

Trabajo Práctico Cuatrimestral - Grupo X

Mizrahi, Thomas (60154)

tmizrahi@itba.edu.ar

Chayer, Iván (61360)

ichayer@itba.edu.ar

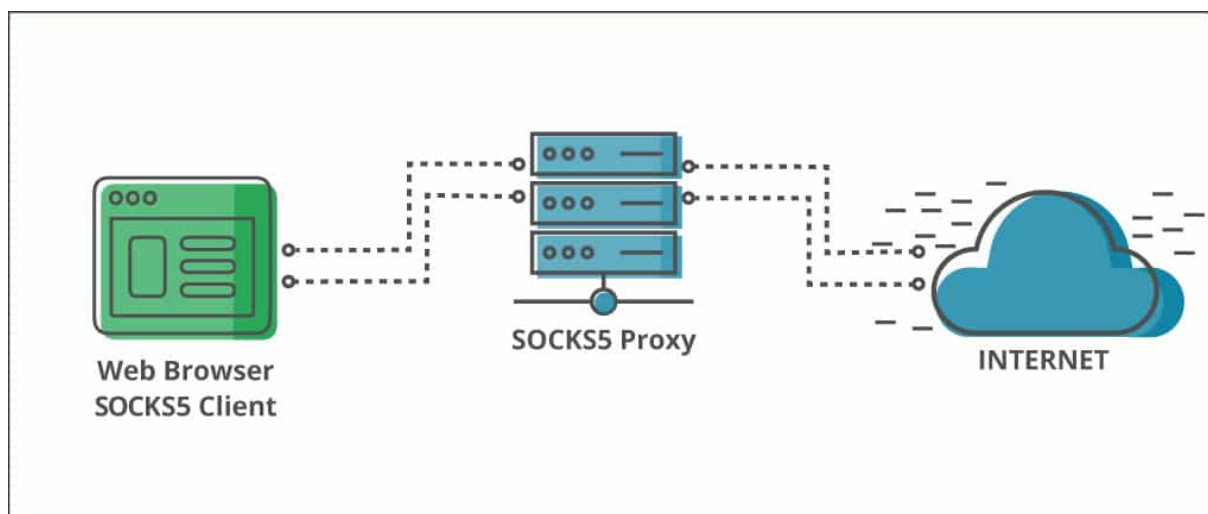
Di Toro, Camila (62576)

cditoro@itba.edu.ar

Catino, Kevin (61643)

kcatino@itba.edu.ar

Implementación de un servidor proxy socks5



Profesores:

- Marcelo Fabio Garberoglio
- Juan F. Codagnone
- Sebastian Kulesz

72.07 Protocolos de Comunicación
Instituto Tecnológico de Buenos Aires
Segundo cuatrimestre 2022

Abstracto

En el presente documento, se describe el desarrollo del Trabajo Práctico Especial para la materia Protocolos de Comunicación. La aplicación desarrollada es un servidor proxy que implementa el protocolo SOCKSv5 y sus funcionalidades asociadas, junto con un protocolo adicional que permite administrar y obtener información relevante del servidor.

Índice

Abstracto	2
Descripción detallada de los protocolos y aplicaciones desarrolladas	5
Negociación	5
Password-dissector	5
Métricas de uso	5
Logger	5
Problemas encontrados durante el diseño y la implementación	6
Limitaciones de la aplicación	7
Posibles extensiones	7
Ejemplos de prueba	7
Pruebas para determinar un tamaño de buffer adecuado	7
Buffer de 4K	7
Buffer de 8K	9
Buffers de 16K, 32K y 64K	10
Test de carga	11
Tiempo de respuesta	12
Throughput de bytes	13
Conclusiones	14
Verificación de la integridad de los datos	14
Uso como proxy de Chrome	15
Conexión a un origin server que no presta servicio (IPv4 e IPv6)	15
Verificación del funcionamiento del password dissector	16
Guía de instalación	17
Servidor proxy SOCKS5	17
Cliente de monitoreo	17
Instrucciones para la configuración	18
Servidor proxy SOCKS5	18
Cliente de monitoreo	18
Protocolo de monitoreo	19
Estados de una sesión	19
Estado de autorización	20
Estado de transacción	20
USERS	21
ADD USER	21
DELETE USER	23
CHANGE PASSWORD	23
CHANGE ROLE	24
GET DISSECTOR STATUS	25
SET DISSECTOR STATUS	26

GET AUTHENTICATION STATUS	27
SET AUTHENTICATION STATUS	27
STATISTICS	28
Ejemplos de configuración y monitoreo	30
Uso del token	30
USERS	30
ADD-USER	31
DELETE-USER	32
CHANGE PASSWORD	32
CHANGE ROLE	33
GET DISSECTOR STATUS	33
SET DISSECTOR STATUS	34
GET AUTHENTICATION STATUS	34
SET AUTHENTICATION STATUS	35
STATISTICS	35
Conclusiones	36

Descripción detallada de los protocolos y aplicaciones desarrolladas

En primer lugar, se diseñó y desarrolló un protocolo aplicado a un servidor no bloqueante que atiende conexiones IPv4 e IPv6 de manera multiplexada. El mismo es una versión con esteroides de SOCKS5 ([RFC 1928](#)) que permite, en principio, autenticar por usuario y contraseña, así también como interceptar nombres de usuario y contraseñas que viajen a través del protocolo POP3.

En segundo lugar, se diseñó un protocolo de monitoreo que permite consultar datos históricos como la cantidad de conexiones, bytes transferidos, etc. Para leer más detalles sobre el protocolo, referirse a la sección [protocolo de monitoreo](#).

A continuación haremos un desarrollo de los módulos llevados a cabo para el protocolo basado en SOCKS5:

Negociación

Al iniciar una conexión al proxy socks, el cliente inicia con una etapa de negociación en la que se define el método de autenticación a utilizar. La implementación soporta el uso de usuario y contraseña según lo definido en el protocolo [RFC 1929](#), así también como que el usuario se conecte sin autenticarse.

Mediante el protocolo de monitoreo, es posible establecer si los usuarios deberán autenticarse o no. Cuando el servidor comienza, la autenticación de los clientes es obligatoria. Luego puede desactivarse en caso de ser necesario utilizando el protocolo de monitoreo.

Password-dissector

El *password dissector* intercepta el tráfico en el puerto 110 para capturar credenciales del protocolo POP3. Este se puede activar o desactivar haciendo uso del protocolo de monitoreo. El *sniffing* se mantendrá en la comunicación si el primer mensaje es del servidor y el primer carácter es un "+". Para probar esto, usamos *ncat* y *dovecot*. Información sobre el testeo de esta funcionalidad será desarrollada más adelante en la sección de [Verificación del funcionamiento del password dissector](#).

Métricas de uso

Se implementó una interfaz de recolección de métricas **volátiles** del uso del proxy. Se cuentan cantidad de bytes transferidos, junto con la cantidad total de conexiones históricas, el máximo histórico de conexiones concurrentes, y la cantidad de conexiones actualmente abiertas. Estas métricas pueden ser accedidas a través del protocolo de monitoreo y un **administrador** que haga uso de él.

Logger

El logger fue implementado tanto para el desarrollo del proxy como para la versión final del mismo.

```
typedef enum {  
    LOG_DEBUG = 0,  
    LOG_INFO,  
    LOG_OUTPUT,  
    LOG_WARNING,  
    LOG_ERROR,  
    LOG_FATAL  
} TLogLevel;
```

Este puede imprimir en salida estándar y también a archivos para que los eventos que se quieren *trackear* persistan. En ambos casos, es posible definir un determinado nivel de logging para observar distintos datos. Por ejemplo, el **modo DEBUG** posee una gran cantidad de logs que resultan de utilidad cuando se presenta algún problema y queremos rastrear lo ocurrido.

Particularmente **durante la implementación del proxy**, se utilizó el **modo INFO**, que posee menos logs que DEBUG, pero que consideramos útil observar a lo largo de las pruebas realizadas para verificar el correcto funcionamiento. El **modo OUTPUT** es el modo con el que se realiza la entrega, que posee únicamente los *logs* solicitados.

El logger cuenta con un **define DISABLE_LOGGER**, el cual se encuentra comentado. Este define permite desactivar completamente el sistema de logging. Fue utilizado en pruebas de medición de tiempo, donde no queríamos que el tiempo que lleva realizar los logs interfiera en los resultados.

Los logs del servidor se generan en la carpeta log ubicada en la raíz.

Problemas encontrados durante el diseño y la implementación

Aunque las suscripciones a lectura/escritura fueron explicadas en clase, durante la implementación cometimos más de una vez errores relacionados con esto. Haciendo diferentes tipos de pruebas, algunas de ellas también realizadas en clase, pudimos identificar los errores cometidos. Algunos fueron identificados por mal funcionamiento, otros por el alto overhead que suponía el pasaje del tráfico a través del proxy. Quedó claro que las cosas terminan de entenderse en mucha mayor profundidad una vez que pasamos a la implementación.

Otro de los problemas que tuvimos fue comprender el funcionamiento del selector y de la máquina de estados. El código se encuentra muy bien documentado y también fue explicado. Sin embargo, cometimos errores por no estarlos utilizando correctamente.

Limitaciones de la aplicación

Debido al uso de la función *select()* para la elección de descriptores de archivo que se encuentran libres para una operación de lectura o escritura, la cantidad de clientes concurrentes del servidor (incluyendo tanto clientes del servidor proxy de SOCKS5 como de la aplicación de monitoreo) tiene un límite de 500 usuarios concurrentes aproximadamente.

Por otro lado, se definió un límite de 100 usuarios en el registro del servidor, y cada uno de los usuarios registrados debe tener credenciales (nombre de usuario y contraseña) que no superen los 31 bytes cada una.

Posibles extensiones

Una primera extensión que podemos implementar se relaciona con el manejo de usuarios. En vez de utilizar un archivo para administrarlos, podemos integrar una base de datos y el comportamiento no cambiaría.

Otra funcionalidad que podríamos implementar es la del *timeout*. Al mantener conexiones persistentes, en el proxy podrían mantenerse ocupados muchos *file descriptors* por conexiones ociosas.

Por último, algo interesante que podría llevarse a cabo, es un blacklisting de IPs/dominios. De esta forma, estaríamos simulando un proxy que filtra el acceso a ciertos sitios. Junto con esto, podríamos agregar métricas que indiquen la cantidad de pedidos que fueron denegados o “filtrados”.

Ejemplos de prueba

Pruebas para determinar un tamaño de buffer adecuado

Aclaración: todos los requests realizados durante esta prueba utilizando el comando *curl*, fueron ejecutadas reiteradas veces hasta conseguir que los tiempos se estabilicen. Se consideró que los tiempos eran estables al conseguir resultados similares en 3 ejecuciones consecutivas.

Buffer de 4K

Al comenzar el desarrollo, el buffer utilizado era de 4K. Sin embargo, una vez avanzado, la performance del proxy no era óptima.

Se muestra a continuación una comparación del tiempo que tardaba un *curl* en realizar un request de un archivo de casi 5GBs pasando a través del proxy (Figura 1) y no pasando a través de él (Figura 2):

```

camila@camila-VivoBook:~$ time ALL_PROXY=socks5h://127.0.0.1:1080 curl http://bar > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 4768M  100 4768M    0     0  570M      0  0:00:08  0:00:08 --:--:-- 574M

real    0m8,405s
user    0m0,760s
sys      0m1,101s
camila@camila-VivoBook:~$ time ALL_PROXY=socks5h://127.0.0.1:1080 curl http://bar > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 4768M  100 4768M    0     0  563M      0  0:00:08  0:00:08 --:--:-- 572M

real    0m8,483s
user    0m0,784s
sys      0m1,228s
camila@camila-VivoBook:~$ time ALL_PROXY=socks5h://127.0.0.1:1080 curl http://bar > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 4768M  100 4768M    0     0  544M      0  0:00:08  0:00:08 --:--:-- 560M

real    0m8,792s
user    0m0,841s
sys      0m1,378s

```

Figura 1: Request a través del proxy utilizando un buffer de 4K.

```

camila@camila-VivoBook:~$ time curl http://bar > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 4768M  100 4768M    0     0 3120M      0  0:00:01  0:00:01 --:--:-- 3118M

real    0m1,554s
user    0m0,453s
sys      0m1,097s
camila@camila-VivoBook:~$ time curl http://bar > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 4768M  100 4768M    0     0 3056M      0  0:00:01  0:00:01 --:--:-- 3054M

real    0m1,585s
user    0m0,481s
sys      0m1,105s
camila@camila-VivoBook:~$ time curl http://bar > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 4768M  100 4768M    0     0 3060M      0  0:00:01  0:00:01 --:--:-- 3060M

real    0m1,586s
user    0m0,506s
sys      0m1,064s

```

Figura 2: Request sin pasar por el proxy.

Con estas imágenes, vemos que el curl tarda **casi 6 veces más**, cuando se esperaba que esté cerca de 2 veces más. Utilizando el comando *perf* para correr el proyecto, se generó un reporte que indica en qué funciones está la mayor cantidad de tiempo el programa, considerando también las de kernel. Los resultados obtenidos para un buffer de 4K fueron los siguientes:

Samples: 57K of event 'cycles', Event count (approx.): 39368533427			
Overhead	Command	Shared Object	Symbol
10,62%	socks5v	[kernel.kallsyms]	[k] syscall_return_via_sysret
9,74%	socks5v	[kernel.kallsyms]	[k] syscall_exit_to_user_mode
8,84%	socks5v	[kernel.kallsyms]	[k] copy_user_enhanced_fast_string
5,56%	socks5v	[kernel.kallsyms]	[k] entry_SYSCALL_64_after_hwframe
4,23%	socks5v	[kernel.kallsyms]	[k] __entry_text_start
2,55%	socks5v	[kernel.kallsyms]	[k] clear_page_erms
2,37%	socks5v	[kernel.kallsyms]	[k] _raw_spin_lock_irqsave

Figura 3: Resultados de utilizar *perf* con un buffer de 4K.

Dado que es un único request de un archivo pesado, se esperaba que el programa pasara una gran parte del tiempo copiando texto. Sin embargo, las funciones que más tiempo consumieron fueron *syscall_return_via_sysret* y *syscall_exit_to_user_mode*. A pesar de que no fue determinado con exactitud qué hacen estas funciones del kernel, por su nombre parecería indicar que el programa está más tiempo yendo del *kernel space* al *user space* que efectivamente haciendo las copias del texto.

Buffer de 8K

Los resultados anteriores, fueron comparados con los obtenidos en una nueva experiencia, esta vez, con un buffer de 8K. Estos fueron los resultados:

```
camila@camila-VivoBook:~$ time ALL_PROXY=socks5h://127.0.0.1:1080 curl http://bar > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left  Speed
100 4768M  100 4768M    0     0  934M      0  0:00:05  0:00:05 --:--:-- 942M

real    0m5,142s
user    0m0,727s
sys     0m1,205s
camila@camila-VivoBook:~$ time ALL_PROXY=socks5h://127.0.0.1:1080 curl http://bar > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left  Speed
100 4768M  100 4768M    0     0  875M      0  0:00:05  0:00:05 --:--:-- 876M

real    0m5,476s
user    0m0,743s
sys     0m1,173s
camila@camila-VivoBook:~$ time ALL_PROXY=socks5h://127.0.0.1:1080 curl http://bar > /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left  Speed
100 4768M  100 4768M    0     0  868M      0  0:00:05  0:00:05 --:--:-- 861M

real    0m5,532s
user    0m0,776s
sys     0m1,224s
```

Figura 4: Request a través del proxy utilizando un buffer de 8K.

Nuevamente, puede verse que el tiempo es casi 4 veces más que el obtenido en la Figura 1. Sin embargo, analizemos los resultados de ejecutar *perf* para el buffer de 8K:

12,49%	socks5v	[kernel.kallsyms]	[k] copy_user_enhanced_fast_string
8,63%	socks5v	[kernel.kallsyms]	[k] syscall_return_via_sysret
7,72%	socks5v	[kernel.kallsyms]	[k] syscall_exit_to_user_mode
4,56%	socks5v	[kernel.kallsyms]	[k] entry_SYSCALL_64_after_hwframe
3,70%	socks5v	[kernel.kallsyms]	[k] clear_page_erms
3,50%	socks5v	[kernel.kallsyms]	[k] __entry_text_start
2,04%	socks5v	[kernel.kallsyms]	[k] _raw_spin_lock_irqsave
1,90%	socks5v	[kernel.kallsyms]	[k] __check_object_size
1,67%	socks5v	[kernel.kallsyms]	[k] read_tsc
1,61%	socks5v	[kernel.kallsyms]	[k] sock_poll
1,58%	socks5v	[kernel.kallsyms]	[k] copy_user_generic_unrolled

Figura 5: Resultados de utilizar *perf* con un buffer de 8K.

Como muestra la Figura 5 el primer puesto se lo lleva la función *copy_user_enhanced_fast_string* que sí parece ser una función que está copiando texto.

Buffers de 16K, 32K y 64K

Las mismas pruebas fueron realizadas para buffers de 16K, 32K y 64K. Para no extender la explicación, se resumen los resultados en la siguiente tabla:

Tamaño de buffer (K)	Tiempo de ejecución (s)	Tiempo relativo al curl sin proxy (s)	Función que más tiempo de ejecución utilizó	Porcentaje de tiempo
4	8,56	5,7	<i>syscall_return_via_sysret</i>	10,62%
8	5,3833	3,5888	<i>copy_user_enhanced_fast_string</i>	12,49%
16	3,9273	2,6182	<i>copy_user_enhanced_fast_string</i>	17,29%
32	2,7283	1,8188	<i>copy_user_enhanced_fast_string</i>	22,31%
64	2,341	1,5606	<i>copy_user_enhanced_fast_string</i>	27,81%

Tabla 1: Resumen de las pruebas con los distintos tamaños de buffers.

Algunas aclaraciones:

- El tiempo de ejecución se considera como el promedio de 3 ejecuciones estables (el concepto de estable siendo el definido al [comienzo de esta sección](#)).
- El tiempo relativo al *curl* sin proxy es la división entre el tiempo de ejecución con el proxy y el tiempo de ejecución sin el proxy.
- El tiempo de ejecución sin el proxy fue de 1,5s.
- El porcentaje corresponde al porcentaje del tiempo de ejecución utilizado por la función indicada en la columna anterior.

Estos datos nos llevaron a descartar el buffer de 4K como una solución posible, los resultados con el buffer de 64K mejoraron, pero no con una diferencia tan marcada como los anteriores.

Decidimos evaluar estos datos en conjunto con los tests de carga para determinar un resultado adecuado, basándonos en la hipótesis de que un mayor tamaño de buffer puede mejorar significativamente la performance para un request de gran tamaño por más que también pueda empeorar (menos significativamente) para muchos requests pequeños.

Test de carga

Se utilizó la herramienta *JMeter* para medir la respuesta del servidor a una carga de usuarios concurrentes.

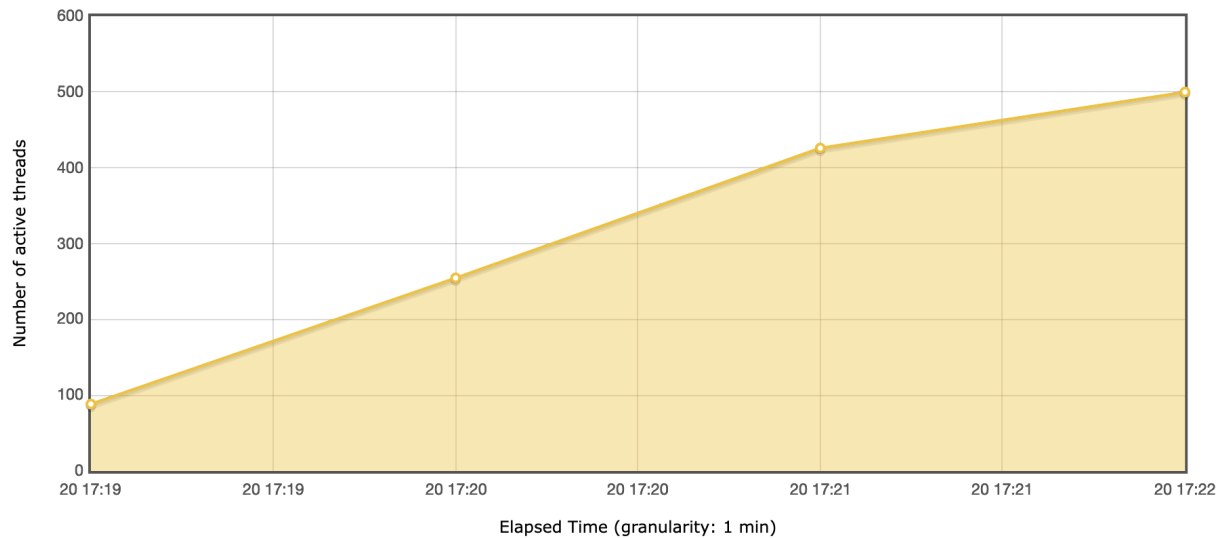


Figura 6: Cantidad de usuarios concurrentes en función del tiempo

Como se puede ver en la Figura 6, la cantidad de usuarios concurrentes aumenta de forma lineal a lo largo de 3 minutos. Durante este transcurso del tiempo, cada hilo realizó un pedido de un archivo con tamaño en el orden de los cientos de bytes que se *hosteaba* en un servidor local *Nginx*. Este proceso se realizó primero sin el proxy de por medio, y luego a través del proxy.

Tiempo de respuesta

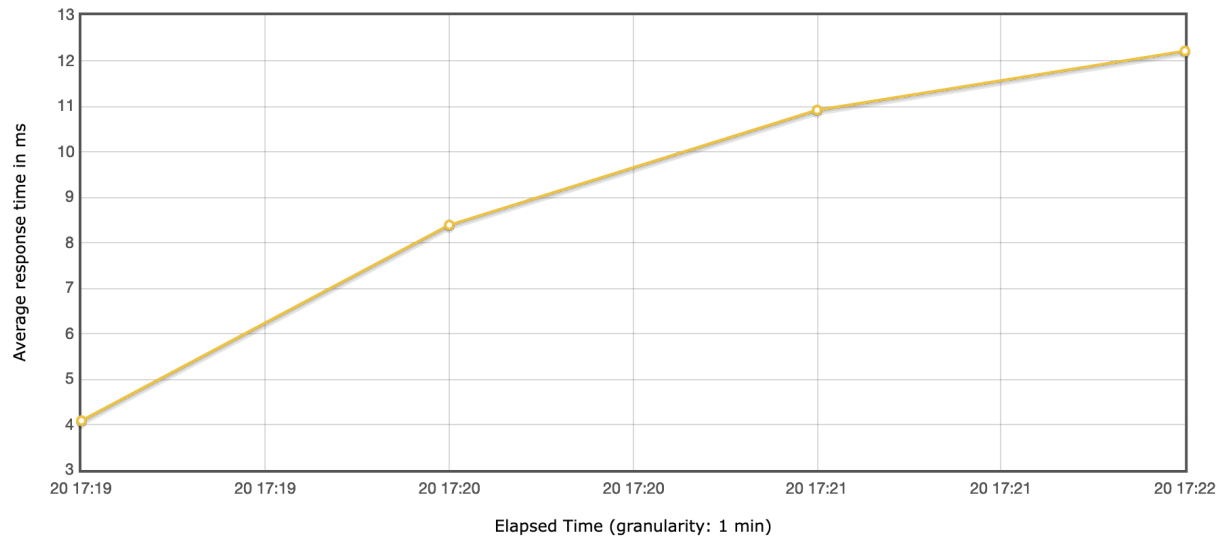


Figura 7: Tiempo de respuesta sin el proxy

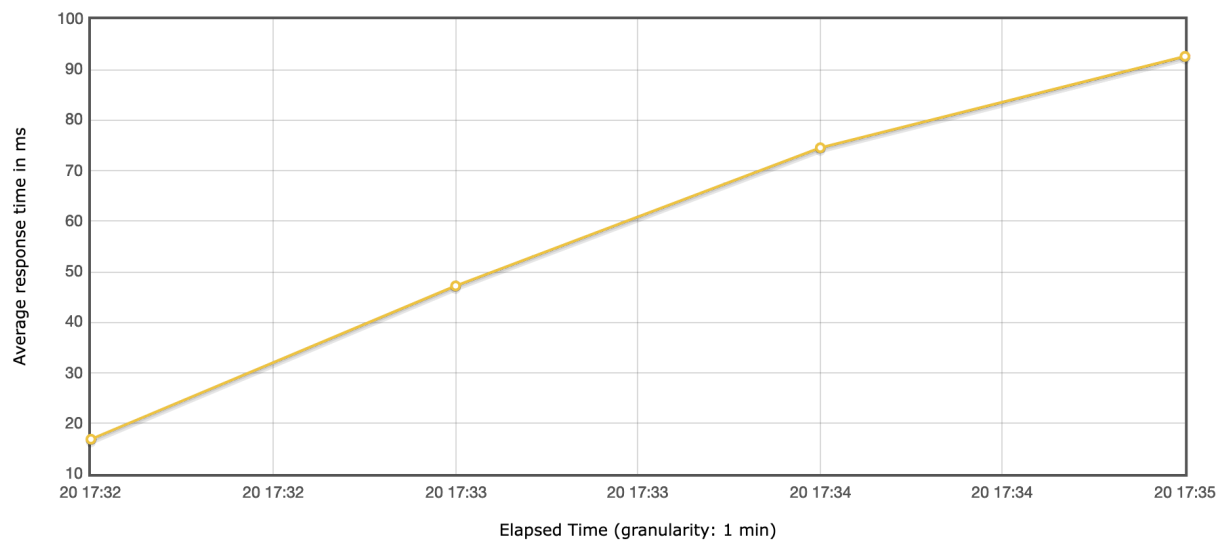


Figura 8: Tiempo de respuesta con el proxy y buffer de 8KB.

Notar: Ambos gráficos tienen escalas distintas en el eje vertical

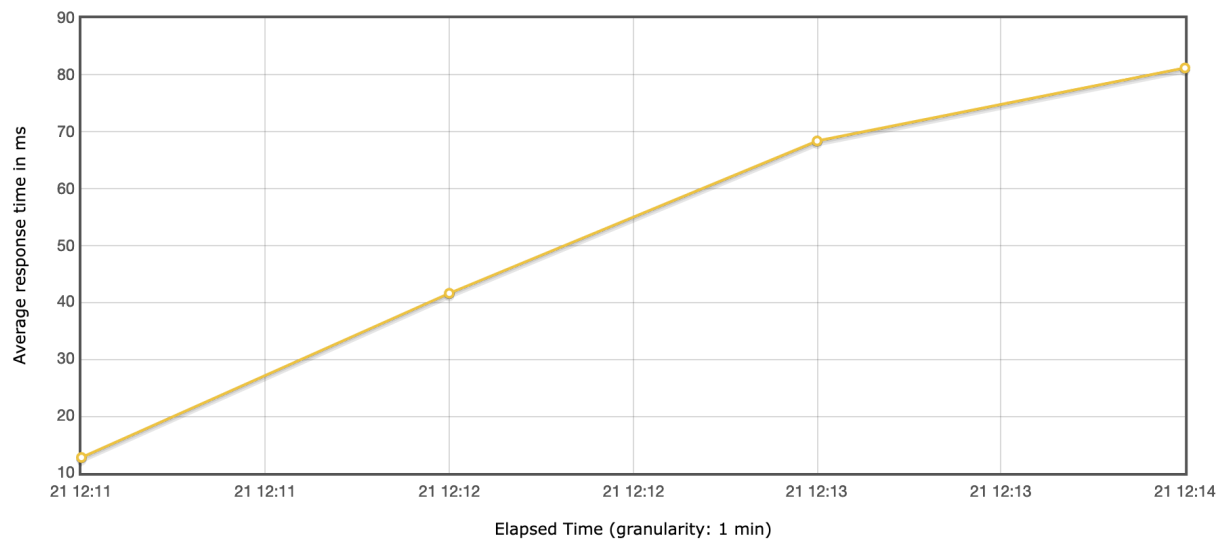


Figura 9: Tiempo de respuesta con el proxy y buffer de 32KB.

Throughput de bytes

En las siguientes imagenes, se muestra en color azul el *throughput* proveniente del cliente, mientras que en amarillo se observa el *throughput* proveniente del servidor origen.

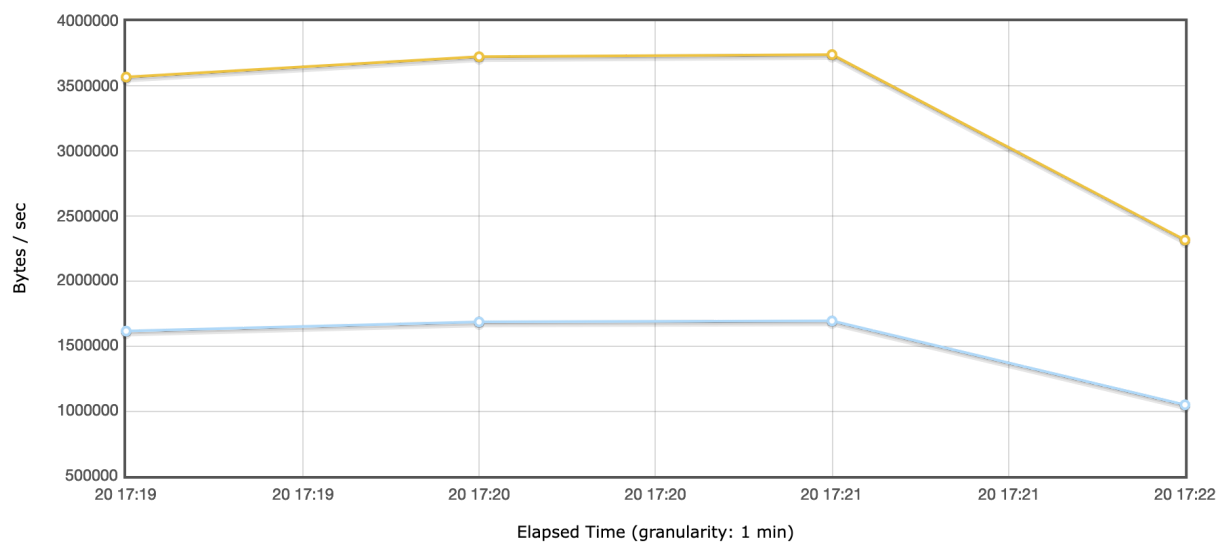


Figura 10: *Throughput* sin el proxy.

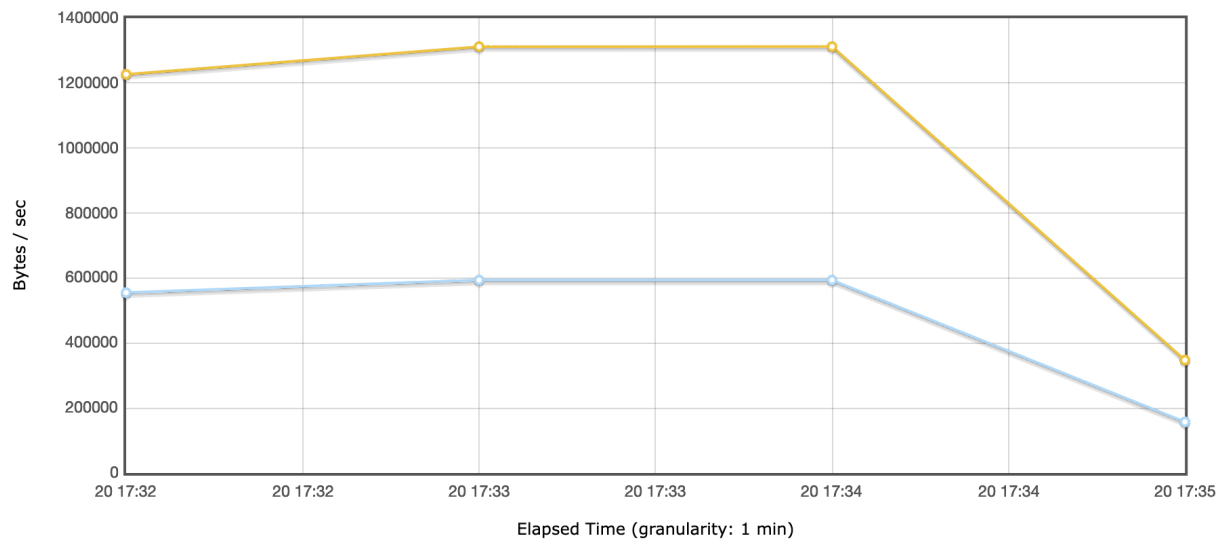


Figura 11: *Throughput* con proxy y buffer de 8KB.

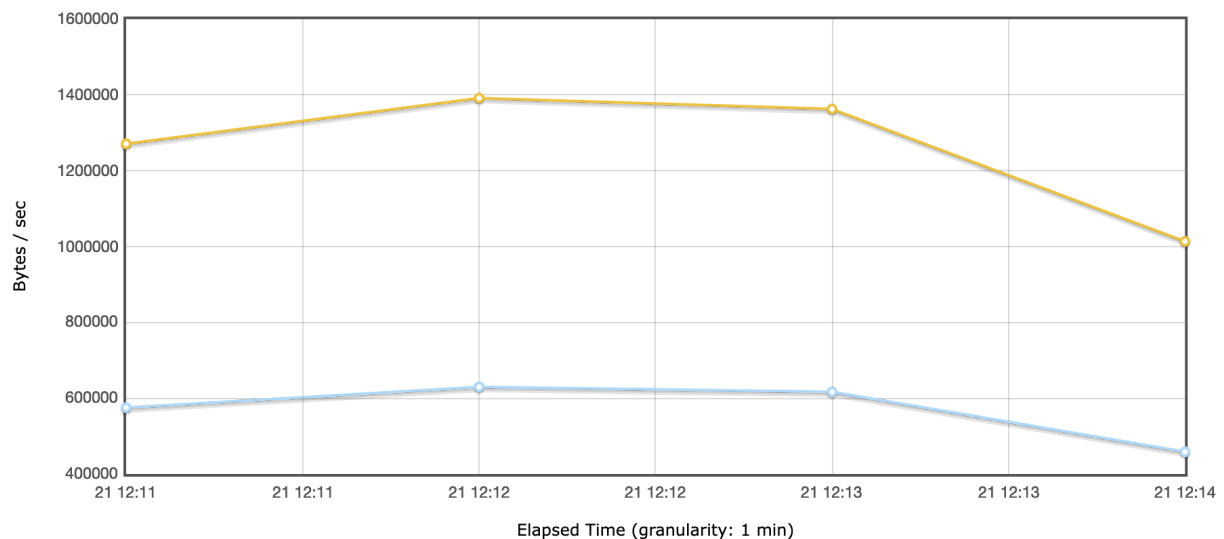


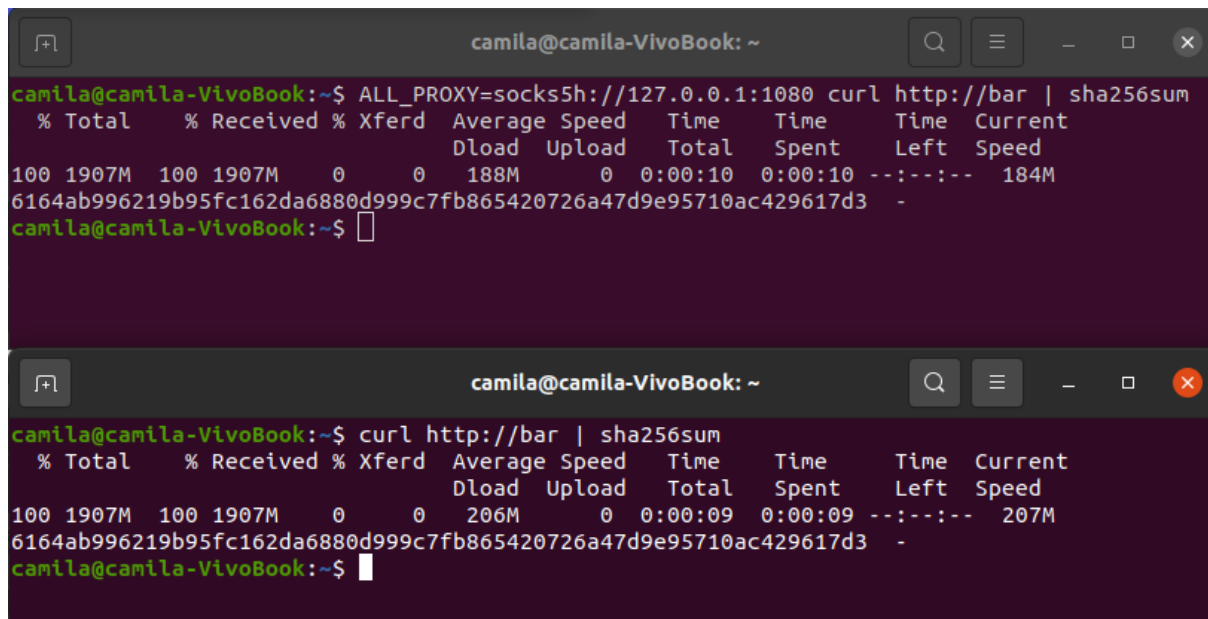
Figura 12: *Throughput* con proxy y buffer de 32KB.

Conclusiones

Con todos estos gráficos, se observa que la performance del test de carga no se ve afectada si se pasa de un buffer de 8KB a uno de 32KB, incluso mejora marginalmente. Esto nos muestra que no hay regresión de performance por el uso extra de memoria, **por lo que se eligió 32KB como tamaño de buffer.**

Verificación de la integridad de los datos

Para realizar esta prueba se configuró un servidor local *Nginx* con un archivo de casi 2MBs con datos aleatorios, servido en el host *bar*, el cual puede ser accedido mediante *http://bar*. El nombre se resuelve a *127.0.0.1* según lo configurado en */etc/hosts*.



```
camila@camila-VivoBook: ~  
camila@camila-VivoBook:~$ ALL_PROXY=socks5h://127.0.0.1:1080 curl http://bar | sha256sum  
% Total % Received % Xferd Average Speed Time Time Time Current  
Dload Upload Total Spent Left Speed  
100 1907M 100 1907M 0 0 188M 0 0:00:10 0:00:10 --:--:-- 184M  
6164ab996219b95fc162da6880d999c7fb865420726a47d9e95710ac429617d3 -  
camila@camila-VivoBook:~$  
  
camila@camila-VivoBook:~$ curl http://bar | sha256sum  
% Total % Received % Xferd Average Speed Time Time Time Current  
Dload Upload Total Spent Left Speed  
100 1907M 100 1907M 0 0 206M 0 0:00:09 0:00:09 --:--:-- 207M  
6164ab996219b95fc162da6880d999c7fb865420726a47d9e95710ac429617d3 -  
camila@camila-VivoBook:~$
```

Figura 13: Prueba con *Nginx* local.

Se puede ver que el hash de ambos outputs es el mismo, por lo que podemos concluir que la integridad de los datos es correcta.

Uso como proxy de Chrome

Utilizamos el siguiente comando para correr el navegador *Google Chrome* configurado con un proxy:

```
camila@camila-VivoBook:~$ google-chrome --proxy-server=socks5://localhost:1080 --user-data-dir=/tmp/x
```

Figura 14: Comando para utilizar el proxy socks como proxy de Chrome.

Se visitaron varias páginas web al mismo tiempo durante un periodo de tiempo y no se encontraron problemas.

Conexión a un origin server que no presta servicio (IPv4 e IPv6)

Si un cliente intenta conectarse a un servidor, a un puerto en el que no presta servicio, pueden ocurrir dos cosas:

- El servidor rechaza el pedido de conexión TCP, en cuyo caso el cliente puede de inmediato recibir el error de conexión.
- No se recibe ninguna respuesta, por ejemplo porque el servidor ignora el pedido de conexión TCP, en cuyo caso el cliente tendrá que esperar a que ocurra un *timeout* antes de recibir un error de conexión.

Podemos ver que el primer caso funciona correctamente. Para ambos IPv4 e IPv6 el comando finaliza de forma casi inmediata con el mensaje de error apropiado:

```
thomas@ThomasMain: ~
thomas@ThomasMain:~$ sudo netstat -nlt | grep 5050
thomas@ThomasMain:~$ # ^ No hay sockets abiertos escuchando en :5050
thomas@ThomasMain:~$
thomas@ThomasMain:~$ ncat --proxy 127.0.0.1:1080 --proxy-type socks5 127.0.0.1 5050
Ncat: Error: Connection refused.
thomas@ThomasMain:~$
thomas@ThomasMain:~$ ncat --proxy 127.0.0.1:1080 --proxy-type socks5 ::1 5050
Ncat: Error: Connection refused.
thomas@ThomasMain:~$
```

Figura 15: Rechazo de conexión por parte del servidor

Y para el caso que la conexión sea ignorada, probaremos conectarnos a 1.2.3.4:4321:

```
thomas@ThomasMain: ~
thomas@ThomasMain:~$ ncat --proxy 127.0.0.1:1080 --proxy-type socks5 1.2.3.4 4321
Ncat: Error: TTL expired.
thomas@ThomasMain:~$
thomas@ThomasMain:~$ ncat --proxy 127.0.0.1:1080 --proxy-type socks5 ::ffff:1.2.3.4 4321
Ncat: Error: TTL expired.
thomas@ThomasMain:~$
```

Figura 16: No se reciben respuestas por parte del servidor.

Podemos ver que nuestro servidor proxy reporta a través del protocolo socks5 el error “TTL Expired”, que es el error que devuelve cuando un intento de conexión resulta en un *timeout*.

Verificación del funcionamiento del *password dissector*

Para verificar el correcto funcionamiento del *password dissector*, podemos utilizar *ncat* para simular una conexión a un servidor POP3.

Con el servidor proxy abierto, iniciamos un *ncat* que simulará el servidor POP3:

```
thomas@ThomasMain:/socks5-server$ ncat -l 110 -C
```

Y en otra terminal iniciamos un cliente, que se conecte a este a través del proxy:

```
thomas@ThomasMain:/socks5-server$ ncat --proxy 127.0.0.1:1080
--proxy-type socks5 --proxy-auth pedro:pedro 127.0.0.1 110 -C
```

Hablando a través de estos dos *ncat*, podemos simular una comunicación POP3, y vemos que el proxy correctamente intercepta las credenciales y estas son impresas en los logs:


```
thomas@ThomasMain: /mnt/c/Users/Thomas/Desktop/socks5-server
thomas@ThomasMain: /mnt/c/Users/Thomas/Desktop/socks5-server$ ./bin/socks5v
2022-11-22T14:34:30 Listening for socks5 connections on TCP address :::1080
2022-11-22T14:34:30 Listening for management connections on TCP address 127.0.0.1:8080
2022-11-22T14:36:06 pedro A ::ffff:127.0.0.1 53522 127.0.0.1 110 0
2022-11-22T14:36:17 pedro P POP3 127.0.0.1 110 superusuario manzana33

thomas@ThomasMain: ~
thomas@ThomasMain:~$ ncat --proxy 127.0.0.1:1080 --proxy-type socks5 --proxy-auth pedro:pedro 127.0.0.1 110 -C
+OK Welcome to my POP3 server!
user superusuario
pass manzana33
+OK Correct user!
+OK Corrent password! Welcome :)

thomas@ThomasMain: ~
thomas@ThomasMain:~$ sudo ncat -l 110 -C
+OK Welcome to my POP3 server!
user superusuario
pass manzana33
+OK Correct user!
+OK Corrent password! Welcome :)
```

Figura 17: Las credenciales de una conexión POP3 son interceptadas.

Guia de instalación

Para la compilación y ejecución del programa se requiere tener instalado **GCC** y **Make** en sus versiones más recientes. Los binarios se pueden compilar corriendo:

```
ichayer@ichayer:/socks5-server$ make all
```

Se generarán dos binarios llamados *sock5v* y *client* dentro del directorio *bin* ubicado en la raíz. El primero corresponde al servidor proxy SOCKSv5, mientras que el segundo es un cliente que permite la administración de dicho servidor a través de un protocolo de monitoreo propietario.

Servidor proxy SOCKS5

El servidor del proxy SOCKS5 se puede correr con:

```
ichayer@ichayer:/socks5-server$ ./bin/socks5v [ARGS]
```

Se puede obtener el detalle de los flags y argumentos posibles corriendo:

```
ichayer@ichayer:/socks5-server$ ./bin/socks5v -h
```

Cliente de monitoreo

Para ejecutar el cliente, se debe correr el comando:

```
ichayer@ichayer:/socks5-server$ ./bin/client <command> [ARGS]
```

Se pueden consultar los posibles comandos y sus argumentos corriendo:

```
ichayer@ichayer:/socks5-server$ ./bin/client -h
```

y teniendo la variable de entorno previamente configurada como se explicará a continuación.

Instrucciones para la configuración

Servidor proxy SOCKS5

El listado de usuarios se debe ubicar en el directorio donde se corre el servidor, y debe tener el nombre *users.txt*. Dentro del mismo, en cada línea se detalla el carácter '@' (privilegio de administrador) o '#' (privilegio de usuario) seguido del nombre de usuario y la contraseña separados por dos puntos (:). Un ejemplo de un archivo *users.txt* válido es el siguiente:

```
@admin:admin  
#cliente:cliente
```

Para utilizar el proxy socks5, es requisito tener una cuenta registrada en el servidor. Para agregar un usuario, se debe contar con el rol de administrador y utilizar el comando comando [ADD-USER](#) que ofrece el protocolo de monitoreo.

Cliente de monitoreo

El cliente extrae las credenciales de una variable de entorno de nombre *TOKEN*. La misma debe contener el formato *<user>:<password>*. Si quisiéramos autenticarnos con el usuario "admin" que tiene la contraseña "admin" podríamos correr el siguiente comando previo a la ejecución del cliente:

```
ichayer@ichayer:/socks5-server$ export TOKEN="admin:admin"
```

Protocolo de monitoreo

En esta sección describimos el protocolo de monitoreo que se utiliza para consultar y modificar el estado del servidor. El mismo está basado en POP3 y SOCKS5.

Cuando un cliente desea hacer uso de este servicio, establece una conexión TCP con el servidor que previamente inició dicho sistema de monitoreo en un puerto puntual (8080 por default). La primera interacción entre el cliente y el servidor es una negociación sobre la versión del protocolo junto con un usuario y contraseña. En nuestra implementación, para que el cliente pueda acceder a la configuración de monitoreo, es requisito no sólo que se encuentre registrado en el servidor, sino que también debe tener privilegios de administrador. Una vez que el cliente se autentica, ambos intercambian *requests* y *responses* para **un comando particular**, es decir, que la conexión **no es persistente**.

Nota: no incluimos un “hello message” (primer mensaje de negociación) por parte del servidor porque el método de autenticación es obligatoriamente a través de usuario y contraseña a diferencia de SOCKS5.

A lo largo de la explicación, haremos uso de los siguientes tipos de dato:

- **String:** una secuencia de $n+1$ bytes donde el primer byte representa la longitud de la palabra (n) y los n bytes siguientes representan los caracteres de la palabra. Un string no puede estar vacío para este protocolo. Como la longitud se almacena en un byte y los strings no pueden estar vacíos se cumple: $0 < n < 256$.
- **NameList:** una cadena de caracteres que representa una lista de nombres, donde cada nombre se separa con un ‘\n’.
- **Byte:** representa un único byte. Si en la tabla se indica un valor entre comillas simples, entonces el valor dentro de las comillas denota el valor del byte (por ejemplo: el valor ‘\n’ representa el fin de línea, es decir el 10 en decimal).
- **Bytes:** representa una secuencia de bytes. Si en la tabla se indica un valor entre comillas dobles, entonces el valor dentro de las comillas en el orden escrito denota el contenido de la secuencia, donde cada caracter representa el valor de un byte (por ejemplo: “ab” denota la secuencia de bytes que contiene el byte 97 seguido del byte 98).

Estados de una sesión

La sesión de este servicio atraviesa dos estados:

1. **Estado de autorización:** en este estado, el cliente debe identificarse ante el servidor.
2. Una vez que el cliente ha hecho esto con éxito, la sesión entra en el **estado de transacción**. En este estado, el cliente solicita acciones por parte del servidor para obtener distintas métricas del servidor.

Estado de autorización

Como se mencionó, el cliente debe identificarse y autenticarse en el servidor, y para hacerlo, debe tener privilegios de administrador. Siguiendo lo indicado en el RFC 1929, el cliente enviará un **Username/Password request** con el siguiente formato:

BYTE	STRING	STRING
VER	UNAME	PASSW

- **VER** contiene la versión del protocolo.
- **UNAME** contiene el nombre de usuario.
- **PASSWD** contiene la contraseña asociada con el **UNAME** dado.

El servidor responde lo siguiente:

BYTE	BYTE
VER	STATUS

Donde:

- **VER** es un byte que contiene la versión del protocolo que maneja el servidor (en este caso es el byte 1).
- **STATUS** es un byte que puede tomar el valor 0 (autenticación válida) o distinto de 0 si hay un error.

Estado de transacción

Una vez autenticado, el cliente puede acceder a la información del servidor.

Sobre los *status codes* del servidor para este estado

Las respuestas a ciertos comandos consisten de un indicador de estado. Hay dos estados por parte del servidor y se representan con '+' y '-'. Seguido de esto, puede acompañar una descripción de qué sucedió.

Nota: La comunicación del cliente administrador hacia el servidor es un protocolo binario, sin embargo la vuelta es un protocolo de texto.

El servidor evalúa la solicitud y devuelve una respuesta formada de la siguiente manera para **cada comando**:

BYTE	BYTES	BYTES
STATUS	STREAM	END

- **STATUS** contiene un + en caso de respuesta positiva y un - en caso de respuesta negativa. Lo llamaremos *success* si es un '+' y *error* si es un '-'.
- **STREAM** contiene la data asociada a la ejecución del comando y su estado.
- **END** contiene en 2 bytes, "/r/n".

A partir del status, el programa cliente puede determinar si hubo un error o no al ejecutar un comando.

USERS

Lista los usuarios del sistema.

```
ichayer@ichayer:/socks5-server$ ./bin/client - - USERS
```

Por detrás, el cliente envía lo siguiente al servidor:

BYTE
CMD

- **CMD** contiene el byte 0 que representa el comando USERS.

El comando siempre retorna una respuesta satisfactoria (status *success*), y el servidor devuelve al cliente en la sección de **STREAM**:

STREAM		
BYTES	BYTE	NAMELIST
"OK listing users:"	'\n'	users

- Donde **users** consiste en una lista de los nombres de los usuarios registrados.

La respuesta del servidor es multilínea. Se define un máximo de 100 usuarios con nombres de usuario de como mucho 31 bytes de largo de forma que la respuesta quepa en el buffer de respuesta.

ADD USER

Agregar un usuario al sistema.

```
ichayer@ichayer:/socks5-server$ ./bin/client - - ADD-USER <username>
<password> <role>
```

Por detrás, el cliente envía lo siguiente al servidor:

BYTE	STRING		STRING		BYTE
CMD	ULEN	UNAME	PLEN	PASSW	ROLE

- **CMD** contiene el byte 1 que representa el comando ADD-USER.
- **ULEN** contiene la longitud del campo **UNAME**.
- **UNAME** contiene el nombre de usuario.
- **PLEN** contiene la longitud del campo **PASSWD**.
- **PASSWD** contiene la contraseña asociada con el **UNAME** dado.
- **ROLE** contiene el rol que se le asigna al usuario, que puede ser 1 (admin) o 0 (usuario sin privilegios).

Si el usuario se agrega correctamente (status *success*), el servidor devuelve al cliente en la sección de **STREAM**:

STREAM
BYTES
"OK user successfully added"

Si el usuario no se agrega correctamente (status *error*), el servidor devuelve al cliente en la sección de **STREAM**:

STREAM	
BYTES	BYTES
"ERR "	MSG

- Donde **MSG** consiste en un mensaje que detalla el error. Este podría ser:
 - "user already exists" si el usuario ya existe
 - "credentials too long" si el usuario o contraseña tienen demasiados caracteres.
 - "users limit reached" si la cantidad de usuarios supera el máximo admitido.
 - "no memory" si se acaba la memoria del lado del servidor.
 - "role doesn't exist" si el valor del byte enviado no corresponde con un rol válido.
 - "can't add user, try again" si ocurre un error desconocido.

DELETE USER

Borrar un usuario al sistema. Para borrarlo solamente se debe indicar el nombre del usuario.

```
ichayer@ichayer:/socks5-server$ ./bin/client - - DELETE-USER <username>
```

Por detrás, el cliente envía lo siguiente al servidor:

BYTE	STRING	
CMD	ULEN	UNAME

- **CMD** contiene el byte 2 que representa el comando DELETE-USER.
- **ULEN** contiene la longitud del campo **UNAME**.
- **UNAME** contiene el nombre de usuario

Si el usuario se agrega correctamente (status *success*), el servidor devuelve al cliente en la sección de **STREAM**:

STREAM
BYTES
"OK user successfully deleted"

Si el usuario no se agrega correctamente porque ya existe (status *error*), el servidor devuelve al cliente en la sección de **STREAM**:

STREAM	
BYTES	BYTES
"ERR "	MSG

- Donde **MSG** consiste en un mensaje que detalla el error. Este podría ser:
 - "user doesn't exist" si el usuario no existe.
 - "cannot delete user because no other admins exist" si el usuario es el único administrador registrado.
 - "can't delete user, try again" si ocurre un error desconocido.

CHANGE PASSWORD

Actualiza la contraseña de un usuario.

```
ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGE-PASSWORD
<username> <password>
```

Por detrás, el cliente envía lo siguiente al servidor:

BYTE	STRING		STRING	
CMD	ULEN	UNAME	PLEN	PASSW

- **CMD** contiene el byte 3 que representa el comando CHANGE-PASSWORD.
- **ULEN** contiene la longitud del campo **UNAME**.
- **UNAME** contiene el nombre de usuario.
- **PLEN** contiene la longitud del campo **PASSW**.
- **PASSW** contiene la nueva contraseña del usuario.

Si el servidor responde con un resultado satisfactorio (status *success*), entonces devuelve al cliente en la sección de **STREAM**:

STREAM
BYTES
"OK password successfully changed"

Si la contraseña no se cambia correctamente por algún motivo (status *error*), el servidor devuelve al cliente en la sección de **STREAM**:

STREAM	
BYTES	BYTES
"ERR "	MSG

- Donde **MSG** consiste en un mensaje que detalla el error. Este podría ser:
 - "user doesn't exist" si el usuario no existe.
 - "bad password" si la contraseña no cumple con los requisitos.
 - "credentials too long" si la contraseña es demasiado larga.
 - "no memory" si se acaba la memoria del lado del servidor.
 - "can't change password, try again" si ocurre un error desconocido.

CHANGE ROLE

Actualiza el rol de un usuario.


```
ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGE-ROLE <username>
<role>
```

Por detrás, el cliente envía lo siguiente al servidor:

BYTE	STRING		BYTE
CMD	ULEN	UNAME	ROLE

- **CMD** contiene el byte 4 que representa el comando CHANGE-ROLE.
- **ULEN** contiene la longitud del campo **UNAME**.
- **UNAME** contiene el nombre de usuario.
- **ROLE** contiene el rol que se le asigna al usuario, que puede ser 1 (admin) o 0 (usuario sin privilegios).

Si el servidor responde con un resultado satisfactorio (status *success*), entonces devuelve al cliente en la sección de **STREAM**:

STREAM
BYTES
"OK user successfully changed role"

Si el rol no se cambia correctamente por algún motivo (status *error*), el servidor devuelve al cliente en la sección de **STREAM**:

STREAM	
BYTES	BYTES
"ERR "	MSG

- Donde **MSG** consiste en un mensaje que detalla el error. Este podría ser:
 - "user doesn't exist" si el usuario no existe.
 - "role doesn't exist" si el byte enviado no corresponde con un rol válido.
 - "cannot change role because no other admins exist" si el usuario es el único administrador registrado y el cambio le quitaría los privilegios.
 - "can't change user role, try again" si ocurre un error desconocido.

GET DISSECTOR STATUS

Muestra el estado del disector de contraseñas.

```
ichayer@ichayer:/socks5-server$ ./bin/client - - GET-DISSECTOR-STATUS
```

Por detrás, el cliente envía lo siguiente al servidor:

BYTE
CMD

- **CMD** contiene el byte 5 que representa el comando DISSECTOR-STATUS.

El servidor siempre responde con un resultado satisfactorio (status *success*), y devuelve al cliente en la sección de **STREAM**:

STREAM	
BYTES	BYTES
"OK dissector status: "	status

- Donde **status** consiste en el estado actual del disector, que puede tomar el valor "on" u "off"

SET DISSECTOR STATUS

Habilita o inhabilita el disector de contraseñas.

```
ichayer@ichayer:/socks5-server$ ./bin/client - - SET-DISSECTOR-STATUS  
<status>
```

Si **<status>** tiene algún valor distinto de 'ON' u 'OFF', se arroja un error.

Por detrás, el cliente envía lo siguiente al servidor:

BYTE	BYTE
CMD	STATUS

- **CMD** contiene el byte 6 que representa el comando SET-DISSECTOR-STATUS.
- **STATUS** contiene el byte que representa el status (0 para off y distinto de 0 para on).

El status siempre se setea satisfactoriamente (status *success*), y el servidor devuelve al cliente en la sección de **STREAM**:

STREAM	
BYTES	BYTES
"OK dissector status: "	status

- Donde **status** consiste en el estado actual del disector, que puede tomar el valor "on" u "off"

GET AUTHENTICATION STATUS

Muestra el estado del disector de contraseñas.

```
ichayer@ichayer:/socks5-server$ ./bin/client - -  
GET-AUTHENTICATION-STATUS
```

Por detrás, el cliente envía lo siguiente al servidor:

BYTE
CMD

- **CMD** contiene el byte 7 que representa el comando DISSECTOR-STATUS.

El servidor siempre responde con un resultado satisfactorio (status *success*), y devuelve al cliente en la sección de **STREAM**:

STREAM	
BYTES	BYTES
"OK authentication method: "	status

- Donde **status** consiste en el estado actual del disector, que puede tomar el valor "no authentication" o "username/password required"

SET AUTHENTICATION STATUS

Habilita o inhabilita la autenticación por usuario/contraseña.

```
ichayer@ichayer:/socks5-server$ ./bin/client - -  
SET-AUTHENTICATION-STATUS <status>
```

Por detrás, el cliente envía lo siguiente al servidor:

BYTE	BYTE
CMD	STATUS

- **CMD** contiene el byte 8 que representa el comando SET-DISSECTOR-STATUS.
- **STATUS** contiene el byte que representa el status (0 para off y distinto de 0 para on).

Si el status se cambia satisfactoriamente (status *success*), el servidor devuelve al cliente en la sección de **STREAM**:

STREAM	
BYTES	BYTES
"OK authentication method: "	status

- Donde **status** consiste en el estado actual del dissector, que puede tomar el valor "no authentication" o "username/password required"

STATISTICS

Se cuentan cantidad de bytes transferidos, junto con la cantidad de conexiones históricas y la cantidad de conexiones concurrentes

```
ichayer@ichayer:/socks5-server$ ./bin/client - - STATISTICS
```

Por detrás, el cliente envía lo siguiente al servidor:

BYTE
CMD

- **CMD** contiene el byte 9 que representa el comando STATISTICS.

El status de respuesta es siempre satisfactorio (status *success*), y el servidor devuelve al cliente en la sección de **STREAM**:

STREAM							
BYTES	BYTES	BYTE	BYTES	BYTES	BYTE	BYTES	BYTES
"OK CONC: "	CONC	'\n'	"MCONC: "	MCONC	'\n'	"TBRECV:"	TBRECV

STREAM (cont.)					
BYTES	BYTES	BYTE	BYTES	BYTES	BYTE
TBSENT: "	TBSENT	'\n'	"TCON: "	CON	'\n'

- **CONC** consiste en una cadena de caracteres que contiene el número de conexiones concurrentes.
- **MCONC** consiste en una cadena de caracteres que contiene el número de conexiones concurrentes máxima que hubo históricamente.
- **TBRECV** consiste en una cadena de caracteres que contiene el número total de bytes recibidos.
- **TBSENT** consiste en una cadena de caracteres que contiene el número total de bytes transferidos.
- **TCON** consiste en una cadena de caracteres que contiene el número total de conexiones históricas.

Aclaración: Todos estos números están en formato decimal

Ejemplos de configuración y monitoreo

Uso del token

```
ichayer@ichayer:/socks5-server$ echo $TOKEN
kevin:correctpassword

ichayer@ichayer:/socks5-server$ ./bin/client - - GET-DISSECTOR-STATUS
+OK dissector status: on

ichayer@ichayer:/socks5-server$ export TOKEN=kevin:incorrectpassword

ichayer@ichayer:/socks5-server$ ./bin/client - - GET-DISSECTOR-STATUS
Username/password does not correspond to a registered admin user
Could not authenticate in server
```

```
ichayer@ichayer:/socks5-server$ echo $TOKEN
# Token not defined

ichayer@ichayer:/socks5-server$ ./bin/client - - <command> <args>
No token provided for connection
```

USERS

```
ichayer@ichayer:/socks5-server$ ./bin/client - - USERS
# TCP connection established.
+OK listing users:
camila
ivan
kevin
thomas

ichayer@ichayer:/socks5-server$ ./bin/client - - USER
# TCP connection not established. Command doesn't exist.
USER: Command doesn't exist

ichayer@ichayer:/socks5-server$ ./bin/client - - USERS a b c
# TCP connection established. Extra arguments ignored.
+OK listing users:
camila
ivan
kevin
thomas
```

ADD-USER

```
ichayer@ichayer:/socks5-server$ ./bin/client - - ADD-USER ichayer
password 0
# TCP connection established.
+OK user successfully added

ichayer@ichayer:/socks5-server$ ./bin/client - - ADDUSER ichayer
password 0
# TCP connection not established. Command doesn't exist.
ADDUSER: Command doesn't exist

ichayer@ichayer:/socks5-server$ ./bin/client - - ADD-USER ichayer
password abcd
# TCP connection not established. Invalid role format.
Invalid role format, must be a digit

ichayer@ichayer:/socks5-server$ ./bin/client - - ADD-USER ichayer
password 9
# TCP connection established.
-ERR role doesn't exist

ichayer@ichayer:/socks5-server$ ./bin/client - - ADD-USER password 0
# TCP connection not established. Missing username argument.
ADD-USER: Few arguments

ichayer@ichayer:/socks5-server$ ./bin/client - - ADD-USER thomas
password 0 a b c
# TCP connection established. Extra arguments ignored.
+OK user successfully added

ichayer@ichayer:/socks5-server$ ./bin/client - - ADD-USER ichayer
password 0
# TCP connection established.
-ERR user already exists

ichayer@ichayer:/socks5-server$ ./bin/client - - ADD-USER ichayer
veeeryyyy-looooong-paaasssswooord 0
# TCP connection established.
-ERR credentials too long.
```

Aclaración: En <role>, un 0 indica rol de “usuario” y un 1 indica “administrador”.

DELETE-USER

```
ichayer@ichayer:/socks5-server$ ./bin/client - - DELETE-USER thomas
# TCP connection established.
+OK user successfully deleted

ichayer@ichayer:/socks5-server$ ./bin/client - - DELETE-USER ichayer
# TCP connection established.
-ERR cannot delete user because no other admins exist

ichayer@ichayer:/socks5-server$ ./bin/client - - DELETE-USER kcatino
# TCP connection established.
-ERR user doesn't exist

ichayer@ichayer:/socks5-server$ ./bin/client - - DELETE-USER
# TCP connection not established. Missing username argument.
DELETE-USER: Few arguments

ichayer@ichayer:/socks5-server$ ./bin/client - - DELETEUSER ichayer
# TCP connection not established. Command doesn't exist.
DELETEUSER: Command doesn't exist

ichayer@ichayer:/socks5-server$ ./bin/client - - DELETE-USER thomas a b
c
# TCP connection established. Extra arguments ignored.
+OK user successfully deleted
```

CHANGE PASSWORD

```
ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGE-PASSWORD name
1234
# TCP connection established
+OK password successfully changed

ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGE-PASSWORD thomas
1234
# TCP connection established
+ERR user doesn't exist

ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGE-PASSWORD name
# TCP connection not established. Missing username argument.
CHANGE-PASSWORD: Few arguments

ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGEPASSWORD
# TCP connection not established. Command doesn't exist.
```



```
CHANGEPASSWORD: Command doesn't exist
```

CHANGE ROLE

```
ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGE-ROLE name 1
# TCP connection established
+OK user successfully changed role
```

```
ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGE-ROLE name
# TCP connection not established. Missing username argument.
CHANGE-ROLE: Few arguments
```

```
ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGE-ROLE name A
# TCP connection not established. Missing username argument.
Invalid role format, must be a digit
```

```
ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGE-ROLE name 9
# TCP connection established.
-ERR role doesn't exist
```

```
ichayer@ichayer:/socks5-server$ ./bin/client - - CHANGEROLE
# TCP connection not established. Command doesn't exist.
CHANGEROLE: Command doesn't exist
```

GET DISSECTOR STATUS

```
ichayer@ichayer:/socks5-server$ ./bin/client - - GET-DISSECTOR-STATUS
# TCP connection established
+OK dissector status: on
```

```
ichayer@ichayer:/socks5-server$ ./bin/client - - GET-DISSECTOR-STATUS
# TCP connection established
+OK dissector status: off
```

```
ichayer@ichayer:/socks5-server$ ./bin/client - - GET-DISSECTOR-STATUS a
b c
# TCP connection established. Extra arguments ignored.
+OK dissector status: off
```

```
ichayer@ichayer:/socks5-server$ ./bin/client - - GETDISSECTORSTATUS
# TCP connection not established. Command doesn't exist.
DISSECTORSTATUS: Command doesn't exist
```

SET DISSECTOR STATUS

```
ichayer@ichayer:/socks5-server$ ./bin/client - - SET-DISSECTOR-STATUS on
# TCP connection established
+OK dissector status: on

ichayer@ichayer:/socks5-server$ ./bin/client - - SET-DISSECTOR-STATUS ON
# TCP connection established
+OK dissector status: on

ichayer@ichayer:/socks5-server$ ./bin/client - - SET-DISSECTOR-STATUS
oFf
# TCP connection established
+OK dissector status: off

ichayer@ichayer:/socks5-server$ ./bin/client - - SET-DISSECTOR-STATUS
off a b c
# TCP connection established. Extra arguments ignored.
+OK dissector status: off

ichayer@ichayer:/socks5-server$ ./bin/client - - SET-DISSECTOR-STATUS
# TCP connection not established. Missing status.
SET-DISSECTOR-STATUS: Few arguments
```

GET AUTHENTICATION STATUS

```
ichayer@ichayer:/socks5-server$ ./bin/client - -
GET-AUTHENTICATION-STATUS
# TCP connection established
+OK authentication method: no authentication

ichayer@ichayer:/socks5-server$ ./bin/client - - GETAUTHENTICATIONSTATUS
# TCP connection not established. Command doesn't exist.
GETAUTHENTICATIONSTATUS: Command doesn't exist

ichayer@ichayer:/socks5-server$ ./bin/client - - GET-AUTHENTICATION-STATUS
a b c
# TCP connection established. Extra arguments ignored.
+OK authentication method: username/password required
```

SET AUTHENTICATION STATUS

```
ichayer@ichayer:/socks5-server$ ./bin/client - -  
SET-AUTHENTICATION-STATUS ON  
# TCP connection established  
+OK authentication method: username/password required
```

```
ichayer@ichayer:/socks5-server$ ./bin/client - -  
SET-AUTHENTICATION-STATUS OFF  
# TCP connection established  
+OK authentication method: No Authentication
```

```
ichayer@ichayer:/socks5-server$ ./bin/client - -  
SET-AUTHENTICATION-STATUS of  
# TCP connection not established. Invalid status.  
invalid status value of
```

```
ichayer@ichayer:/socks5-server$ ./bin/client - -  
SET-AUTHENTICATION-STATUS  
# TCP connection not established. Missing status.  
SET-AUTHENTICATION-STATUS: Few arguments
```

STATISTICS

```
ichayer@ichayer:/socks5-server$ ./bin/client - - STATISTICS  
+OK. Showing stats:  
CONC:4  
MCONC:45  
TBRECV:32647  
TBSSENT:25687  
TCON:230
```

Conclusiones

El trabajo resultó ser un desafío extraordinario que puso a prueba nuestro entendimiento de las implementaciones de los distintos tipos de protocolos de comunicación. A diferencia de la gran mayoría de los trabajos desempeñados a lo largo de nuestra carrera, valoramos enormemente la utilidad inmediata que este provee, y derivado de esto, la capacidad de poder realizar pruebas en el proyecto a través de casos de uso reales como puede ser navegar en internet, comunicar dos hosts entre sí por TCP, entre otros. También, durante el desarrollo del módulo de monitoreo, aprendimos la importancia de la buena definición de los protocolos ya que hacen que el desarrollo sea más dinámico. Fue una gran forma de decantar todo lo visto a lo largo del cuatrimestre.