Informe de Trabajo Práctico Especial

15/07/2025

Protocolos de Comunicación - Grupo 9

Integrantes:

- Agustin Ronda - 64507
- Tomás Borda - 64517
- Lautaro Paletta - 64499
- Nicolás Arancibia - 64481

Profesores:

- Garberoglio, Marcelo Fabio
- Kulesz, Sebastian
- Axt Roberto Oscar, Roberto Oscar
- Stupenengo Faus, Hugo Javier

Índice

Descripción detallada de los protocolos y aplicaciones desarrolladas

Servidor SOCKS5

Se desarrolló un servidor TCP/IP que cumple con lo definido en el *RFC 1928 - SOCKS Protocol Version 5*, siendo capaz de funcionar como un servidor proxy entre un cliente y un servidor final para redirigir el tráfico de red entre ellos como intermediario. Si bien se siguieron los lineamientos descritos en el ya mencionado RFC, es importante destacar que no se implementó GSSAPI como método de autenticación y que el único comando aceptado para el servidor es CONNECT, mientras que el RFC define además de éste BIND y UDP ASSOCIATE.

El servidor desarrollado utiliza operaciones no bloqueantes, permitiendo aceptar más de 500 conexiones simultáneas. En caso de requerir bloquearse, por ejemplo para resolver un nombre de dominio, deriva la operación a otro hilo de ejecución.

Servidor HotDogs y aplicación cliente

Con el fin de monitorear y configurar el servidor SOCKS5 en tiempo de ejecución, se desarrolló un protocolo binario sobre TCP al que se denominó *HotDogs Protocol* (de ahora en adelante *HDP*). La documentación para el mismo fue registrada en el archivo *HotDogsProtocol.txt* ubicado en la carpeta doc del proyecto realizado con el formato de RFC. Para la utilización de este protocolo, se desarrolló por un lado una aplicación cliente que permite utilizar toda la potencia de HDP, así como los sistemas pertinentes para atender el protocolo desde el servidor TCP/IP. De esta manera, el servidor cuenta con sistemas de registros de conexiones, usuarios y métricas que pueden ser consultadas por la aplicación cliente.

HDP cuenta con las siguientes prestaciones:

1. Autenticación con usuario y contraseña: siendo que las operaciones que se pueden realizar dentro de HDP son pensadas para usuarios con privilegios sobre el protocolo, para poder ingresar el usuario debe autenticarse con usuario y contraseña.
2. Dos modos de operación: RETR (para obtener información de métricas del servidor y de los usuarios) y MOD (para realizar configuraciones en los parámetros del servidor).

El modo de operación RETR es útil dado que nos permite acceder a datos sobre la ejecución del servidor en tiempo real, siendo estos:

1. Cantidad de conexiones históricas.
2. Cantidad de conexiones concurrentes actuales.
3. Cantidad de conexiones fallidas.
4. Cantidad de bytes transferidos entre el servidor y el servidor destino.
5. Lista de nombres de usuarios.
6. Registros de conexiones al servidor.

Por otro lado, las operaciones que el modo MOD nos permite realizar son las siguientes:

1. Configurar el tamaño de los buffers usados para el protocolo SOCKS5.
2. Agregar nuevos usuarios con su nombre de usuario y contraseña.
3. Eliminar un usuario dado su nombre de usuario.

En el documento anteriormente mencionado se encontrará un detalle de las operaciones disponibles en cada modo de operación y el formato de los mensajes intercambiados, así como ejemplos de uso.

Problemas encontrados durante el diseño y la implementación

Una de las dificultades principales fue la determinación de un tamaño de buffer adecuado. Para esto, se ejecutaron pruebas de descarga de un archivo de 100MB con el objetivo de hallar un tamaño de buffer que hiciera que el tiempo de descarga sin el proxy fuera lo más similar posible al que se obtiene sin utilizarlo. Para ver este primer tiempo, se corre la primer prueba con las métricas que se observan en la figura a continuación:

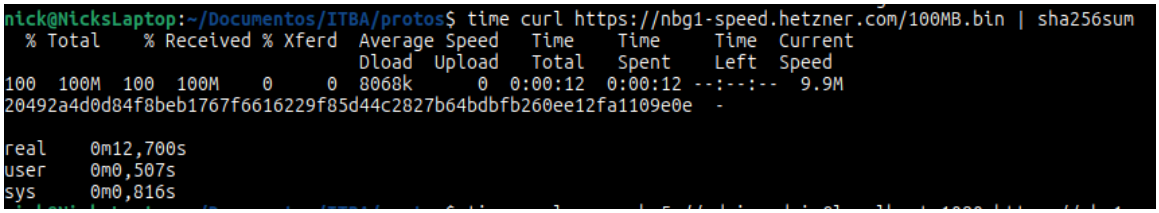

Figura 1: Prueba de descarga del archivo de 100MB sin utilizar el proxy SOCKS

Para el buffer, se consideraron tamaños de 4KB, 16KB y 64KB, ya que tamaños inferiores resultan demasiado chicos para las requests comunes, llevando consecuentemente a una peor performance del servidor, y considerando que se podrían tener al menos 1000 buffers (en el caso de 500 clientes concurrentes), buscando entonces un tamaño que no acelere el funcionamiento del servidor en un detrimento considerable del espacio en memoria ocupado. A continuación se detallan las pruebas para cada uno de los buffers mencionados:

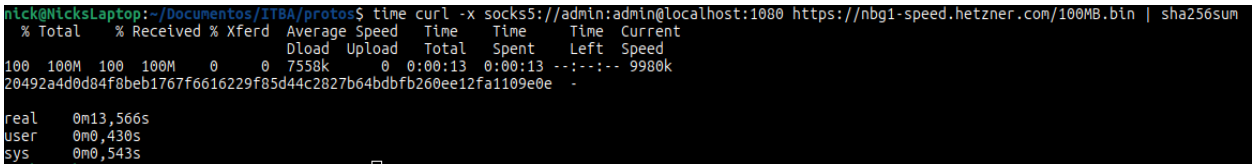

Figura 2: Prueba de descarga del archivo utilizando un tamaño de buffer de 4KB

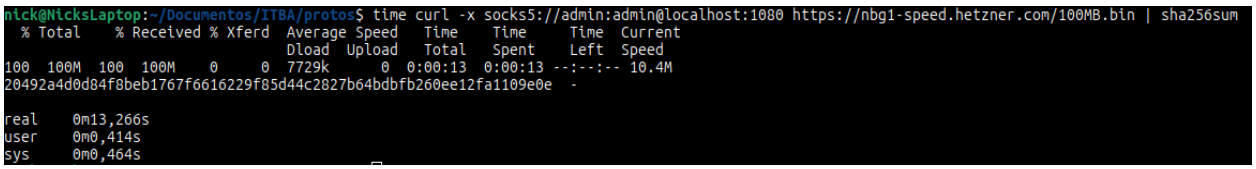

Figura 3: Prueba de descarga del archivo utilizando un tamaño de buffer de 16KB

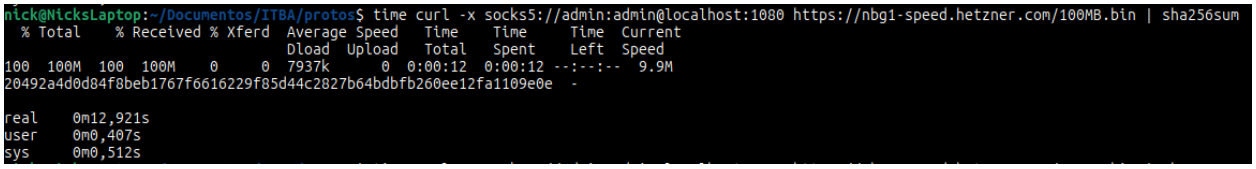

Figura 4: Prueba de descarga del archivo utilizando un tamaño de buffer de 64KB

Si bien el buffer de 64KB trae un tiempo de descarga similar al que se obtiene sin utilizar el proxy, se consideró que la diferencia no es lo suficientemente importante como para mantener 1000 buffers de 64KB cada uno, así como también se analizó que un buffer de 16KB trae resultados similares sin el espacio en memoria que el mayor tamaño implica. Por lo tanto, habiéndo hecho estas pruebas se concluyó en utilizar un tamaño de 16KB.

Otra decisión que implicó análisis fue el decidir cómo almacenar los registros de acceso; si bien éstos se imprimen en el servidor al momento de una conexión SOCKS, el protocolo HDP permite acceder a los mismos, en concordancia por lo solicitado en la consigna. Se manejaron dos alternativas: manejar registros almacenados en memoria o perdurarlos en un archivo en disco. Las desventajas de tener los registros todo el tiempo en memoria listos para ser consultados es justamente el espacio que estos ocupan y las implicaciones que esto tiene, ya que si se define un límite de registros entonces va a haber un punto donde se va a tener que limpiar todos los registros e iniciar de nuevo, perdiendo los anteriores en una misma ejecución del servidor, mientras que si no se limitan entonces ocuparía gran parte de memoria que el proxy SOCKS podría querer utilizar, evitando así que atienda clientes de forma efectiva. En vista de estas desventajas, se optó por mantener los registros en memoria, en un archivo fácilmente accesible que permite consultar cuando hace falta y evita el uso innecesario de memoria útil para el proxy.

Un último problema, pero no menor, fue al momento de diseñar el protocolo, más precisamente los formatos de respuesta de listas de usuarios y registros. Se manejaron, nuevamente, dos posibilidades: envíar un array, ya sea de usuarios o registros, teniendo que envíar a su vez otro array donde cada elemento indica la longitud del usuario o del registro en cuestión, o envíar una respuesta ya formateada con su longitud. Debido a la facilidad para la lectura y el formato predefinido de los registros, se optó por esta última opción, separando cada uno de los elementos por un carácter \r para facilitar su lectura, tokenizando la entrada en base a ese carácter y pudiendo imprimir fácilmente el contenido de la respuesta.

Limitaciones de la aplicación

La principal limitación del sistema es la cantidad de conexiones concurrentes que acepta. Esto es debido a que se utiliza la función *pselect(),* que en la mayoría de los casos permite como máximo monitorear 1024 *file descriptors*.Teniendo en cuenta que para los sockets pasivos de SOCKS5 y HotDogs necesitamos 4 *file descriptors,* otros 2 *file descriptors* para stdout y stderr (stdin se omite en la cuenta dado que el server lo cierra pero en casos de que el sistema operativo no lo permita este también debe sumarse a la cuenta), nos quedarán 1018 libres. Luego, por cada conexión de SOCKS5 necesitamos uno para el extremo con el cliente y otro para el extremo con el servidor destino, teniendo entonces 2 *file descriptors* por cada cliente de SOCKS5, por último el server se reserva 1 *file descriptors* para ser capaz de ir rechazando aquellas nuevas conexiones que ya no puede tomar. Como consecuencia, sin tener conexiones activas en HotDogs, tendremos como máximo 508 conexiones activas en el servidor SOCKS5. Sin embargo dicho cálculo es únicamente válido si la máquina corriendo la aplicación servidor no abre otros *file descriptors* para configuración interna de la misma y además es capaz de cerrar la entrada estándar al servidor.

Otra limitación del sistema consecuencia de la anterior que, dado el caso de que tengamos los 1024 *file descriptors* ocupados con conexiones de SOCKS5 y la entrada estándar al server no se pudo cerrar al inicializarse, un usuario de HotDogs quiera conectarse para monitorear el servicio, no podrá hacerlo.

Posibles extensiones

Como principal extensión al sistema, con el fin de que se cumpla al 100% la especificación del RFC 1928, se podría agregar el método de autenticación GSSAPI al protocolo SOCKS5 y los comandos BIND y UDP ASSOCIATE.

Además, se puede mejorar el rendimiento del servidor mediante una refactorización de la interfaz *selector* provista, dado que éste aspecto es transparente para el proxy SOCKS5 pero una implementación más eficiente puede ayudar a la escalabilidad y eficiencia del mismo. Una alternativa, por ejemplo, sería utilizar *epoll* en vez de *select*, eliminando la restricción de los *1024 file descriptors* máximo y quedando únicamente restringido al sistema operativo.

Así también, HDP fue desarrollado teniendo en mente posibles extensiones, utilizando como indicador el campo *version* durante la autenticación. Asimismo, se pueden agregar fácilmente códigos de estado, modos de operación u opciones para cada modo. Por ejemplo podría permitirse cambiar la contraseña de un usuario, limpiar el historial de registros o permitir extraer éstos utilizando algún filtro, por ejemplo especificando el día y mes de los registros que se desee obtener.

Conclusiones

En conclusión, se alcanzó la construcción de un servidor que permite su utilización como proxy SOCKS y que es posible utilizar para navegar por internet de forma transparente y configurado en tiempo de ejecución mediante el protocolo diseñado. Si bien es posible realizar ajustes para mejorar la performance del mismo, el resultado conseguido permite que un usuario pueda tener una experiencia de navegación por internet similar a la que tendría sin el proxy.

El desarrollo de este trabajo práctico fue desafiante pero útil para entender consideraciones en el diseño e implementación tanto de protocolos como servidores. Asimismo, permitió comprender la importancia de las operaciones no bloqueantes en este tipo de sistemas para su escalabilidad y para prevenir usos maliciosos del servidor en pos de lograr bloquearlo para otros usuarios.

Ejemplos de prueba

Para comprobar que el servidor sea capaz de mantener más de 500 conexiones concurrentes de acuerdo a la normativa del trabajo práctico, se hizo uso de un archivo extra en la carpeta de test, cuya implementación se encargará de lanzar n cantidad de hijos a conectarse en un dominio y puerto especificado por línea de comandos así como la cantidad n de procesos concurrentes. A continuación puede observarse la ejecución del test (Ver Figura 5) y los resultados del monitoreo de métricas mediante la aplicación cliente (Ver Figura 6).

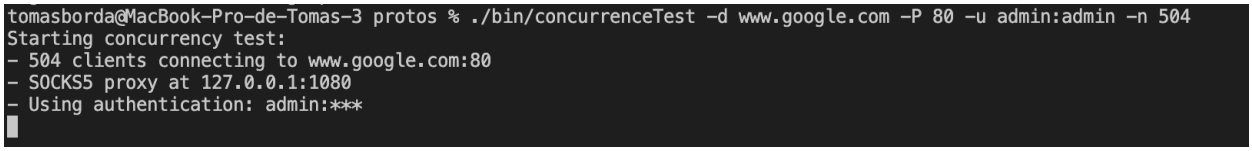

Figura 5: Ejecución del programa de testeo de la concurrencia

Figura 6: Ejecución del cliente para revisar las métricas

Por otro lado, se ejecutaron en paralelo 10 descargas para comprobar las mismas métricas que el caso anterior con el proxy funcionando de forma concurrente. Para ello, se ejecutó el comando que se ve a continuación

```
seq 1 10 | xargs -P10 -I{} bash -c '
  /usr/bin/time -f "real: %e s\nuser: %U s\nsys: %S s" \
    curl -s -o /dev/null -x socks5h://admin:admin@localhost:1080 \
    -w "Velocidad: %{speed_download} bytes/s\n\n" \
    https://nbg1-speed.hetzner.com/100MB.bin
'
```

Figura 4: Se puede observar el comando a ejecutar para obtener las métricas de 10 descargas de 100MB concurrentes utilizando el proxy SOCKS.

Finalizada la ejecución, se pudo extraer una velocidad de descarga promedio de 2.92MB/s, lo cual es estable para el tamaño del archivo utilizado y para haber mantenido 10 descargas

concurrentes y un tiempo de descarga promedio de aproximadamente 38 segundos. Los detalles de las pruebas realizadas pueden ser observados a continuación:

Métrica	Promedio	Mínimo Observado	Máximo Observado
Tiempo (s)	38.79	25.06	48.99
Velocidad de descarga (MB)	2.92	2.14	4.18

Tabla 1: Se observan los resultados de la ejecución de 10 descargas en paralelo de un archivo de 100MB para cada una de las métricas previamente utilizadas.

Guía de instalación

La instalación del proyecto puede realizarse siguiendo los siguientes pasos:

1. Parado en la raíz del proyecto, correr el comando *make* o *make all* para compilar el servidor y la aplicación cliente.
2. Tras un rato, se podrá ver una carpeta *bin* que contiene el ejecutable del servidor (*socks5d*), de la aplicación cliente (*client*) y del test de concurrencia (*concurrenceTest)* . Podrán ser ejecutados con los flags correspondientes, pudiendo utilizar -*h* para obtener información acerca de todos los flags disponibles para ambos.

Instrucciones para la configuración

Para el servidor, nuevamente, es posible visualizar todos los flags de configuración disponibles con -h. Se recomienda que al inicializar el servidor se lo haga con el flag de -*u*, que sirve para indicar un administrador con usuario y contraseña, quien luego podrá modificar y/o acceder a la información del servidor SOCKS mediante HDP. Además, otras configuraciones que se pueden realizar son, por ejemplo, las direcciones y puertos donde se servirán los servicios provistos por el servidor.

Para el cliente, al igual que el servidor, pueden visualizarse las opciones de ejecución con -*h*. Sin embargo, la aplicación cliente siempre va a requerir que el usuario especifique el nombre de usuario y contraseña de un administrador del servidor con el flag -*u*, ya que, por diseño, la utilización del protocolo requiere de autenticación. Al finalizar las acciones solicitadas al ejecutar la aplicación con los flags, ésta finalizará.

Ejemplos de configuración

Para el ejemplo de configuración vamos a inicializar un servidor con un usuario admin cuyas credenciales van a ser admin:admin para nombre de usuario y contraseña respectivamente. Como no especificamos ningún puerto ni para SOCKS ni para HDP, se utilizarán los defaults, es decir, 1080 y 42069 respectivamente. A continuación, se observa la configuración inicial mencionada:

Figura 5: Inicialización del servidor con el usuario admin:admin

Ahora vamos a utilizar el cliente. Recordemos que para ello requerimos autenticarnos en el servidor, por lo que vamos a utilizar el usuario creado. Para ello utilizamos el flag *-u* con las credenciales correspondientes. Además, utilizaremos el flag de *-add* para añadir un usuario nuevo con las credenciales *hot:dogs* y luego vamos a listar todos los usuarios con *-lu*. Todo este proceso descrito puede observarse a continuación:

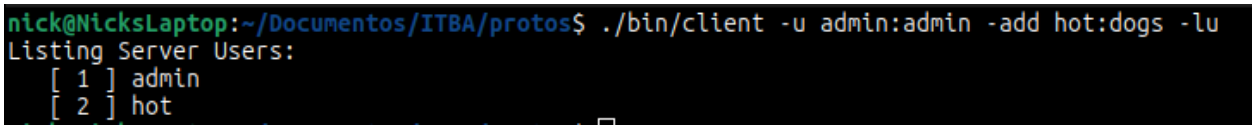

Figura 6: Utilización del cliente para añadir un nuevo usuario y listar los usuarios existentes.

Por último borramos el usuario admin inicial para quedarnos solo con el ya creado con el flag *-rm*, listamos los usuarios para verificar que efectivamente fue eliminado y vemos las métricas del servidor con el flag *-m*, tal como se muestra en la siguiente figura:

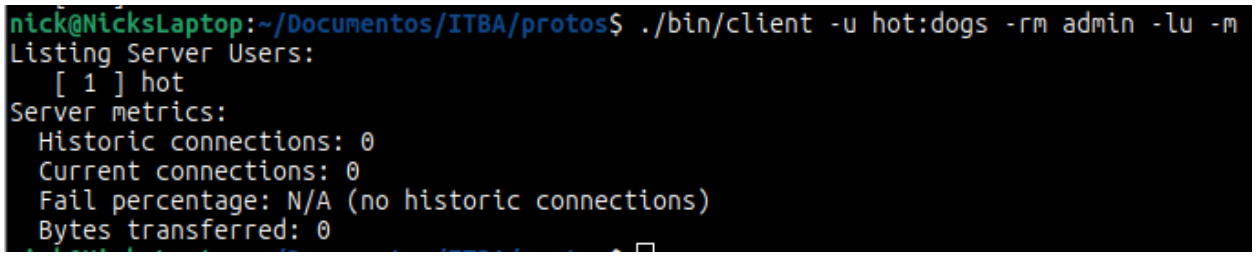

Figura 7: Utilización del cliente para eliminar el usuario admin, listar los usuarios existentes, verificando que efectivamente solo queda el usuario hot, y listar las métricas.

Documento de diseño del proyecto

Como se mencionó al inicio, el servidor se desarrolló bajo un esquema de conexión TCP con operaciones no bloqueantes para el manejo de clientes con el fin de tener múltiples conexiones sincrónicas. Ambos servidores, tanto el proxy como el de HotDogs, pueden recibir conexiones desde sus direcciones IPv4 o IPv6.

El servidor pasivo, al recibir una conexión, aceptará al cliente y lo registrará en el selector con los handlers correspondientes de acuerdo al puerto en el que se recibió esa conexión. En caso de que haya sido el puerto asociado al servidor proxy, el handler tendrá los triggers correspondientes a ser ejecutados cuando se reciba una lectura o escritura en ese puerto para interactuar con el protocolo SOCKS5, caso contrario, con el protocolo HotDogs en su puerto correspondiente.

Los handlers de cada servicio se apoyan en una máquina de estados que permite ir recorriendo el conjunto de estados posibles para una conexión dada una entrada recibida o dada una salida que se tenga que enviar dependiendo del protocolo y del estado en el que se encuentre la conexión. Estos estados con sus posibles transiciones fueron definidos por nosotros durante el desarrollo del servicio SOCKS5 para lograr congruencia con la especificación del mismo y del servicio HotDogs para cumplir con un protocolo binario con los requerimientos impuestos por la cátedra.

Es importante destacar que, para la redirección de datos entre un cliente del servidor SOCKS5 y un destino final (o viceversa), se utilizaron dos buffers: uno para almacenar los datos que se reciben del cliente y se tienen que enviar al destino final, y otro buffer para el camino inverso.

Para el desarrollo de los servidores, se utilizaron las herramientas de manejo de buffers, el pequeño motor para la máquina de estados, los parsers y el manejador de selectores provistos por la cátedra. Estas herramientas nos permitieron realizar el desarrollo desde un punto de partida más completo en cuanto a utilidades.