Trabajo Práctico Especial

Segundo Cuatrimestre de 2022

Asignatura: 72.07 - Protocolos de Comunicación

Grupo 7: CONAN Protocol

Autores:

Banfi, Malena	**(61008)**
Fleischer, Lucas	**(61153)**
Perez Rivera, Mateo	**(61170)**
Szejer, Ian	**(61171)**

Índice

Introducción

El principal objetivo de este trabajo práctico es lograr la implementación de un servidor proxy para el protocolo SOCKSv5. Dicho servidor cumple con una serie de requisitos tanto funcionales como no funcionales, además de ser compatible con los RFCs correspondientes a cada una de estas funcionalidades. Consecuentemente, el mismo podrá ser usado tanto por curl como un navegador web o servidor POP3.

Estas implementaciones pudieron ser realizadas gracias a los conocimientos adquiridos en clase, tanto de programación en sockets como de comprensión de conocimientos acerca de protocolos. Cabe aclarar que decidimos trabajar a través de la extensión de Visual Studio Code llamada *LiveShare* para poder trabajar de forma colaborativa y tener una buena división de tareas.

Descripción de los protocolos y aplicaciones

Servidor proxy SOCKSv5

Implementamos un servidor proxy del tipo **SOCKSv5** con las características descritas en el RFC 1928. El cual cumple con las siguientes requisitos pedidos por la cátedra:

- Atender 500 clientes de forma simultánea
- Soporta autenticación via usuario/contraseña o uso de forma anónima
- Permite la resolución de cualquier tipo de nombre (FQDN) con la función getaddrinfo() sugerida por la cátedra, en un hilo aparte ya que esta operación es bloqueante.
- Si una de las respuestas del FQDN no da una conexión exitosa, se prueba con la siguiente,
- En tanto a los errores, si la conexión pedida no se pudo realizar, notificará al usuario por TTL exceeded, y de haber otros errores se respetaran los status codes determinados por el RFC 1928.

Protocolo CONAN para el monitoreo del servidor

Se diseñó e implementó un protocolo binario, que corre sobre TCP y no está orientado a conexión. Este se puede usar (de la forma que se mencionara más adelante) para pedir información del servidor y modificar configuración en él.

Para poder realizar esto, se debe ser un usuario administrador. Estos se identifican por un token de tamaño fijo de 16 bytes, los cuales deben ser obligatoriamente caracteres ASCII alfanuméricos.

Los request tienen la siguiente forma, siendo los números la cantidad de bytes de cada campo:

VERSION	TOKEN	METHOD	LENGTH	DATA
1	16	1	2	1-65535

Figura 1: Forma del request requerido por el protocolo

El detalle de cada campo es el siguiente:
Versión: Debe contener un byte que hace referencia a la versión del protocolo, por el momento solo se acepta la versión 1.
Token: Debe contener 16 caracteres ASCII en inglés para la autenticación como se mencionó previamente.
Método: Debe contener un byte que identifica la acción que se quiere realizar con el servidor, las opciones posibles son las siguiente, se aclara que con cliente se refiere a los usuario que usen el proxy, mientras que administradores son aquellos que utilizan el protocolo para interactuar con el servidor:

- Pedir cantidad de conexiones históricas.
- Pedir cantidad de conexiones actuales.
- Pedir la cantidad de bytes transferidos.
- Activar o desactivar el disector POP3.
- Agregar un cliente al proxy.
- Eliminar un cliente del proxy.
- Agregar un administrador al proxy.
- Eliminar un administrador del proxy.
- Obtener un listado de los clientes del proxy.

Aplicación de cliente de nuestro protocolo

Se desarrolló un cliente para poder darle uso práctico al protocolo CONAN, este tiene la característica de ser bloqueante y presentar una serie de opciones al usuario, que serán descritas posteriormente, para poder ejecutar sobre el servidor.

Se le agregó la característica de poder hacer varias consultas en una sola ejecución, pudiendo hacer hasta 10 consultas, las cuales realiza por separado, de forma iterativa y bloqueante entre ellas. Creando una conexión por el protocolo CONAN por cada una.

Además puede conectarse via IPv4 e IPv6 al servidor.

Problemas encontrados

Diseño

En cuanto a problemas de diseño, tuvimos problemas en cuanto a las implementaciones base del proyecto, más que nada con códigos provistos por la cátedra:

- En cuanto a los parches 8 y 9, no pudimos aplicarlos ya que algunos archivos generaban conflictos. Este problema fue solucionado editando los archivos problemáticos.
- Otro problema encontrado fue con los códigos base de la cátedra, ya que los mismos eran bastante complejos. Nos costó poder relacionarlos entre sí, como por ejemplo, el código de la máquina de estado (stm.c). De todos modos, luego de las clases de consulta pudimos entenderlos.
- También se nos dificulto pensar el protocolo, queriendo buscar algo simple pero que cumpla con lo requerido por la cátedra. Una vez que finalmente llegamos a una versión que nos parecía correcta, el diseño del parser fue una mayor dificultad.

Implementación

En cuanto a problemas de implementación, tuvimos problemas a la hora de tener que debuggear para poder encontrar errores, dado que no estábamos tan acostumbrados al uso de gdb, gracias a este trabajo pudimos mejorar nuestras habilidades con esta herramienta, lo cual será útil para próximos proyectos.

Iniciamos con una implementación propia para el manejo de los sockets, pero rápidamente nos dimos cuenta que nos convenía utilizar la implementación proporcionada de socks5nio.c el cual estaba incompleto. Completarlo fue una gran problemática y tardamos bastante tiempo, principalmente en desarrollar todos los estados.

Otra mayor dificultad encontrada, fue la creación de un thread aparte para evitar el bloqueo que significa la consulta de DNS, con el cual tuvimos varios errores que nos costó solucionar

Limitaciones de la aplicación

Protocolo

Hablando del protocolo, la primera limitación es que no está orientado a conexión, por lo que el cliente no puede hacer un ida y vuelta con el servidor, sino que está restringido a hacer consultas aisladas. Intentamos suplir esta limitación permitiendo que el cliente ejecute hasta 10 comandos en un solo mensaje, pero no compensa totalmente.

Proxy

La otra limitación que encontramos está en el proxy y viene de la mano con la forma de recibir las nuevas conexiones. Ya que como no creamos un hilo por cada conexión, y en vez de eso reservamos un file descriptor para el consumidor y otro para su consulta. Utilizando el pselect para escuchar hasta que haya necesidad y este tiene un límite de 1024 file descriptors suscritos, si a esto le sumamos que se usan 4 file descriptors usados para escuchar de forma pasiva en IPv4 y IPv6, nos deja con un límite de usuarios de 510 (509 si contamos los FD de STDOUT y STDERR que quedan abiertos). Cumpliendo con la cantidad de clientes pedidos por la cátedra pero altamente limitados en la escalabilidad del proyecto. Además al no preservar la información, cosa que estaba permitida por la cátedra, se pierde toda la información del uso del proxy

Posibles extensiones

Protocolo

Una posible extensión del protocolo sería hacerlo orientado a conexión, dejando muchas más claridad al cliente para trabajar con el servidor. También se podrían agregar más funcionalidades, como pausar el proxy, arrancarlo, limitar la cantidad de usuarios que tenga, agregar filtros y otras funcionalidades útiles.

Proxy

Una extensión posible para el Proxy podría intentar mitigar la limitación previamente mencionada, se podría adaptar el proxy para funcionar con hilos en vez de en un solo hilo(con excepción de los procesos bloqueantes) para mejorar su escalabilidad y posiblemente también su rendimiento

Otra posibilidad sería preservar la información en disco para que las estadísticas tengan mucho más sentido, quizá también agregar profundidad a estas. Como por ejemplo a la hora de contabilizar la cantidad de bytes transferidos, que se asigne con ellos la fecha y la hora, para poder estar informado de los momentos de más uso del proxy.

Conclusiones

Para concluir, gracias a la implementación de este trabajo práctico, pudimos terminar de plasmar los conocimientos adquiridos tanto en las clases prácticas como en las teóricas. Se desarrolló un servidor proxy que implementa el protocolo SOCKSv5 y gracias a que pudimos consolidar los conocimientos de los protocolos de comunicación vistos en clase pudimos generar uno propio, experimentando de punta a punta el proceso de diseño y el desarrollo de un proxy flexible, con variedad de aplicaciones y simplemente escalable. Así mismo tuvimos la posibilidad de diseñar un protocolo propio, experimentar con él y entender con más

profundidad el funcionamiento de estos. Reforzamos además, otros conceptos como el de máquina de estados, parsers y manejo de estructuras complejas, cosa por la que consideramos un trabajo práctico sumamente integrador.

Más allá de las dificultades y errores encontrados en el camino, creemos que gracias a nuestro trabajo práctico nos llevamos una gran experiencia y enseñanza del mismo.

Guía de instalación detallada y precisa.

Basta con simplemente ejecutar el comando: ***make all*** el cual generará los archivos *socks5d* y *client5* dentro de la raiz del proyecto. De todas formas, en el repositorio se cuenta con un archivo README.md que da más detalle sobre la instalación del mismo.

Instrucciones para la configuración.

Para configurar el proxy socks5, se debe correr en la carpeta del proyecto el ejecutable ./socks5d y se puede aplicar los siguientes flags para modificar su funcionamiento:

```
user@user:~/TPE-Protos$ ./socks5d -h
Usage: ./socks5d [OPTION]...
   -h               Imprime la ayuda y termina.
   -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.
   -v               Imprime información sobre la versión y termina.

   --doh-ip    <ip>
   --doh-port  <port>  XXX
   --doh-host  <host>  XXX
   --doh-path  <host>  XXX
   --doh-query <host>  XXX
```

Figura 2: Opciones para la configuración del proxy socks5

Para configurar el cliente usuario del protocolo CONAN, se debe correr en la carpeta del proyecto el ejecutable ./client5, especificando como mínimo una consulta a realizarle al servidor, las cuales se pueden ver a continuación:

```
user@user:~/TPE-Protos$ ./client5 -h
Usage: ./client5 [OPTIONS]... TOKEN [DESTINATION] [PORT]
Options:
-h                  imprime los comandos del programa y termina.
-r <user#pass>      agrega un usuario del proxy con el nombre y contraseña indicados.
-R <user#token>     agrega un usuario administrador con el nombre y token indicados.
-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.
-l                  imprime una lista con los usuarios del proxy.
-d <user>           borra el usuario del proxy con el nombre indicado.
-D <user>           borra el usuario administrador con el nombre indicado.
-O                  enciende el password disector en el server.
-F                  apaga el password disector en el server.
-v                  imprime la versión del programa y termina.
```

Figura 3: Opciones para la configuración del cliente del protocolo CONAN

Para profundizar sobre la configuración de los mismos, se pide leer el archivo README.md que se encuentra en el repositorio.

Ejemplos de configuración, monitoreo y performance

Ejemplos de configuración

Primero podemos ver una prueba del servidor corriendo sin ningún flag, y se comporta de la siguiente manera:

```
ianlucklol3@DESKTOP-1VOF8C2:/mnt/d/itba/Protos/TpProtos/TPE-Protos$ ./socks5d
Socks: listening on IPv4 TCP port 1080
Protocol: listening on IPv4 TCP port 8080
Socks: listening on IPv6 TCP port 1080
Protocol: listening on IPv6 TCP port 8080
```

Figura 4: Prueba del proxy corriendo sin ningún flag

Podemos ver que se escucha en los puertos default, 8080 en el caso del protocolo CONAN y en el puerto 1080 el socks5, como se ve también en netstat:

```
ianlucklol3@DESKTOP-1VOF8C2:~$ sudo netstat -nlp | grep socks
[sudo] password for ianlucklol3:
tcp        0      0 0.0.0.0:1080            0.0.0.0:*               LISTEN      220/./socks5d
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN      220/./socks5d
tcp6       0      0 :::1080                 :::*                    LISTEN      220/./socks5d
tcp6       0      0 ::1:8080                :::*                    LISTEN      220/./socks5d
```

Figura 5: Prueba sin flag netstat

Corriendo el servidor con el socket del proxy escuchando en la dirección ::1 y el puerto 7070 y el socket del protocolo en la direccion 127.0.0.1 y en el puerto 6070:

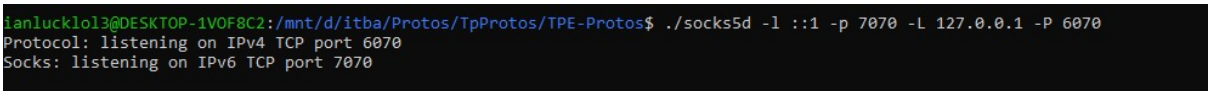

*Figura 6:*Prueba del proxy corriendo especificando puertos y direcciones

Viendo con netstat que efectivamente se está escuchando en esos puertos:

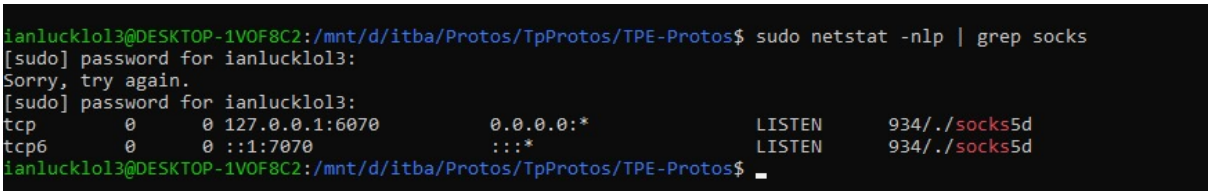

Figura 7: Prueba del proxy corriendo especificando puertos y direcciones netstat

Mismo setup, pero registrando los usuarios ian szejer y mateo perez:

Figura 8: Prueba del proxy modificando las direcciones, los puertos y registrando 2 usuarios

Ahora consultando como cliente por el protocolo, con el token roottokenspecial default por los usuarios registrados:

Figura 9: Demostración de la consulta de los usuarios registrados en el servidor

Pruebas de Performance

En este ejemplo se puede observar como el browser firefox corre correctamente varios streams de video en simultáneo.

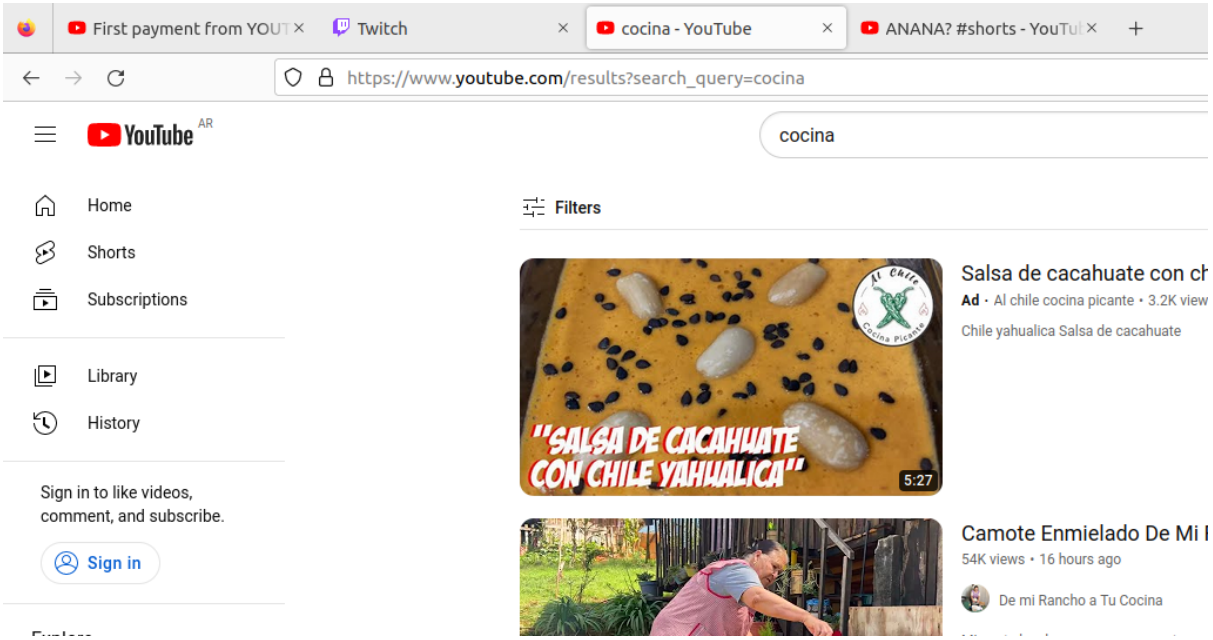

Figura 10: Firefox corriendo streams de video en simultáneo a través del proxy

```
male@male-HP:~/Documents/ITBA/2022/2Cuatri/Protos/TPE-Protos$ ./client5 -c -C -b roottokenspecial 127.0.0.1 8080
The amount of concurrent connections is: 66
The amount of historic connections is: 86
The amount of transferred bytes is: 55162296
male@male-HP:~/Documents/ITBA/2022/2Cuatri/Protos/TPE-Protos$
```

Figura 11: Cantidad de conexiones concurrentes, históricas y bytes transferidos del ejemplo de figura

En este otro ejemplo, se utilizó el server como proxy socks del browser firefox, seteando en el mismo la configuración de network adecuada dependiendo del caso de uso. Este ejemplo de testeo se realizó con "http://www.http2demo.io/" provista en clase.

Gracias a la figura 12 y 13 podemos observar como, tanto en http/1.1 y http/2, los tiempos de las pruebas hechas son muy similares, pero con un leve aumento al usar el proxy (figura 11). La velocidad con el proxy es levemente menor comparada a la velocidad sin el proxy.

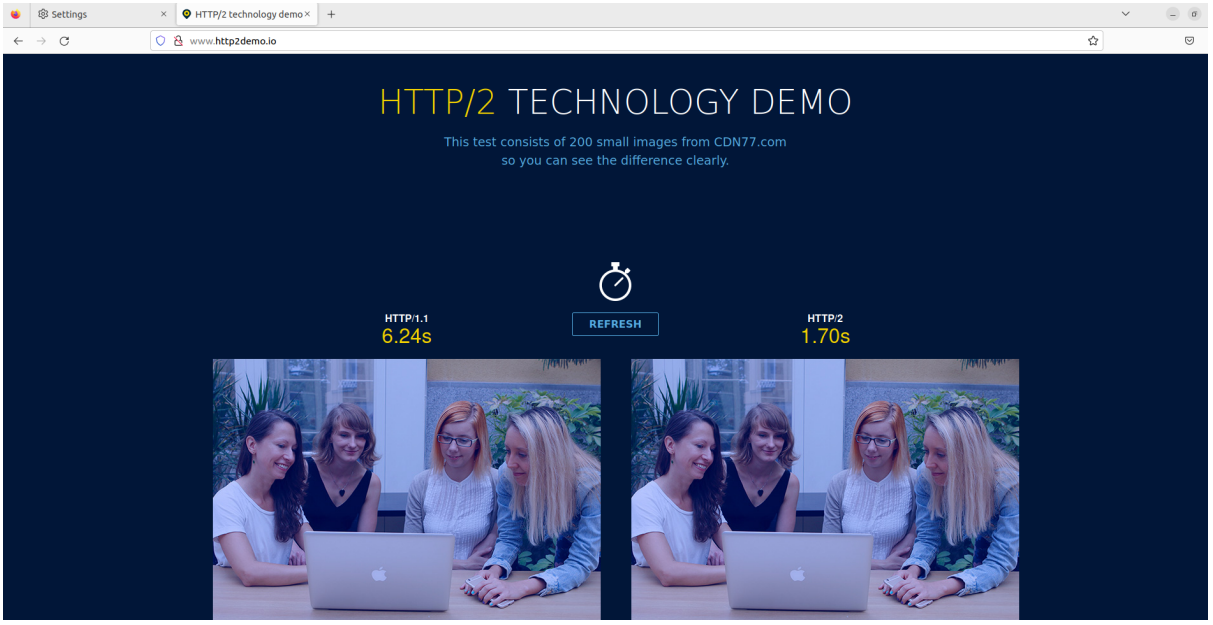

Figura 12: Testeo con el proxy en la página provista por la cátedra

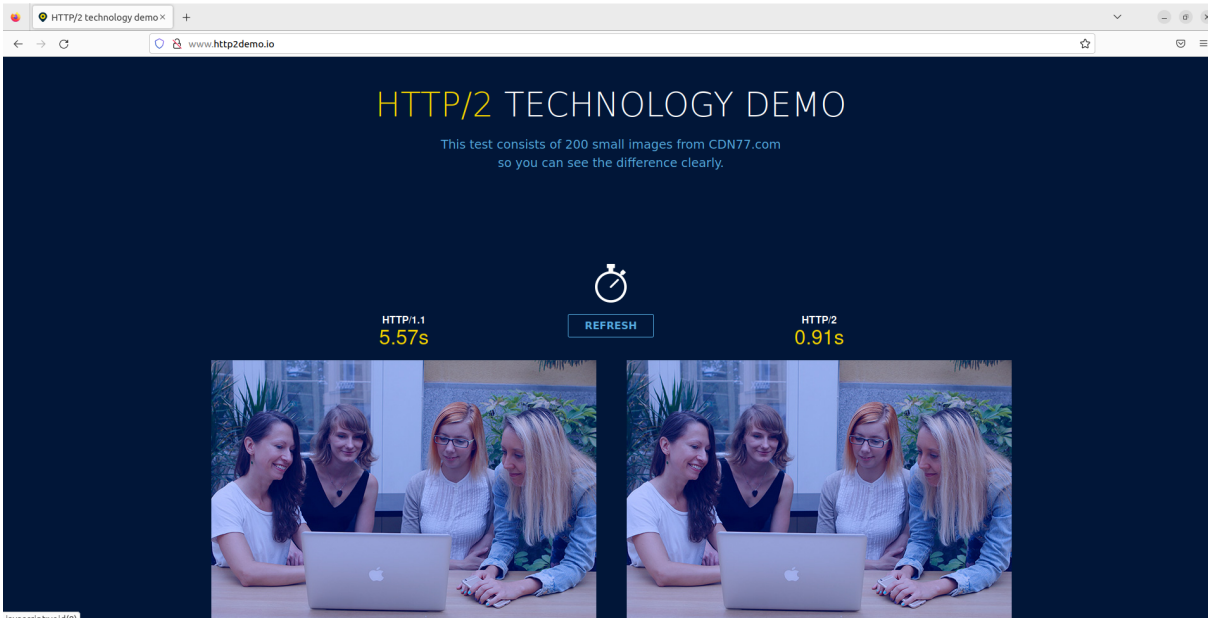

Figura 13: Testeo sin proxy en la página provista por la cátedra

A continuación, seguiremos mostrando resultados de los testeos pero ahora serán testeos ejecutando el comando time.

Sin Proxy

```
lucas@lucas-GF65-Thin-9SD:~/Downloads$ time cat ubuntu-22.04.1-desktop-amd64.iso | ncat localhost 1234

real    0m6,410s
user    0m1,503s
sys     0m3,769s
lucas@lucas-GF65-Thin-9SD:~/Downloads$ cat ubuntu-22.04.1-desktop-amd64.iso | sha256sum
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d  -
```

Figura 14: Testeo sin proxy de transferencia de contenido (ISO de ubuntu)

```
lucas@lucas-GF65-Thin-9SD:~/Downloads$ ncat -l localhost 1234 > transferSinProxy.iso
lucas@lucas-GF65-Thin-9SD:~/Downloads$ cat transferSinProxy.iso | sha256sum
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d  -
```

Figura 15: Inicialización del ncat y el verificacion del checksum (256-bits)

Con Proxy (Buffer 4096 bytes)

```
lucas@lucas-GF65-Thin-9SD:~/Downloads$ time cat ubuntu-22.04.1-desktop-amd64.iso | ncat --proxy-type socks5 --proxy 127.0.0.1:8081 127.0.0.1 1234

real    0m21,110s
user    0m1,641s
sys     0m4,640s
lucas@lucas-GF65-Thin-9SD:~/Downloads$ cat ubuntu-22.04.1-desktop-amd64.iso | sha256sum
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d  -
```

Figura 16: Testeo con proxy de transferencia de contenido (ISO de ubuntu)

```
lucas@lucas-GF65-Thin-9SD:~/Downloads$ ncat -l localhost 1234 > transferConProxy.iso
lucas@lucas-GF65-Thin-9SD:~/Downloads$ cat transferConProxy.iso | sha256sum
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d  -
```

Figura 17: Inicialización del ncat y el verificacion del checksum (256-bits)

Con Proxy (Buffer 2048 bytes)

```
lucas@lucas-GF65-Thin-9SD:~/Downloads$ time cat ubuntu-22.04.1-desktop-amd64.iso | ncat --proxy-type socks5 --proxy 127.0.0.1:8081 127.0.0.1 1234

real    0m20,445s
user    0m1,682s
sys     0m4,524s
lucas@lucas-GF65-Thin-9SD:~/Downloads$ cat ubuntu-22.04.1-desktop-amd64.iso | sha256sum
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d  -
```

Figura 18: Testeo con proxy de transferencia de contenido (ISO de ubuntu)

```
lucas@lucas-GF65-Thin-9SD:~/Downloads$ ncat -l localhost 1234 > transferConProxy.iso
lucas@lucas-GF65-Thin-9SD:~/Downloads$ cat transferConProxy.iso | sha256sum
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d  -
```

Figura 19: Inicialización del ncat y el verificacion del checksum (256-bits)

Con Proxy (Buffer 1024 bytes)

```
lucas@lucas-GF65-Thin-9SD:~/Downloads$ time cat ubuntu-22.04.1-desktop-amd64.iso | ncat --proxy-type socks5 --proxy 127.0.0.1:8081 127.0.0.1 1234

real    0m21,126s
user    0m1,743s
sys     0m4,106s
lucas@lucas-GF65-Thin-9SD:~/Downloads$ cat ubuntu-22.04.1-desktop-amd64.iso | sha256sum
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d  -
```

Figura 20: Testeo con proxy de transferencia de contenido (ISO de ubuntu)

```
lucas@lucas-GF65-Thin-9SD:~/Downloads$ ncat -l localhost 1234 > transferConProxy.iso
lucas@lucas-GF65-Thin-9SD:~/Downloads$ cat transferConProxy.iso | sha256sum
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d  -
```

Figura 21: Inicialización del ncat y el verificacion del checksum (256-bits)

Con Proxy (Buffer 512 bytes)

```
lucas@lucas-GF65-Thin-9SD:~/Downloads$ time cat ubuntu-22.04.1-desktop-amd64.iso | ncat --proxy-type socks5 --proxy 127.0.0.1:8081 127.0.0.1 1234

real    0m21,106s
user    0m1,662s
sys     0m3,881s
lucas@lucas-GF65-Thin-9SD:~/Downloads$ cat ubuntu-22.04.1-desktop-amd64.iso | sha256sum
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d  -
```

Figura 22: Testeo con proxy de transferencia de contenido (ISO de ubuntu)

```
lucas@lucas-GF65-Thin-9SD:~/Downloads$ ncat -l localhost 1234 > transferConProxy.iso
lucas@lucas-GF65-Thin-9SD:~/Downloads$ cat transferConProxy.iso | sha256sum
c396e956a9f52c418397867d1ea5c0cf1a99a49dcf648b086d2fb762330cc88d  -
```

Figura 23: Inicialización del ncat y el verificacion del checksum (256-bits)

Pruebas de Stress

Luego de correr el proxy con el programa JMeter (una herramienta de prueba de carga para analizar y medir el rendimiento de una variedad de servicios, con énfasis en aplicaciones web.) y utilizar el nombre de servidor foo.leak.com.ar, se obtuvieron los siguientes resultados (ver figuras 24 y 25)

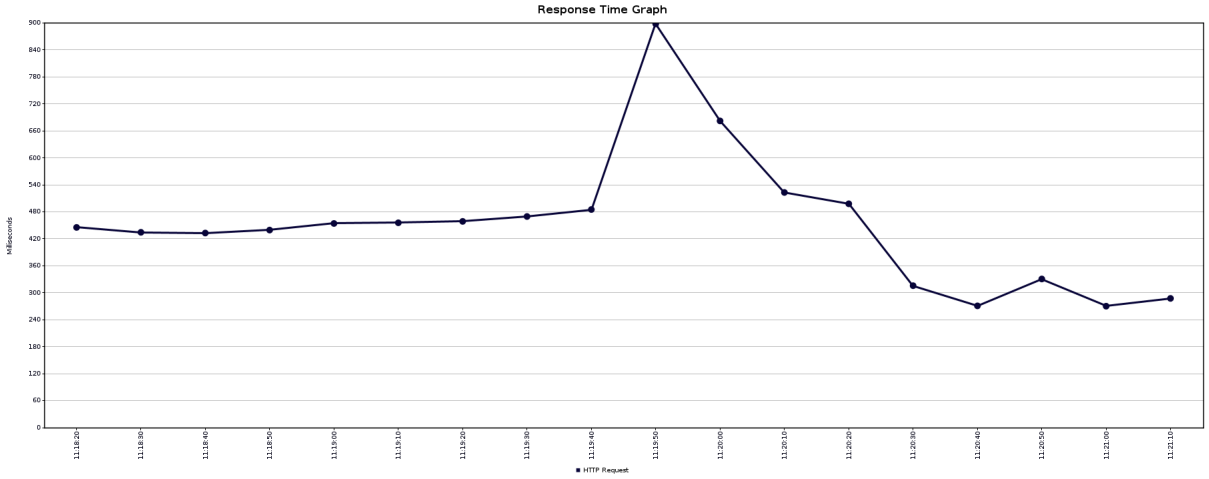

Figura 24: Response Time Graph from JMeter

Figura 25: Throughput from JMeter

Diseño del proyecto

El servidor socks5 dispone sockets pasivos escuchando por conexiones con IPv4 y/o IPv6, como se utiliza un único thread para correr todo el servidor, excepto en excepciones muy puntuales, a cada socket requerido se le asigna un fd y utilizar pselect() para monitorear sus accesos, en un inicio serán solo los pasivos del servidor.

Al querer conectar un cliente al proxy, se dividen las etapas en distintos estados, cada una con varias acciones posibles dependiendo de los resultados o lo pedido. Cuyo funcionamiento queda a cargo de la máquina de estados “stm” y los estados declarados en el archivo sock5nio.c . Para concretar la conexión primero se llevan a cabo los pasos determinados por el RFC 1928.

En tanto a métodos de autenticación, sólo se soporta el medio nombre/contraseña, los cuales son registrados previamente, al correr el servidor o mediante el uso del protocolo CONAN.

Una vez la conexión está completa, se resuelve el nombre del REQUEST si es necesario (utilizando getaddrinfo(), corriendo en un hilo aparte) y se intenta concretar la conexión pedida en el REQUEST.

Pasando así a la etapa de COPY, en el cual el proxy es un intermediario con y adicionalmente como fue pedido por la cátedra se diseñó un protocolo el cual está especificado en el RFC creado, presente en el repositorio.