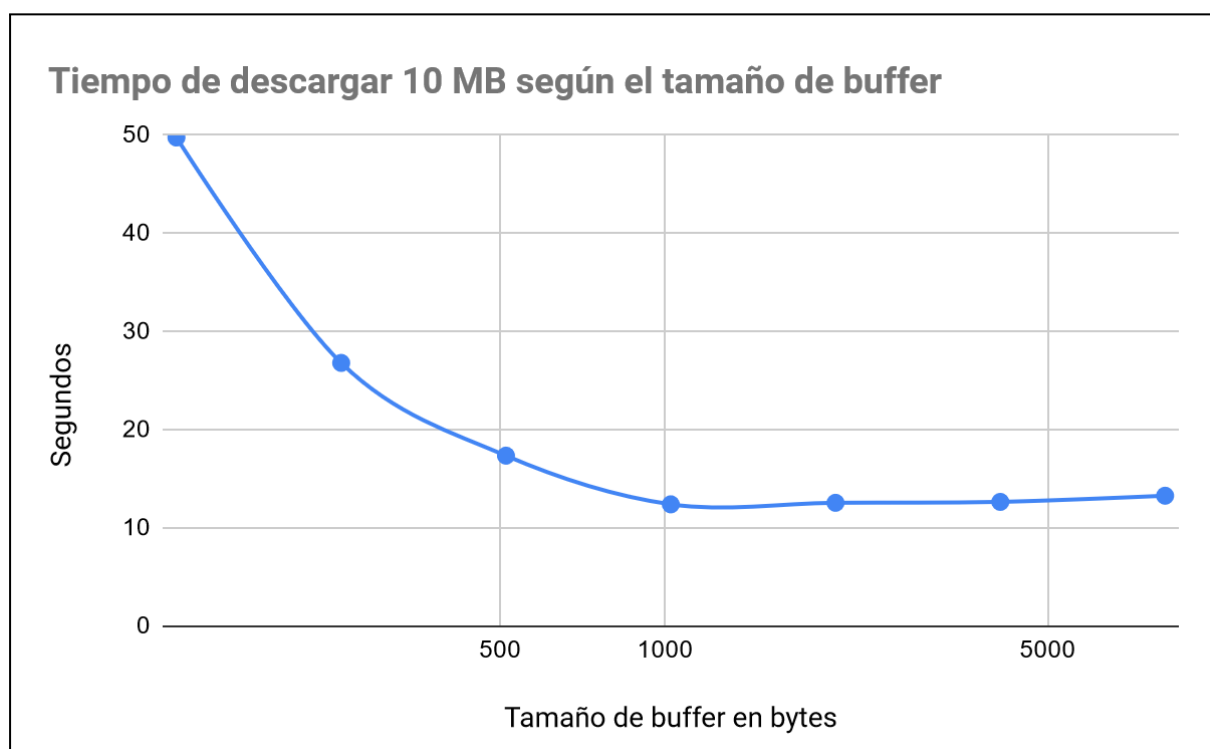


Figura 4. Gráfico de promedio de tiempo en descargar un archivo para diferentes tamaños de buffer

Como se puede observar en el gráfico, podemos decir que a partir de un buffer de tamaño 1024, el tiempo de descarga se mantiene constante, e incluso es levemente mayor para tamaños superiores de buffer como 2048 y 4096.

Por otro lado, al revisar con strace las respuestas recibidas por los origin servers, se determinó que a partir del tamaño de buffer 4096, aunque leve, ya no se llenaba el buffer al realizar una lectura.

```
read(3, "7fff7000-8fff7000 rw-p 00000000 "..., 4096) = 4005
read(3, "7f091c2f9000-7f091c2fc000 r--p 0"..., 91) = 91
close(3) = 0
munmap(0x7f091cd5c000, 4096) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f091cd5b000
open("/proc/self/maps", O_RDONLY) = 3
read(3, "7fff7000-8fff7000 rw-p 00000000 "..., 8192) = 4005
read(3, "7f091c2f9000-7f091c2fc000 r--p 0"..., 4187) = 2595
read(3, "", 1592) = 0
close(3) = 0
munmap(0x7f091cd5e000, 8192) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f091cd5f000
open("/proc/self/maps", O_RDONLY) = 3
read(3, "7fff7000-8fff7000 rw-p 00000000 "..., 4096) = 4005
read(3, "7f091c2f9000-7f091c2fc000 r--p 0"..., 91) = 91
close(3) = 0
```

Figura 5. Vista con comando strace de lectura en buffer de tamaño 4096 bytes

Esto, sumado a las pruebas de tiempo antes realizadas, determinan que los valores óptimos de buffer en cuanto a tiempo y no desperdiciar espacio son 1024 y 2048. Pero, dado que un tamaño de buffer ocupa la mitad de espacio en memoria que el otro, se decidió utilizar un buffer de tamaño 1024 para las conexiones de usuarios del proxy.

Por último, cabe destacar que las tablas con los valores observados para las mediciones de cada tamaño de buffer se encuentran en la sección de Anexo del informe (tablas 1 a 7).

4.3 Generales

Por otro lado, encontramos otras limitaciones como la imposibilidad de mandar múltiples pedidos al server de monitoreo en una conexión, la de guardar datos permanentes en memoria como cantidades totales de conexiones históricas o bytes transferidos o la de guardar como máximo 10 usuarios de proxy al mismo tiempo en el servidor.

Sin embargo, como la primera es inherente a la naturaleza del protocolo de monitoreo usado mientras que las otras 2 son aceptadas por la consigna de este trabajo práctico, son limitaciones menores en comparación a las otras 2 previamente presentadas.

También cabe mencionar la limitación de que un cliente no puede mandar más de 20 requests para ser procesadas, pero esta limitación fue impuesta buscando que el cliente no envíe una gran cantidad de requests al servidor.

5. Posibles extensiones

Aunque el servidor desarrollado cumpla con los requisitos pedidos, eso no implica que no se puedan realizar mejoras a las funcionalidades del mismo o incluso reemplazarlas por otras mejores. Así, una lista de extensiones aplicables serían:

- ❖ Poder guardar parámetros del servidor de forma permanente, buscando que persistan más allá de cuando se apague el servidor (datos no volátiles).
- ❖ Poder activar o desactivar la autenticación de los usuarios del proxy manualmente y no que se active/desactive según el primer usuario que haga una petición al servidor.
- ❖ Aumentar la cantidad de parámetros disponibles del servidor del protocolo de monitoreo, como el *uptime*, que le permitan a un usuario administrador poseer un mayor *feedback* sobre el estado del servidor.
- ❖ Modificar el protocolo de monitoreo para hacerlo orientado a conexión y permitir que las aplicaciones clientes puedan realizar múltiples peticiones en una única conexión.

6. Conclusiones

A lo largo de este mes realizando el trabajo práctico, se pudieron aplicar y consolidar los conocimientos adquiridos sobre los distintos protocolos de comunicación vistos durante el transcurso de la materia. Así, se pudo desarrollar tanto un servidor proxy que implemente el protocolo SOCKSv5 como un protocolo propio, experimentando de punta a punta todo el proceso de diseño y desarrollo de ambos.

Por lo tanto, a pesar de las dificultades encontradas en la realización de este trabajo práctico especial, nos llevamos el conocimiento y la experiencia de este proceso.

7. Ejemplos de prueba

7.1 Bindings del servidor

1. Caso servidor sin IPs ni puertos pasados en parámetros

>> **./socks5d**

```
root@COMPU-NICO:/home/nico/protos/socksv5-protocol# netstat -nlp | grep socks5d
tcp        0      0 127.0.0.1:8080      0.0.0.0:*           LISTEN      12847/./socks5d
tcp        0      0 0.0.0.0:1080        0.0.0.0:*           LISTEN      12847/./socks5d
tcp6       0      0 :::1:8080           :::*                 LISTEN      12847/./socks5d
tcp6       0      0 :::1080             :::*                 LISTEN      12847/./socks5d
```

2. Caso servidor proxy con IPv4 y servidor de monitoreo con IPv6 (ambos con puerto custom)

>> **./socks5d -I 127.127.45.34 -p 1950 -L 2800:810:5e7:dd9:9 -P 9100**

```
vyeli@vyeli-F117:~$ sudo netstat -nlp | grep socks5d
tcp        0      0 127.127.45.34:1950  0.0.0.0:*           LISTEN      6280/./socks5d
tcp6       0      0 2800:810:5e7:dd9:9:9100 :::*                 LISTEN      6280/./socks5d
```

3. Caso servidor proxy con IPv6 y servidor de monitoreo con IPv4 (ambos con puerto custom)

>> **./socks5d -I 2800:810:5e7:dd9:9 -p 1476 -L 127.230.12.99 -P 10450**

```
vyeli@vyeli-F117:~$ sudo netstat -nlp | grep socks5d
tcp        0      0 127.230.12.99:10450 0.0.0.0:*           LISTEN      6254/./socks5d
tcp6       0      0 2800:810:5e7:dd9:9:1476 :::*                 LISTEN      6254/./socks5d
```

7.2 Integridad y velocidad de descarga de archivos

Para las siguientes pruebas se utilizó un archivo de 10MB, que se descargó por una conexión ethernet de 100Mb.

1. Descarga de archivo con curl y md5sum de dicho archivo

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ time curl http://212.183.159.230/10MB.zip --output - | md5sum
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left  Speed
100 10.0M  100 10.0M    0     0 1672k      0  0:00:06  0:00:06 --:--:-- 2106k
3aa55f03c298b83cd7708e90d289afbd -

real    0m6.127s
user    0m0.042s
sys     0m0.049s
```

2. Descarga de archivo con curl y md5sum de dicho archivo usando el servidor proxy

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ time curl -x socks5h://localhost:1080 http://212.183.159.230/10MB.zip --output - | md5sum
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
100 10.0M  100 10.0M    0     0 1740k      0  0:00:05  0:00:05 --:--:-- 2410k
3aa55f03c298b83cd7708e90d289afbd -

real    0m5.890s
user    0m0.036s
sys     0m0.038s
```

Como se puede ver en ambos casos, el tiempo de descarga es parecido y el hash md5 es idéntico, mostrando que se mantuvo la integridad del archivo post descarga.

7.3 Desconexión repentina

1. Del cliente

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ ./socks5d
Socks: listening on IPv4 TCP port 1080
Monitor: listening on IPv4 TCP port 8080
Socks: listening on IPv6 TCP port 1080
Monitor: listening on IPv6 TCP port 8080

----- LOGS -----

2022-06-21T02:03:43-03 <anonymous> A 127.0.0.1 54502 212.183.159.230 80 0

nico@COMPU-NICO:~/protos/socksv5-protocol$ curl -x socks5h://localhost:1080 http://212.183.159.230/10MB.zip --output - > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
 9 10.0M    9 998k    0     0 291k      0  0:00:35  0:00:03  0:00:32 291k^C
nico@COMPU-NICO:~/protos/socksv5-protocol$
```

El cliente se desconecta pero el server no sufre ningún fallo, sino que sigue operando de manera normal.

2. Del servidor

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ ./socks5d
Socks: listening on IPv4 TCP port 1080
Monitor: listening on IPv4 TCP port 8080
Socks: listening on IPv6 TCP port 1080
Monitor: listening on IPv6 TCP port 8080

----- LOGS -----

2022-06-21T02:05:47-03 <anonymous> A 127.0.0.1 54506 212.183.159.230 80 0
^Csignal 2, cleaning up and exiting
closing: Interrupted system call
nico@COMPU-NICO:~/protos/socksv5-protocol$

nico@COMPU-NICO:~/protos/socksv5-protocol$ curl -x socks5h://localhost:1080 http://212.183.159.230/10MB.zip
--output - > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
14 10.0M  14 1477k    0     0 408k      0  0:00:25  0:00:03  0:00:22 408k
curl: (18) transfer closed with 8972341 bytes remaining to read
nico@COMPU-NICO:~/protos/socksv5-protocol$
```

7.4 Errores en origin servers

1. Cuando el origin server (IPv4) no presta servicio, peticiones usando curl con y sin el proxy:

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ curl http://127.0.0.1:3333
curl: (7) Failed to connect to 127.0.0.1 port 3333: Connection refused
nico@COMPU-NICO:~/protos/socksv5-protocol$
```

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ curl -x socks5h://localhost:1080 http://127.0.0.1:3333
curl: (7) Can't complete SOCKS5 connection to 127.0.0.1:3333. (5)
nico@COMPU-NICO:~/protos/socksv5-protocol$
```

2. Cuando el origin server (IPv6) no presta servicio, peticiones usando curl con y sin el proxy:

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ curl http://[::1]:3333
curl: (7) Failed to connect to ::1 port 3333: Connection refused
nico@COMPU-NICO:~/protos/socksv5-protocol$
```

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ curl -x socks5h://localhost:1080 http://[::1]:3333
curl: (7) Can't complete SOCKS5 connection to ::1:3333. (5)
nico@COMPU-NICO:~/protos/socksv5-protocol$
```

3. Falla la resolución de nombres

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ curl -x socks5h://localhost:1080 http://xxxxxxxxxx
curl: (7) Can't complete SOCKS5 connection to xxxxxxxxxxxx:80. (4)
nico@COMPU-NICO:~/protos/socksv5-protocol$
```

4. Origin server resuelve DNS IPv6

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ curl -x socks5h://localhost:1080 http://ipv6.leak.com.ar
curl: (7) Can't complete SOCKS5 connection to ipv6.leak.com.ar:80. (5)
nico@COMPU-NICO:~/protos/socksv5-protocol$
```

En este caso al averiguar la ip del dominio pedido, vemos que es **[::1]** o localhost, por lo que dicha ip se resuelve en el proxy pero no se puede conectar. Además, se puede ver en los casos anteriores que el código de respuesta del pedido SOCKSv5 es acorde al RFC del protocolo, siendo 4 un 'Host Unreachable' (en el 3, por no poder resolver el FQDN) y 5 'Connection refused'. (en los demás por sí resolver el FQDN, pero no poder conectarse).

5. Origin server con múltiples direcciones IP, donde una falla

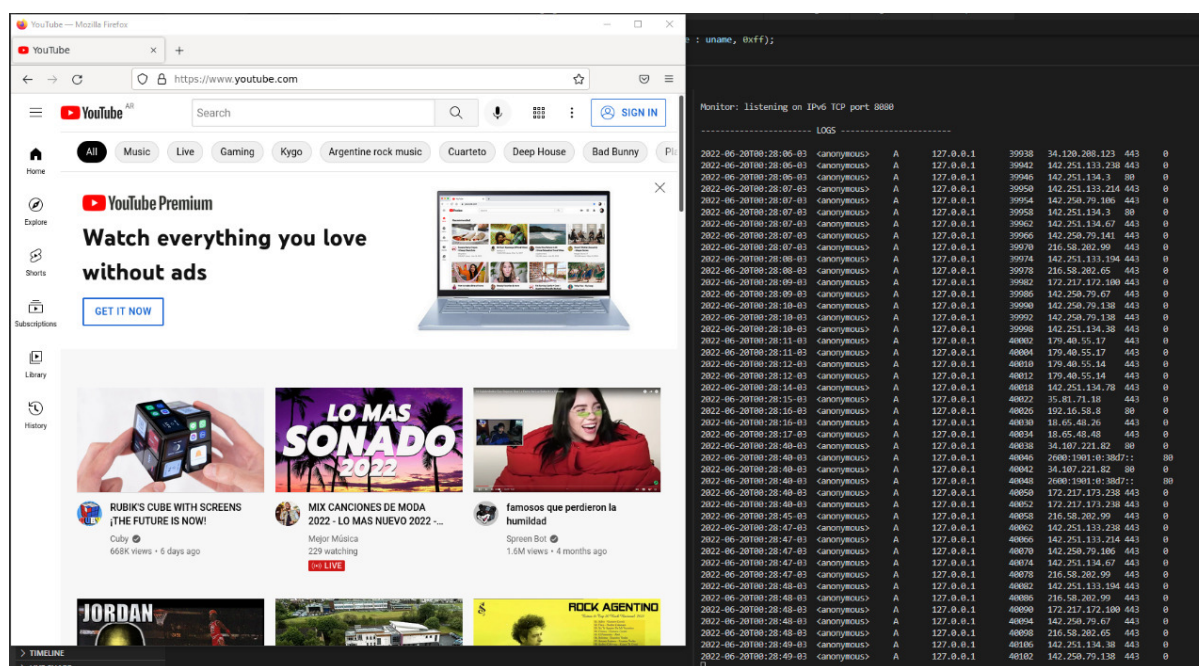
```
nico@COMPU-NICO:~/protos/socksv5-protocol$ dig +short tpe.proto.leak.com.ar
127.0.0.1
240.0.0.1
nico@COMPU-NICO:~/protos/socksv5-protocol$

nico@COMPU-NICO:~/protos/socksv5-protocol$ curl -x socks5h://localhost:1080 http://tpe.proto.leak.com.ar
curl: (7) Can't complete SOCKS5 connection to tpe.proto.leak.com.ar:80. (6)
nico@COMPU-NICO:~/protos/socksv5-protocol$
```

Como se puede ver en la respuesta SOCKSv5, el código devuelto es 6, que se corresponde con el de 'TTL Exceeded', indicando que se intentó llegar al host pedido pero no se pudo con el TTL predeterminado.

7.5 Uso con browser (Mozilla Firefox)

1. Conexión a un sitio web (https://www.youtube.com)



The image shows a Mozilla Firefox browser window displaying the YouTube homepage. The address bar shows 'https://www.youtube.com'. The page content includes the YouTube logo, search bar, and various video recommendations. To the right of the browser window, a terminal window is open, displaying network logs. The logs show multiple concurrent connections to the YouTube server, with the status 'A' (Active) and the IP address '127.0.0.1'. The logs also show the status 'E' (Error) for some connections, indicating a failure to establish a connection.

Como se puede ver en los Logs del server, al ingresar al sitio web de Youtube, se disparan múltiples conexiones concurrentes que se registran en dichos logs. En este caso de prueba se observaron alrededor de 20 conexiones concurrentes.

7.6 Uso del password disector

1. Conexión al proxy mediante POP3 usando dovecot

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ ./socks5d
Socks: listening on IPv4 TCP port 1080
Monitor: listening on IPv4 TCP port 8080
Socks: listening on IPv6 TCP port 1080
Monitor: listening on IPv6 TCP port 8080

----- LOGS -----
2022-06-21T03:55:12-03 <anonymous> A 127.0.0.1 54598 localhost 110 0
2022-06-21T03:55:26-03 <anonymous> P POP3 localhost 110 nico [REDACTED]
```

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ sudo service dovecot start
* Starting IMAP/POP3 mail server dovecot
 [ OK ]
nico@COMPU-NICO:~/protos/socksv5-protocol$ nc -C -X 5 -x localhost:1080 localhost 110
+OK Dovecot (Ubuntu) ready.
user nico
+OK
pass [REDACTED]
+OK Logged in.
```

Como se puede observar, el usuario se loguea al servidor mediante POP3 y, con el password disector activado, se imprimen las credenciales de user y pass ingresadas para realizar dicho login.

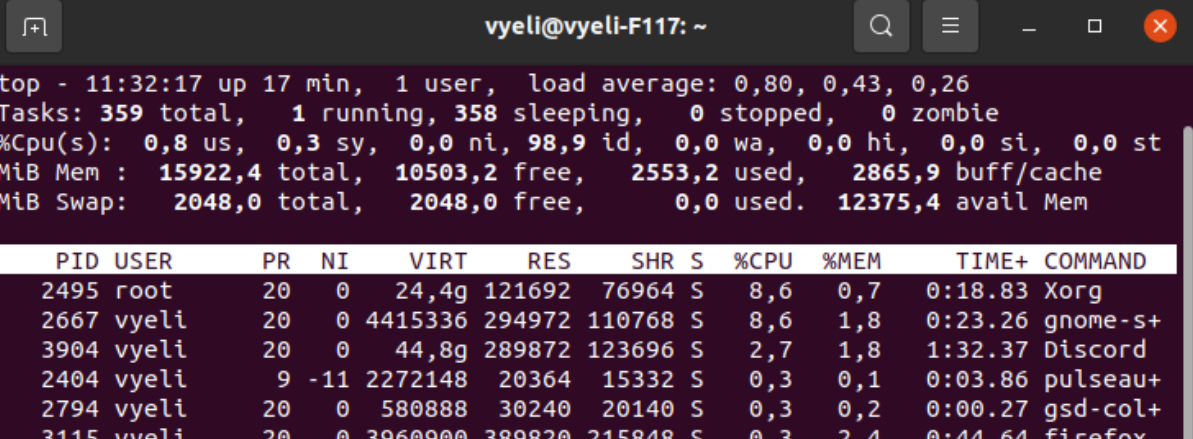
7.7 Prueba de estrés

La prueba de estrés consiste en utilizar este comando e ir aumentando la cantidad de terminales que lo corran

```
while true; do time curl -x socks5h://localhost:1080 http://212.183.159.230/100MB.zip  
--output - > /dev/null; done
```

Para ver el estado del uso de los recursos se optó por la aplicación de **top** que nos muestra la ocupación de la cpu y de la memoria durante esta prueba.

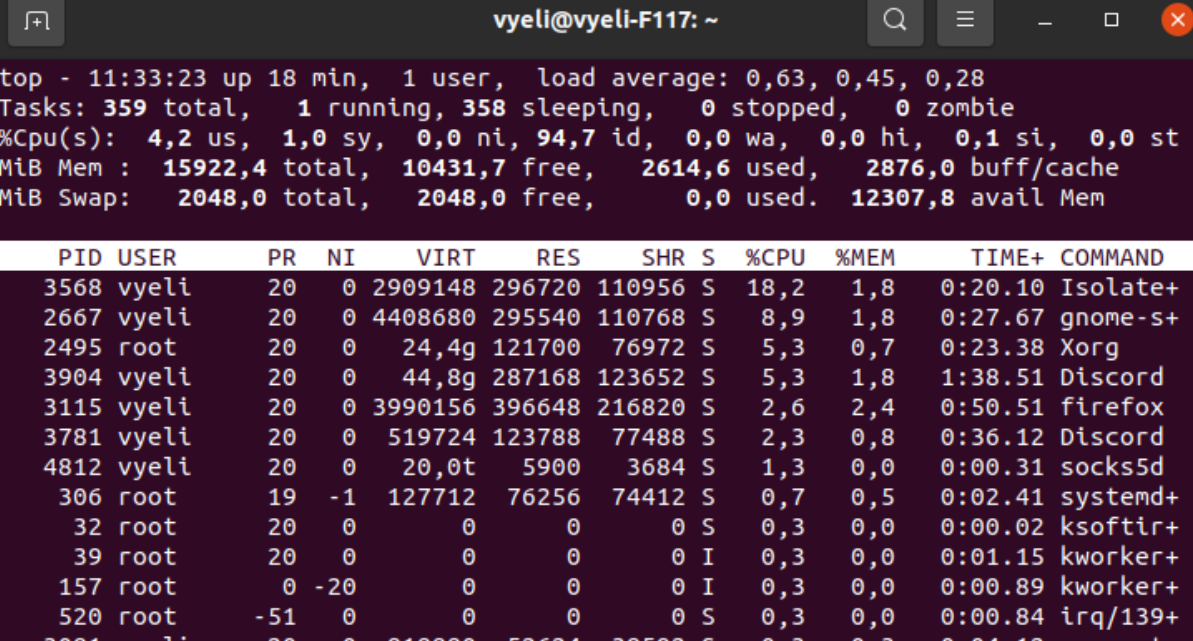
Antes:



```
top - 11:32:17 up 17 min, 1 user, load average: 0,80, 0,43, 0,26  
Tasks: 359 total, 1 running, 358 sleeping, 0 stopped, 0 zombie  
%Cpu(s): 0,8 us, 0,3 sy, 0,0 ni, 98,9 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st  
MiB Mem : 15922,4 total, 10503,2 free, 2553,2 used, 2865,9 buff/cache  
MiB Swap: 2048,0 total, 2048,0 free, 0,0 used. 12375,4 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2495	root	20	0	24,4g	121692	76964	S	8,6	0,7	0:18.83	Xorg
2667	vyeli	20	0	4415336	294972	110768	S	8,6	1,8	0:23.26	gnome-s+
3904	vyeli	20	0	44,8g	289872	123696	S	2,7	1,8	1:32.37	Discord
2404	vyeli	9	-11	2272148	20364	15332	S	0,3	0,1	0:03.86	pulseau+
2794	vyeli	20	0	580888	30240	20140	S	0,3	0,2	0:00.27	gsd-col+
3115	vyeli	20	0	3960900	389820	215848	S	0,3	2,4	0:44.64	firefox

1 terminal:



```
top - 11:33:23 up 18 min, 1 user, load average: 0,63, 0,45, 0,28  
Tasks: 359 total, 1 running, 358 sleeping, 0 stopped, 0 zombie  
%Cpu(s): 4,2 us, 1,0 sy, 0,0 ni, 94,7 id, 0,0 wa, 0,0 hi, 0,1 si, 0,0 st  
MiB Mem : 15922,4 total, 10431,7 free, 2614,6 used, 2876,0 buff/cache  
MiB Swap: 2048,0 total, 2048,0 free, 0,0 used. 12307,8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3568	vyeli	20	0	2909148	296720	110956	S	18,2	1,8	0:20.10	Isolate+
2667	vyeli	20	0	4408680	295540	110768	S	8,9	1,8	0:27.67	gnome-s+
2495	root	20	0	24,4g	121700	76972	S	5,3	0,7	0:23.38	Xorg
3904	vyeli	20	0	44,8g	287168	123652	S	5,3	1,8	1:38.51	Discord
3115	vyeli	20	0	3990156	396648	216820	S	2,6	2,4	0:50.51	firefox
3781	vyeli	20	0	519724	123788	77488	S	2,3	0,8	0:36.12	Discord
4812	vyeli	20	0	20,0t	5900	3684	S	1,3	0,0	0:00.31	socks5d
306	root	19	-1	127712	76256	74412	S	0,7	0,5	0:02.41	systemd+
32	root	20	0	0	0	0	S	0,3	0,0	0:00.02	ksoftir+
39	root	20	0	0	0	0	I	0,3	0,0	0:01.15	kworker+
157	root	0	-20	0	0	0	I	0,3	0,0	0:00.89	kworker+
520	root	-51	0	0	0	0	S	0,3	0,0	0:00.84	irq/139+
3081	vyeli	20	0	818880	52624	38592	S	0,3	0,3	0:04.12	gnome-t+

2 terminales:

```
vyeli@vyeli-F117: ~
top - 11:35:56 up 21 min, 1 user, load average: 0,35, 0,41, 0,28
Tasks: 356 total, 1 running, 355 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1,2 us, 0,8 sy, 0,0 ni, 97,8 id, 0,0 wa, 0,0 hi, 0,2 si, 0,0 st
MiB Mem : 15922,4 total, 10823,0 free, 2308,1 used, 2791,2 buff/cache
MiB Swap: 2048,0 total, 2048,0 free, 0,0 used. 12703,7 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2667	vyeli	20	0	4416392	296340	110768	S	5,6	1,8	0:37.53	gnome-s+
2495	root	20	0	24,4g	121896	77100	S	4,7	0,7	0:34.93	Xorg
4812	vyeli	20	0	20,0t	5908	3684	S	3,3	0,0	0:03.30	socks5d
3115	vyeli	20	0	3976120	397284	211264	S	1,3	2,4	1:01.12	firefox
520	root	-51	0	0	0	0	S	0,7	0,0	0:01.61	irq/139+
3478	vyeli	20	0	9095444	259512	108476	S	0,7	1,6	0:20.46	Isolate+
13	root	20	0	0	0	0	I	0,3	0,0	0:00.65	rcu_sch+
164	root	20	0	0	0	0	I	0,3	0,0	0:00.15	kworker+
535	root	0	-20	0	0	0	I	0,3	0,0	0:00.17	kworker+
3081	vyeli	20	0	819388	53184	38800	S	0,3	0,3	0:09.88	gnome-t+
3475	vyeli	20	0	2972620	415204	146628	S	0,3	2,5	0:42.50	Isolate+
3568	vyeli	20	0	2907040	243608	110896	S	0,3	1,5	0:22.95	Isolate+

3 terminales:

```
top - 11:36:51 up 22 min, 1 user, load average: 0,57, 0,46, 0,31
Tasks: 354 total, 1 running, 353 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0,9 us, 0,4 sy, 0,0 ni, 98,5 id, 0,0 wa, 0,0 hi, 0,1 si, 0,0 st
MiB Mem : 15922,4 total, 10790,8 free, 2338,5 used, 2793,1 buff/cache
MiB Swap: 2048,0 total, 2048,0 free, 0,0 used. 12672,2 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2667	vyeli	20	0	4416608	296408	110768	S	10,0	1,8	0:41.18	gnome-s+
2495	root	20	0	24,4g	121984	77188	S	8,3	0,7	0:39.57	Xorg
3478	vyeli	20	0	9095444	261872	108476	S	1,7	1,6	0:20.77	Isolate+
4812	vyeli	20	0	20,0t	5908	3684	S	1,7	0,0	0:04.40	socks5d
3115	vyeli	20	0	3974672	402460	213572	S	0,7	2,5	1:07.41	firefox
5599	vyeli	20	0	103728	11188	9856	S	0,7	0,1	0:00.17	curl
8	root	20	0	0	0	0	I	0,3	0,0	0:00.44	kworker+
157	root	0	-20	0	0	0	I	0,3	0,0	0:01.21	kworker+

4 terminales:

```
top - 11:37:45 up 22 min, 1 user, load average: 0,98, 0,61, 0,37
Tasks: 356 total, 3 running, 353 sleeping, 0 stopped, 0 zombie
%Cpu(s): 6,0 us, 5,0 sy, 0,0 ni, 87,9 id, 0,0 wa, 0,0 hi, 1,0 si, 0,0 st
MiB Mem : 15922,4 total, 10834,9 free, 2304,7 used, 2782,7 buff/cache
MiB Swap: 2048,0 total, 2048,0 free, 0,0 used. 12717,5 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4812	vyeli	20	0	20,0t	5908	3684	R	29,2	0,0	0:12.37	socks5d
3568	vyeli	20	0	2910112	258856	110896	S	23,3	1,6	0:25.59	Isolate+
520	root	-51	0	0	0	0	S	9,3	0,0	0:04.54	irq/139+
2667	vyeli	20	0	4416548	296400	110768	S	7,6	1,8	0:44.87	gnome-s+
2495	root	20	0	24,4g	122060	77264	S	6,3	0,7	0:44.61	Xorg
3115	vyeli	20	0	3964224	405136	211140	S	3,7	2,5	1:13.46	firefox
5686	vyeli	20	0	103728	11548	10188	S	2,0	0,1	0:00.25	curl
157	root	0	-20	0	0	0	I	1,7	0,0	0:01.46	kworker+
5680	vyeli	20	0	103728	11380	10024	S	1,7	0,1	0:00.25	curl

5 terminales:

```
top - 11:40:10 up 25 min,  1 user,  load average: 0,98, 0,72, 0,44
Tasks: 358 total,   3 running, 355 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2,6 us,  3,4 sy,   0,0 ni, 92,9 id,   0,0 wa,   0,0 hi,   1,0 si,   0,0 st
MiB Mem : 15922,4 total, 10814,0 free,  2323,6 used,  2784,8 buff/cache
MiB Swap:  2048,0 total,  2048,0 free,    0,0 used. 12697,9 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
  4812 vyeli    20   0   20,0t    5936   3684 S   30,6   0,0   0:46.09 socks5d
    520 root    -51   0        0        0        0 R    9,3   0,0   0:14.89 irq/139+
  3475 vyeli    20   0 2984956 393284 151280 S    6,0   2,4   1:09.20 Isolate+
  3115 vyeli    20   0 3972532 425652 223424 S    5,0   2,6   1:30.81 firefox
  2667 vyeli    20   0 4417108 296368 110768 S    4,0   1,8   0:53.08 gnome-s+
  2495 root     20   0   24,4g 122540 77744 S    3,0   0,8   0:55.14 Xorg
  5844 vyeli    20   0 103728   11412 10052 S    2,7   0,1   0:00.42 curl
    157 root      0 -20        0        0        0 I    2,0   0,0   0:02.70 kworker+
  5832 vyeli    20   0 103728   11140 9808 S    2,0   0,1   0:00.42 curl
```

Se pudo notar que no se colapsó el ./socks5d, y estuvo aumentando el uso del microprocesador y las velocidades de descarga en cada terminal fueron bajando a medida que aumentamos la cantidad.

Ahora probando localmente:

Configuramos nuestro servidor nginx para que sirva este archivo iso de ubuntu.

```
server {
    listen 80 default_server;

    server_name _;

    root /home/vyeli/Downloads;
    index ubuntu-20.04.4-desktop-amd64.iso;
    location / {

    }
}
```

while true; do time curl -x socks5h://localhost:1080 http://fooxd/ --output -> /dev/null; done

Corremos este comando que previamente hicimos que fooxd apunte a localhost en /etc/hosts

Después con service nginx restart

vemos que ahora no tenemos las limitaciones de ancho de banda.

1 terminal:

Podemos ver ya que el socks5d al no tener la limitación de ancho de banda se aprovecha bien los recursos y utiliza en este caso casi el 100 % de la cpu, mientras que no se colapsó.

```
vyeli@vyeli-F117: ~/i... x vyeli@vyeli-F117: ~/i... x vyeli@vyeli-F117: ~/it... x
```

```
top - 12:24:34 up 1:09, 1 user, load average: 1,50, 1,43, 1,15
Tasks: 384 total, 2 running, 382 sleeping, 0 stopped, 0 zombie
%Cpu(s): 9,1 us, 9,6 sy, 0,0 ni, 80,8 id, 0,1 wa, 0,0 hi, 0,4 si, 0,0 st
MiB Mem : 15922,4 total, 5416,1 free, 3578,4 used, 6927,9 buff/cache
MiB Swap: 2048,0 total, 2048,0 free, 0,0 used, 11238,2 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8064	vyeli	20	0	20,0t	6372	3788	R	99,7	0,0	0:26.87	socks5d
7036	vyeli	20	0	45,0g	405120	149888	S	24,9	2,5	12:58.11	Discord
9232	vyeli	20	0	103728	11568	10208	S	19,3	0,1	0:01.62	curl
2404	vyeli	9	-11	4107996	23544	17988	S	4,7	0,1	3:37.22	pulseau+
2495	root	20	0	24,5g	152592	97864	S	3,0	0,9	3:11.15	Xorg
3478	vyeli	20	0	9095460	268056	109516	S	3,0	1,6	0:56.45	Isolate+
9054	www-data	20	0	55884	5140	3448	S	2,7	0,0	0:01.33	nginx
2667	vyeli	20	0	4694896	302672	111544	S	1,7	1,9	2:25.74	gnome-s+
3115	vyeli	20	0	4235904	498116	230792	S	1,0	3,1	4:03.62	firefox
6771	vyeli	20	0	579092	155240	89540	S	1,0	1,0	0:51.20	Discord
306	root	19	-1	274820	178604	176716	S	0,3	1,1	0:10.91	systemd+
2824	vyeli	20	0	274260	27964	17860	S	0,3	0,2	0:00.42	nsd-wac+

2 terminales:

```
vyeli@vyeli-F117: ~/lba/20221c/Prot... x vyeli@vyeli-F117: ~/lba/20221c/Prot... x
```

```
top - 12:28:35 up 1:13, 1 user, load average: 1,47, 1,49, 1,24
Tasks: 385 total, 4 running, 381 sleeping, 0 stopped, 0 zombie
%Cpu(s): 9,7 us, 10,7 sy, 0,0 ni, 78,3 id, 0,0 wa, 0,0 hi, 1,4 st, 0,0 st
MiB Mem : 15922,4 total, 5314,3 free, 3672,8 used, 6935,3 buff/cache
MiB Swap: 2048,0 total, 2048,0 free, 0,0 used, 11145,8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8064	vyeli	20	0	20,0t	6388	3788	R	100,0	0,0	0:54.17	socks5d
9406	vyeli	20	0	103728	11540	10180	S	24,6	0,1	0:04.14	curl
9417	vyeli	20	0	103728	11496	10136	S	24,6	0,1	0:03.29	curl
7036	vyeli	20	0	45,0g	409576	150160	S	23,6	2,5	14:37.71	Discord
2495	root	20	0	24,5g	153360	98632	S	5,3	0,9	3:29.57	Xorg
2667	vyeli	20	0	4699684	302872	111676	R	4,7	1,9	2:39.69	gnome-s+
2404	vyeli	9	-11	4107996	23544	17988	S	4,3	0,1	3:53.74	pulseau+
9054	www-data	20	0	55884	5140	3448	S	2,3	0,0	0:02.00	nginx
306	root	19	-1	291204	192372	190472	S	0,7	1,2	0:12.01	systemd+
13	root	20	0	0	0	0	I	0,3	0,0	0:02.47	rcu_sch+
520	root	-51	0	0	0	0	S	0,3	0,0	0:27.27	irq/139+
1022	root	20	0	128800	9236	8408	S	0,3	0,1	0:00.35	thermald

```
vyeli@vyeli-F117: ~$ while true; do time curl -x socks5h://localhost:1080 http://fooxd/ --output -> /dev/null; done
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload	Upload	Total	Spent	Left
52	3222M	52	1700M	0	0	113M	0
				0	0:00:28	0:00:15	0:00:13
							114M

----- LOGS -----						
2022-06-21T12:06:15-03	<anonymous>	A	127.0.0.1	59664	localhos	
t 80 0						
2022-06-21T12:14:32-03	<anonymous>	A	127.0.0.1	59666	fooxd	8
0 0						
2022-06-21T12:18:34-03	<anonymous>	A	127.0.0.1	59668	fooxd	8
0 0						
2022-06-21T12:18:50-03	<anonymous>	A	127.0.0.1	59670	fooxd	8
0 0						
2022-06-21T12:19:28-03	<anonymous>	A	127.0.0.1	59672	fooxd	8
0 0						
2022-06-21T12:24:25-03	<anonymous>	A	127.0.0.1	59674	fooxd	8
0 0						
2022-06-21T12:28:17-03	<anonymous>	A	127.0.0.1	59676	fooxd	8
0 0						
2022-06-21T12:28:21-03	<anonymous>	A	127.0.0.1	59678	fooxd	8
0 0						
2022-06-21T12:28:44-03	<anonymous>	A	127.0.0.1	59680	fooxd	8
0 0						
2022-06-21T12:28:50-03	<anonymous>	A	127.0.0.1	59682	fooxd	8
0 0						

Podemos ver que soportó con dos terminales haciendo while true y descargando ese archivo de 3.3 GB de iso.

8. Instalación

Para construir el proyecto simplemente se debe correr el comando make/make all en el directorio principal del proyecto:

```
~/socksv5-protocol$ make all
```

En ese momento se generarán dos (2) ejecutables, ambos ubicados en el directorio raíz del proyecto como **socks5d**, que es el server proxy y **client** que es una implementación de cliente del protocolo de monitoreo mencionado anteriormente en el informe.

9. Instrucciones de configuración

Al momento de correr el servidor se debe setear una variable de entorno que contenga el token de 16 caracteres del administrador default del servidor, **root**. En caso de no tenerla seteada, el servidor informará tanto del error como el comando a correr para setearlo.

```
nico@COMPU-NICO:~/protos/socksv5-protocol$ ./socks5d
Socks: listening on IPv4 TCP port 1080
Monitor: listening on IPv4 TCP port 8080
Socks: listening on IPv6 TCP port 1080
Monitor: listening on IPv6 TCP port 8080
./socks5d: token not present, set it with 'export MONITOR_ROOT_TOKEN=<token>'.
nico@COMPU-NICO:~/protos/socksv5-protocol$
```

Dicho token debe ser luego usado para las operaciones de un cliente de monitoreo hasta que se agregue otro usuario administrador, pues es el único disponible al iniciar el servidor.

Luego, se puede configurar el estado inicial del servidor con una serie de opciones que se agregan al comando del servidor, las cuales son:

```
-l<SOCKS addr> Dirección donde servirá el proxy SOCKS. Por defecto escucha en todas las interfaces.
-N           Deshabilita los passwords disectors.
-L<conf addr> Dirección donde servirá el servicio de management. Por defecto escucha solo en loopback.
-p<SOCKS port> Puerto TCP para conexiones entrantes SOCKS. Por defecto es 1080.
-P<conf port> Puerto TCP para conexiones entrantes del protocolo de configuracion. Por defecto es 8080.
-u<user>:<pass> Usuario y contraseña de usuario que puede usar el proxy. Hasta 10.
```

Por otro lado, tenemos el cliente de monitoreo, que sirve para cambiar la configuración del servidor en tiempo de ejecución y es sencillo de usar. Al momento de correrlo, se le debe pasar como parámetro un token y, opcionalmente, ip destino y puerto destino, sino se tomarán los valores por defecto como ya se vió en la sección de descripción del cliente.

>> ./client [OPTIONS]... TOKEN [DESTINATION] [PORT].

En cuanto a las posibilidades de configuración con el cliente, dentro de las opciones que se pueden elegir encontramos 6 que pueden modificar el estado del servidor, las cuales son:

COMANDO	ARGUMENTO	DESCRIPCIÓN
-n	-	Enciende el disector pop3 de password en el server proxy
-N	-	Apaga el disector pop3 de password en el server proxy
-u	user:pass	Agrega un usuario del proxy con el nombre y contraseña del parámetro
-U	user:token	Agrega un admin de monitoreo con el nombre y token del parámetro
-d	user	Borra el usuario del proxy con el nombre del parámetro
-D	admin	Borra el admin con el nombre del parámetro. No puede borrar el admin default (root).

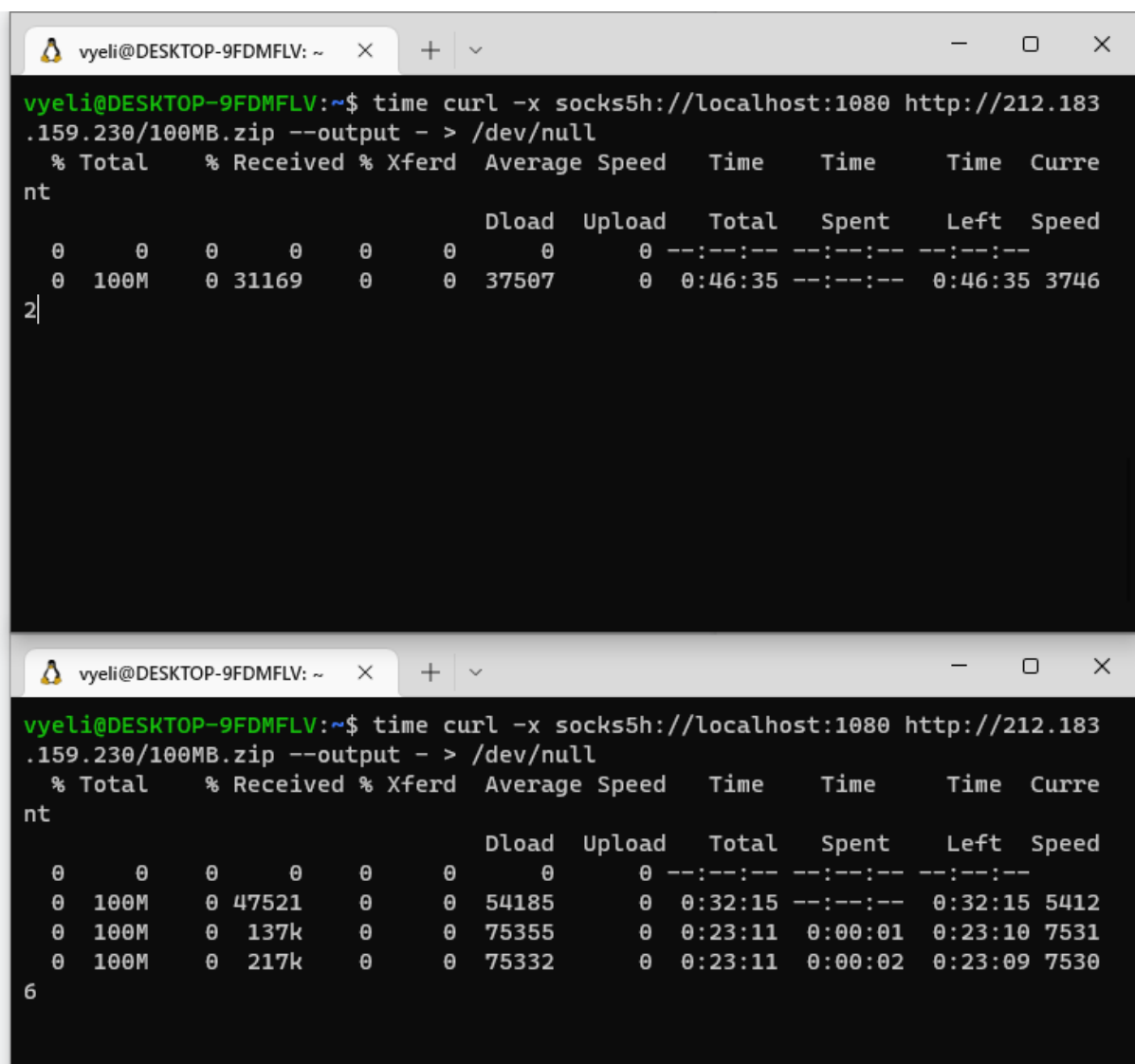
10. Ejemplos de configuración y monitoreo

Estos ejemplos están basados en que tenemos la variable de entorno del token del servidor como `'mybeautifultoken'`.

10.1 Cantidad de conexiones concurrentes e históricas del server

```
vyeli@DESKTOP-9FDMFLV:~$ time curl -x socks5h://localhost:1080 http://212.183.159.230/10MB.zip
--output - > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left     Speed
100 10.0M    100 10.0M    0     0   791k      0  0:00:12  0:00:12 --:--:-- 1670k

real    0m12.942s
user    0m0.029s
sys     0m0.030s
```



```
vyeli@DESKTOP-9FDMFLV:~$ time curl -x socks5h://localhost:1080 http://212.183
.159.230/100MB.zip --output - > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curre
nt
                                 Dload  Upload   Total   Spent    Left     Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
  0 100M    0 31169    0     0  37507      0  0:46:35 --:--:-- 0:46:35 3746
2|

vyeli@DESKTOP-9FDMFLV:~$ time curl -x socks5h://localhost:1080 http://212.183
.159.230/100MB.zip --output - > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curre
nt
                                 Dload  Upload   Total   Spent    Left     Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
  0 100M    0 47521    0     0  54185      0  0:32:15 --:--:-- 0:32:15 5412
  0 100M    0 137k    0     0  75355      0  0:23:11 0:00:01 0:23:10 7531
  0 100M    0 217k    0     0  75332      0  0:23:11 0:00:02 0:23:09 7530
6
```

```
vyeli@DESKTOP-9FDMFLV:~/proto/TPE/socksv5-protocol$ ./client -cC mybeautifultoken
The amount of concurrent connections is: 2
The amount of historic connections is: 3
```

En la primera imagen corrimos el comando **`time curl -x socks5h://localhost:1080 http://212.183.159.230/10MB.zip --output - > /dev/null`**

Y esperamos que termine.

Luego corrimos en dos terminales.

`time curl -x socks5h://localhost:1080 http://212.183.159.230/100MB.zip --output - > /dev/null`

Y antes de que terminen de descargar los 100 MB corrimos el cliente con los parámetros que nos devuelve la cantidad de conexiones concurrentes e históricas.

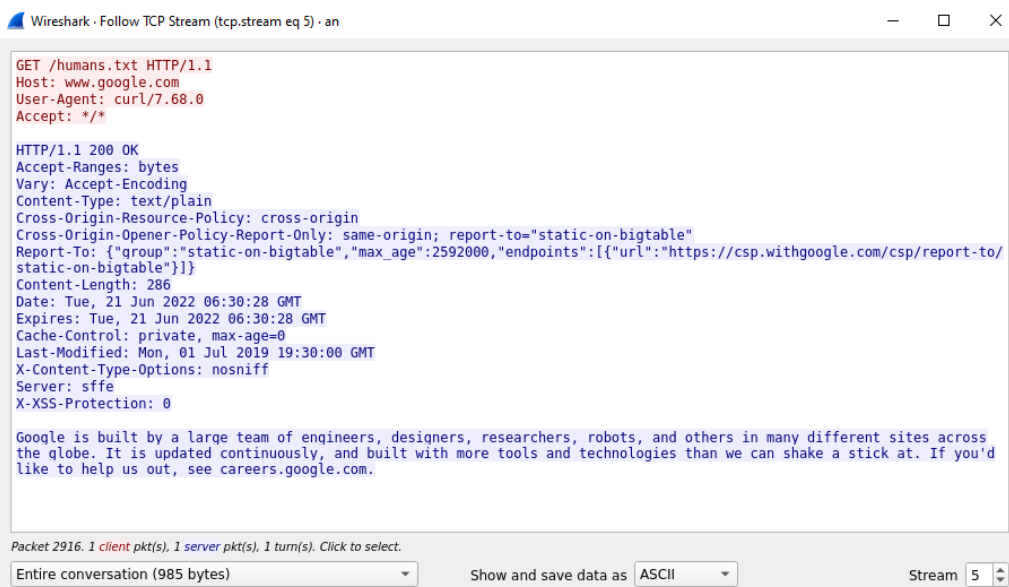
`./client -cC mybeautifultoken`

Donde se puede ver en la última imagen de que las conexiones concurrentes son 2 (las dos de descarga de 100MB) y 3 conexiones históricas (las 2 de descarga de 100 MB y la de la imagen 1 de descarga de 10 MB).

10.2 Cantidad de bytes transferidos

```
vyeli@DESKTOP-9FDMFLV:~$ curl -x socks5h://localhost:1080 http://www.google.com/humans.txt
Google is built by a large team of engineers, designers, researchers, robots, and others in many different sites across the globe. It is updated continuously, and built with more tools and technologies than we can shake a stick at. If you'd like to help us out, see careers.google.com.
vyeli@DESKTOP-9FDMFLV:~$
```

```
vyeli@DESKTOP-9FDMFLV:~/proto/TPE/socksv5-protocol$ ./client -b mybeautifultoken
The amount of transferred bytes is: 985
```



En este caso, pedimos por curl el texto de humans.txt a google, y corriendo el cliente con:
./client -b mybeautifultoken

Nos devuelve la cantidad de bytes que se transfirió entre el cliente con el servidor de google durante el estado de copy que fue de 985 bytes, que se verifica también en el wireshark.

10.3 Añadir un usuario del proxy e imprimir la lista de usuarios del proxy

```
vyeli@DESKTOP-9FDMFLV:~/proto/TPE/socksv5-protocol$ ./client -u foo:bar mybeautifultoken
The proxy user: 'foo' is now added to the server
```

Con esto agregamos al usuario foo con la contraseña bar a nuestra lista de usuarios que pueden usar el proxy.

```
vyeli@DESKTOP-9FDMFLV:~$ curl -x socks5h://localhost:1080 -U foo:bar http://www.google.com/humans.txt
Google is built by a large team of engineers, designers, researchers, robots, and others in many different sites across the globe. It is updated continuously, and built with more tools and technologies than we can shake a stick at. If you'd like to help us out, see careers.google.com.
```

Notemos que este usuario que acabamos de agregar puede usar el proxy socks5 con autenticación.

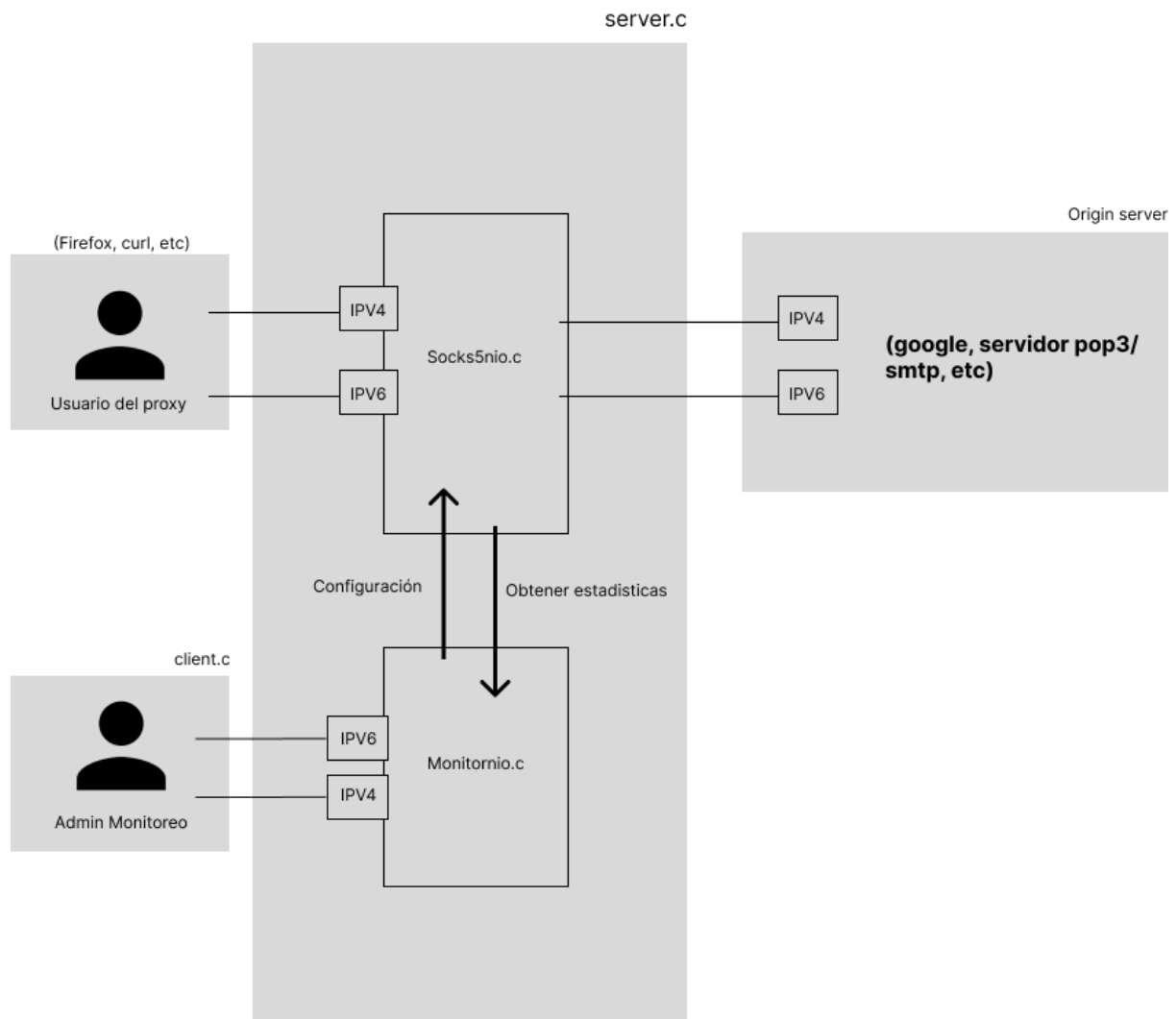
Ahora si le pasamos la contraseña incorrecta nos devuelve el error de que fue rechazado por el servidor de socks5.

```
vyeli@DESKTOP-9FDMFLV:~$ curl -x socks5h://localhost:1080 -U foo:xd http://www.google.com
curl: (7) User was rejected by the SOCKS5 server (1 1).
```

```
vyeli@DESKTOP-9FDMFLV:~/proto/TPE/socksv5-protocol$ ./client -a mybeautifultoken
Printing proxy user list:
foo
```

Y también podemos ver la lista de los usuarios que pueden usar el proxy con **`./client -a mybeautifultoken`**

11. Diseño del proyecto



12. Anexo

Tamaño buffer	Intento	Intento	Intento	Intento	Intento	
128 bytes	1	2	3	4	5	promedio
real	47,96	47,433	51,064	51,882	50,541	49,776
user	1,235	0,821	1,17	1,056	1,216	1,0996
sys	1,942	2,341	2,319	2,496	2,208	2,2612

Tabla 1. Tiempos con comando 'time' descargando un archivo de 10MB con un buffer de tamaño 128.

Tamaño buffer	Intento	Intento	Intento	Intento	Intento	
256 bytes	1	2	3	4	5	promedio
real	27,007	27,316	26,326	25,733	27,735	26,8234
user	0,51	0,57	0,557	0,525	0,607	0,5538
sys	1,157	1,101	1,131	1,143	1,23	1,1524

Tabla 2. Tiempos con comando 'time' descargando un archivo de 10MB con un buffer de tamaño 256.

Tamaño buffer	Intento	Intento	Intento	Intento	Intento	
512 bytes	1	2	3	4	5	promedio
real	17,013	17,561	17,198	18,358	16,767	17,3794
user	0,338	0,235	0,261	0,455	0,282	0,3142
sys	0,483	0,615	0,565	0,508	0,564	0,547

Tabla 3. Tiempos con comando 'time' descargando un archivo de 10MB con un buffer de tamaño 512.

Tamaño buffer	Intento	Intento	Intento	Intento	Intento	
1024 bytes	1	2	3	4	5	promedio
real	11,983	11,358	13,42	12,165	13,195	12,4242
user	0,024	0,02	0,046	0,012	0,025	0,0254
sys	0,037	0,041	0,016	0,049	0,038	0,0362

Tabla 4. Tiempos con comando 'time' descargando un archivo de 10MB con un buffer de tamaño 1024.

Tamaño buffer	Intento	Intento	Intento	Intento	Intento	
2048 bytes	1	2	3	4	5	promedio
real	12,888	11,186	12,603	12,165	14,056	12,5796
user	0,024	0,031	0,03	0,012	0,009	0,0212
sys	0,037	0,031	0,031	0,049	0,056	0,0408

Tabla 5. Tiempos con comando 'time' descargando un archivo de 10MB con un buffer de tamaño 2048.

Tamaño buffer	Intento	Intento	Intento	Intento	Intento	
4096 bytes	1	2	3	4	5	promedio
real	14,082	13,598	11,21	13,747	10,726	12,6726
user	0	0	0	0	0,042	0,0084
sys	0,044	0,061	0,044	0,06	0	0,0418

Tabla 6. Tiempos con comando 'time' descargando un archivo de 10MB con un buffer de tamaño 4096.

Tamaño buffer	Intento	Intento	Intento	Intento	Intento	
8192 bytes	1	2	3	4	5	promedio
real	12,686	13,598	14,689	13,818	11,712	13,3006
user	0	0	0,016	0,045	0,024	0,017
sys	0,05	0,061	0,032	0	0,025	0,0336

Tabla 7. Tiempos con comando 'time' descargando un archivo de 10MB con un buffer de tamaño 8192.