Protocolos de comunicación

Trabajo práctico especial

Grupo 10

Alumnos:

- Gonzalo Beade - 61223
- Salvador Castagnino - Legajo 60590
- Marcos Dedeu - Legajo 60469
- Santiago Hadad - Legajo 61505

Primer Cuatrimestre 2022

Índice

Protocolos y aplicaciones desarrolladas: **3**

Problemas encontrados y arquitectura de la implementación: **3**

Limitaciones de las aplicaciones y posibles extensiones: **4**

Conclusiones **5**

Ejemplos de prueba **5**

Generación de los ejecutables **7**

Ejemplos de CoolProtocol: **7**

Diseño propuesto: **8**

Prueba de stress realizadas: **9**

Correctitud de los datos recibidos: 9

Cantidad de conexiones concurrentes: 10

Performance integral del servidor en estado de máxima concurrencia: 10

Protocolos y aplicaciones desarrolladas:

Se desarrolló un servidor que atiende dos protocolos de manera no bloqueante multiplexada. El mismo es accesible tanto con ipv4 como con ipv6. Cuenta con robustez a la hora de resolver nombre de dominios, timeout para evitar ser vulnerable ante prácticas ofensivas (Dos), ante las cuales es especialmente vulnerable por tener un máximo de conexiones concurrentes del orden de FD_SETSIZE .

El primero de ellos es socks5 [RFC 1928], con usuario y contraseña como método de autenticación[RFC 1929]. El mismo satisface la necesidad de nuestros usuarios de tener conectividad ,brindando la posibilidad de que el tráfico “salga” a través de nuestro servidor. Dichas conexiones, son monitoreadas por el servidor permitiendo, entre otras cosas, observar usuarios y contraseñas de conexiones con servidores pop3 [RFC 1939], y recolectar métricas que pueden ser de interés (como bytes transferidos, conexiones actuales, conexiones que realiza un cliente).

Por otra parte, atiende un protocolo para gestión del servidor de manera remota. El mismo llamado Cool Protocol [RFC] brinda la posibilidad a los administradores del sistema de configurar el servidor en tiempo de ejecución.

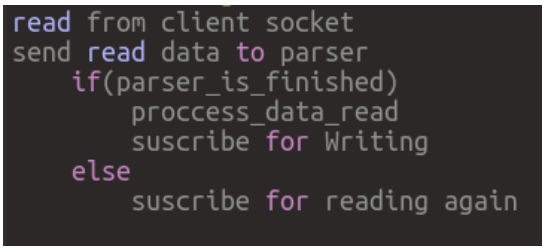
Por último, se implementó una aplicación cliente de dicho protocolo. La misma es un intérprete de comandos que actúa dependiendo de lo solicitado por el administrador. Al ser un cliente no fue desarrollado de manera no bloqueante, pues, únicamente maneja entrada estándar y los sockets que la conectan con el servidor. Además de cambiar la configuración del servidor, la aplicación cliente permite recuperar las métricas obtenidas y los usuarios que tienen acceso a la red mediante el proxy socks.

Problemas encontrados y arquitectura de la implementación:

El trabajo práctico representó un verdadero desafío desde el primer momento. Sumado a la inexperiencia y desconocimiento de la API que provee UNIX para el manejo de sockets, estaba el problema conceptual nunca antes enfrentado de desarrollar software que actúe de acuerdo con el estado en el que se encuentra y con la información que recibe. Una dificultad no menor es el concepto de flujo de bytes que presenta un protocolo de aplicación montado sobre TCP.

La arquitectura general puede entenderse con tres componentes separados que cooperan constantemente. El primero de ellos es una máquina de estados (a.k.a proxy_state_machine.h) que brinda la posibilidad de realizar diferentes acciones dependiendo del estado en que se encuentra la comunicación con el cliente. Cada estado describe las acciones que se deben llevar a cabo en el caso de recibir información o tener que mandar información.

Si bien las operaciones que se realizan luego de recibir información varían, todas siguen un algoritmo general:



Por su parte las operaciones de escrituras escriben todo lo permitido hasta que al fin pudieron vaciar sus buffers de salidas. Las funciones de procesamiento realizan lo solicitado por el cliente ,validan credenciales, y generan la respuesta que luego será enviada.

Por otra parte, se enfrentaron otros problemas que fueron surgiendo una vez la arquitectura había sido definida. Un primer problema fue que la información de cada conexión se encontraba dispersa en diferentes variables, lo cual era propenso al error y dificulta el entendimiento del código. Se resolvió compactando toda la información en una estructura.

Por usar TCP, no se puede asegurar que toda la información se mande en un solo llamado a send. Por lo tanto se manejan buffers donde se almacena la información que debe ser enviada hasta que pueda enviarse por completo. El tamaño de dicho buffer es de 2 KB. El módulo de procesamiento escribe las respuestas generadas en dicho buffer, y si no encuentra espacio se pierde información. Sin embargo, con este espacio nunca se va a dar esta situación, pues para generar los pocos bytes que demanda socks en sus responses el espacio es excesivo.

Para finalizar, consideramos pertinente mencionar que la definicion de nuestro protocolo fue basada en el protocolo que expone el RFC 1928. Este nos sirvió de guía para obtener una definición minuciosa del mismo que permita a terceros implementar tanto servidores como clientes de nuestro protocolo.

Limitaciones de las aplicaciones y posibles extensiones:

Naturalmente, por haber sido desarrollada en un periodo no tan extenso, notamos limitaciones en las aplicaciones que implementamos.

En el caso del servidor, la cantidad de conexiones simultáneas que soporta no es escalable en la actualidad. En una siguiente iteración, podría reemplazarse el uso de select por poll tal como lo recomienda el manual para programadores de linux. Haciendo esto se podría incrementar más de 60 veces la cantidad de conexiones concurrentes soportadas (pasaría a estar limitado por el límite de files descriptor que establezca el sistema operativo por proceso).

Por otra parte, si bien el servidor cuenta con un mecanismo para detectar clientes no activos, y cerrarles la conexión. Sería óptimo que el servidor, también lleve registro de las conexiones que intenta un mismo host, de modo tal que si excede cierto límite en un periodo de tiempo,se pueda actuar en consecuencia (cerrando la conexión automáticamente, filtrando la ip con un firewall).

Además, el servidor no cumple con todos los requisitos que detalla el RFC. Entre otras cosas no provee soporte para autenticación mediante GSSAPI, y no procesa solicitudes del tipo BIND, UDP ASSOCIATE.

En cuanto a las funcionalidades que brinda el protocolo de gestión del servidor es conveniente hacer hincapié en que las métricas recolectadas por el servidor son volátiles. Persistir dichas métricas brindaría la posibilidad de que el servidor pase a ser multihoming y que los datos sean consistentes ,válidos y globales. Sin embargo, esto conlleva una dificultad de implementación que viene dado por la concurrencia, y el posible bloqueo del servidor que la misma trae aparejada.

Para finalizar, cabe destacar que si bien el cliente desarrollado es funcional, y brinda un extenso abanico de posibilidades, en cuanto a la usabilidad se encuentra limitado e idealmente debería brindar una interfaz gráfica para el administrador.

Conclusiones

Para concluir, el desarrollo significó un gran desafío conceptual de entrada, que fue acompañado de un gran volumen de aprendizaje durante la implementación. El resultado es un servidor socks5 usable por clientes de terceros, el cual recolecta información útil para administradores de una red y brinda la posibilidad de configurarlo en tiempo de ejecución, lo cual es importante para no tener que detenerlo al mismo, causando una caída del internet a nuestros clientes.

Ejemplos de prueba

La ejecución del servidor se encuentra detallada en el repositorio. Dicho archivo debe abrirse con man.
Tal como se especifica en dicho archivo para ejecutar el servidor debe ejecutarse el comando :

```
shadad@shadad-ThinkBook-15-G2-ITL:~/Desktop/socks5-proxy$ ./socks5d
Listening on TCP port 1080 for ipv4 socks5
Listening on TCP port 1080 for ipv6 socks5
Listening on TCP port 8080 for ipv4 management
Listening on TCP port 8080 for ipv6 management
```

De esta forma se encuentra el servidor escuchando en las interfaces correspondientes. Dicho comportamiento es modificable.
Por defecto el servidor pide credenciales para permitir el acceso (según RFC 1929).

```
shadad@shadad-ThinkBook-15-G2-ITL:~$ curl -x socks5h://localhost:1080 fibertel.com.ar
curl: (7) No authentication method was acceptable. (It is quite likely that the SOCKS5 server wanted a username/password, since
 none was supplied to the server on this connection.)
```

Esto puede desactivarse desde el cliente. La ejecución del cliente se encuentra especificada en este archivo. Dicho archivo debe ser abierto con man.
Ejecutando los siguiente comandos.

```
shadad@shadad-ThinkBook-15-G2-ITL:~/Desktop/socks5-proxy$ ./client -P 8080 -6 -L ::1
Enter username: shadad
Enter password: shadad

============================================================
Welcome to the super cool sock5 proxy configuration.

Enter "help" in the command prompt to see the posible configuration commands.
Enter "quit" in the command prompt to terminate the session.
============================================================

> help
Super cool socks5 proxy client
Commands can be of two types, queries or modifiers
Entries follow the format: <command> - <description>

Query methods:
gthc - Get Total Historic Connections
gcc - Get Current Connections
gmcc - Get Max. Concurrent Connections
gtbs - Get Total Bytes Sent
gtbr - Get Total Bytes Received
gnuc - Get Number of Users Connected
gul - Get User List

Modification methods:
au - Add User
ru - Remove User
eps - Enable Password Spoofing
dps - Disable Password Spoofing
aa - Activate Authentication
da - Deactivate Authentication
> da

Modification Successful!
>
```

Las opciones de ejecución del cliente pueden verse con más detalle aquí.
Ahora probamos navegar sin credenciales:

```
shadad@shadad-ThinkBook-15-G2-ITL:~/Desktop/socks5-proxy$ curl -x socks5h://localhost:1080 google.com.ar
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com.ar/">here</A>.
</BODY></HTML>
```

Podemos comprobar que la conexión paso por el servidor.

```
shadad@shadad-ThinkBook-15-G2-ITL:~/Desktop/socks5-proxy$ ./socks5d
Listening on TCP port 1080 for ipv4 socks5
Listening on TCP port 1080 for ipv6 socks5
Listening on TCP port 8080 for ipv4 management
Listening on TCP port 8080 for ipv6 management
2022-06-19T18-28-50Z     unknown         A       127.0.0.1:46358         142.251.133.35:80       status=0
```

El formato de lo impreso se encuentra detallado aquí.

Generación de los ejecutables

Las instrucciones para generar los ejecutables pueden encontrarse en este archivo.

Ejemplos de CoolProtocol:

El cliente como se ha mencionado anteriormente cumple la función de modificar el estado del servidor en ejecución mediante el protocolo ideado e implementado.
Una posibilidad es la de eliminar un usuario que tiene acceso al servidor proxy para navegar.

```
> ru
Enter username: salva

Modification Successful!
```

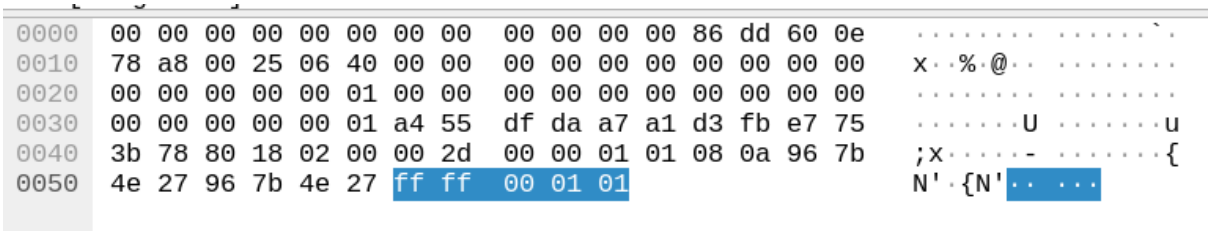
Dicha petición por entrada estándar al cliente se traduce en el siguiente flujo de bytes tal y como lo describe el RFC.

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 05   ........ ......`.
0010  08 cd 00 21 06 40 00 00  00 00 00 00 00 00 00 00   ...!.@.. ........
0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 01 df da  a4 55 e7 75 3b 70 a7 a1   ........ .U.u;p..
0040  d3 fb 80 18 02 00 00 29  00 00 01 01 08 0a 96 7b   .......) .......{
0050  4e 27 96 79 c6 cf be                              N'.y...
```

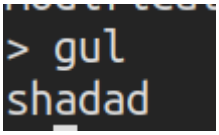
```
0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 05   ........ ......`.
0010  08 cd 00 21 06 40 00 00  00 00 00 00 00 00 00 00   ...!.@.. ........
0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 01 df da  a4 55 e7 75 3b 71 a7 a1   ........ .U.u;q..
0040  d3 fb 80 18 02 00 00 29  00 00 01 01 08 0a 96 7b   .......) .......{
0050  4e 27 96 7b 4e 27 01                              N'.{N'.
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 05   ........ ......`.
0010  08 cd 00 26 06 40 00 00  00 00 00 00 00 00 00 00   ...&.@.. ........
0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 01 df da  a4 55 e7 75 3b 72 a7 a1   ........ .U.u;r..
0040  d3 fb 80 18 02 00 00 2e  00 00 01 01 08 0a 96 7b   ........ .......{
0050  4e 27 96 7b 4e 27 05 73  61 6c 76 61               N'.{N'.s alva
```

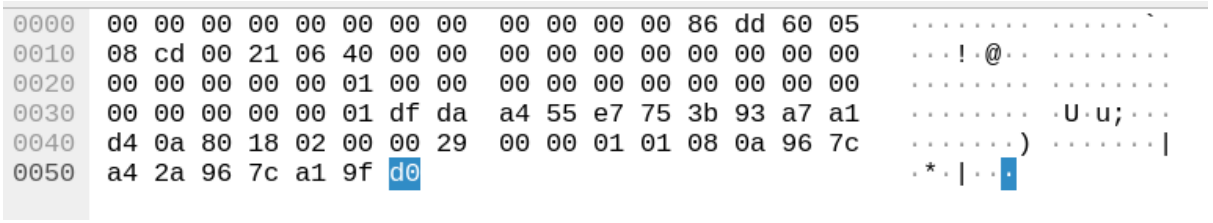
El servidor recibe la petición, la procesa y genera la siguiente respuesta para el cliente. Como se describió, este paquete indica que la operación fue exitosa.

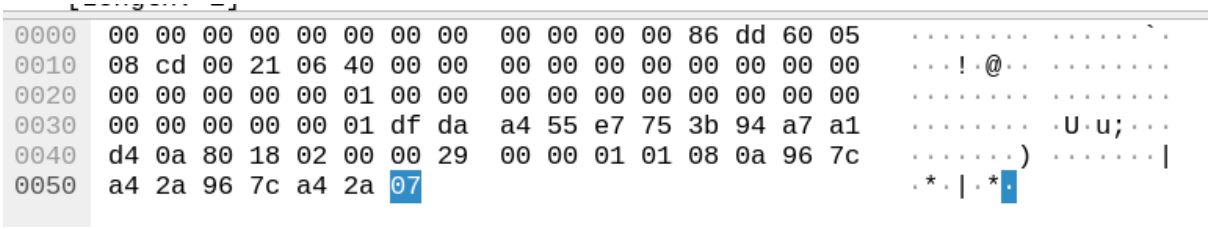


Por otra parte para obtener la lista de usuarios:

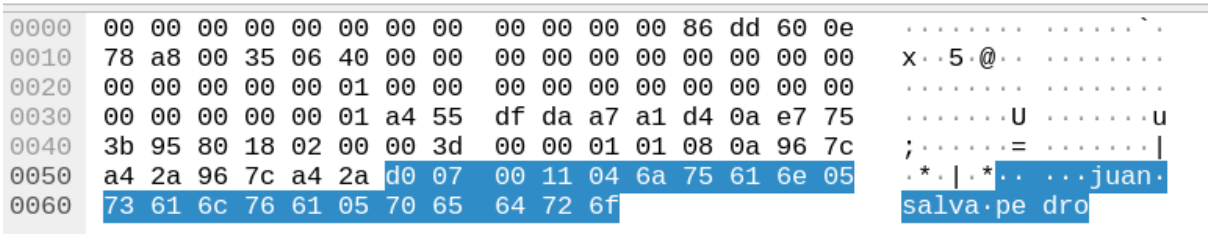


Se generan los siguientes paquetes:





La respuesta recibida por parte del servidor:



Diseño propuesto:

Como se mencionó anteriormente, la solución propuesta debe entenderse como tres componentes que cooperan.

El núcleo de la aplicación pasa por el selector el cual cumple la función de monitorear los sockets y retomar la ejecución cuando se puede realizar alguna operación de entrada y salida en alguno de ellos. De esta manera, cuando se puede leer o escribir información, el selector ejecuta alguna función asociada al socket.

Por otra parte, para personalizar las acciones a ejecutar cuando el selector interrumpe se usa una máquina de estados, la cual asocia un estado de conexión con un conjunto de funciones (por ejemplo, apenas iniciada la conexión como cliente).

En las funciones asociadas se necesita un componente que interprete la información recibida, por ello, al entrar en un estado suele inicializarse el parser correspondiente (línea 7-8). En el momento en el que el parser comunica que se ha recibido una unidad lógica (un flujo de bytes que tienen sentido en el marco del protocolo), se procesa lo recibido, y se avisa al selector que se quiere escribir la respuesta (línea 36-39).

Esta secuencia de pasos, es la guía para todas las operaciones que realiza el servidor tanto en socks5, como en coolProtocol.

Prueba de stress realizadas:

El servidor fue sometido a una amplia gama de pruebas en la cuales se validó la correctitud de los datos recibidos, la performance en cuanto a tiempo de ejecución, y la capacidad de atender clientes simultáneamente.

Correctitud de los datos recibidos:

Se ejecuta una solicitud sin pasar por el servidor y se calcula el hash (md5sum) de lo recibido. Luego se realiza la misma solicitud a través del servidor y se calcula el hash. Se comparan ambos resultados.

```
shadad@shadad-ThinkBook-15-G2-ITL:~$ curl http://pawserver.it.itba.edu.ar/paw-2022a-01/buyer/market | md5sum
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 25265    0 25265    0     0  84498      0 --:--:-- --:--:-- --:--:-- 84498
aa8ddc520556b74f392443d670fa674b  -
shadad@shadad-ThinkBook-15-G2-ITL:~$ curl -x socks5://localhost:1080 http://pawserver.it.itba.edu.ar/paw-2022a-01/buyer/market
| md5sum
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 25265    0 25265    0     0  70572      0 --:--:-- --:--:-- --:--:-- 70376
aa8ddc520556b74f392443d670fa674b  -
```

Chequeamos que se imprima bien la información de conexión, mediante un cliente de dns:

```
shadad@shadad-ThinkBook-15-G2-ITL:~/Desktop/socks5-proxy$ ./socks5d -N
Listening on TCP port 1080 for ipv4 socks5
Listening on TCP port 1080 for ipv6 socks5
Listening on TCP port 8080 for ipv4 management
Listening on TCP port 8080 for ipv6 management
2022-06-19T19-40-27Z     unknown         A       127.0.0.1:46360         200.5.121.138:80       status=0
```

```
shadad@shadad-ThinkBook-15-G2-ITL:~$ dig pawserver.it.itba.edu.ar

; <<>> DiG 9.16.1-Ubuntu <<>> pawserver.it.itba.edu.ar
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41895
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;pawserver.it.itba.edu.ar.      IN      A

;; ANSWER SECTION:
pawserver.it.itba.edu.ar. 3288  IN      A       200.5.121.138
```

Esta idea se sistematiza con el script hash.sh.

Cantidad de conexiones concurrentes:

Para testear que el servidor puede atender un gran número de conexiones concurrentes se usa el script dnsResolving.sh. El mismo genera en background la cantidad de procesos pasada por parámetro, y estos realizan pedidos mediante curl de manera sistemática. Luego de diez segundos finaliza y manda a todos los procesos generados, de manera que también se testea que el cliente cierre la conexión. Ejecutando el servidor con strace, puede verse que se cierran todas las conexiones y quedan en el selector únicamente los sockets pasivos.

Performance integral del servidor en estado de máxima concurrencia:

Como fue mencionado anteriormente, la cantidad de peticiones simultáneas se encuentra limitada por la siguiente ecuación.

$$\#Máx\ conexiones\ concurrentes = \frac{FD\_SETSIZE - (\#sockets\ pasivos + STDIN\_FILENO + STDERR\_FILENO)}{2} = 509$$

, donde la cantidad de sockets pasivos es cuatro (i.e dos para socks5 y dos para coolProtocol).

Previamente se obtuvo el tiempo promedio para solicitudes http, sin pasar por el servidor. Obteniendo como resultado 0,31595 segundos.

```
shadad@shadad-ThinkBook-15-G2-ITL:~/Desktop/socks5-proxy$ ./timeWithoutProxy.sh 100 &> time.txt
shadad@shadad-ThinkBook-15-G2-ITL:~/Desktop/socks5-proxy$ cat time.txt | awk '{total = total + $1}END{print total}'
31,895
```

Luego fueron comparados con los resultados obtenidos mediante el uso de la herramienta jmeter. Dichos resultados pueden verse en el repositorio. El porcentaje de error es 0%, y en máxima concurrencia el tiempo promedio se encuentra debajo del doble.