Trabajo práctico especial

72.07 - “Protocolos de Comunicación”

2022 - 2Q

Grupo IV

Integrantes:
Benvenuto, Agustín - 61448
Cornidez, Milagros - 61432
Galarza, Agustín Ezequiel - 61481
Ortu, Agustina Sol - 61548

22 de Noviembre de 2022

Índice

Descripción detallada de los protocolos y aplicaciones desarrolladas

Se diseñó e implementó un servidor proxy para el protocolo *SOCKS5v5* que atiende clientes desde direcciones IPv4 e IPv6 de manera no bloqueante y multiplexada, que también soporta la comunicación mediante un protocolo denominado YAP (Yet Another Protocol). Este último permite operaciones como: listado de usuarios, obtención de cantidad total de conexiones históricas, concurrentes y bytes enviados, adición y remoción de usuarios y cambios de timeout y de tamaño de buffer.

Además soporta distintos tipos de datos como: ***int*** (u_int8, u_int16, u_int32), ***string***(str_leng, str_value) y ***string list*** (content_len, values), y solicita un usuario y contraseña para la autenticación.

Luego de una correcta autenticación el usuario enviará comandos a través de su consola partiendo de un comando (cmd) el cual puede estar seguido de un argumento en caso de que sea necesario. En caso de error el servidor retorna ‘0xFF’ ‘0xFF’.

A continuación se listan los comandos permitidos:

	REQUEST	**RESPONSE**
Usuarios	cmd se envía como '0x01' y no requiere argumentos	cmd se mantiene e imprime la lista de usuarios existentes
Métricas	cmd se envía como '0x02' y requiere como argumento la métrica la cual puede ser: -'0x00': Conexiones históricas -'0x01': Conexiones concurrentes -'0x03': Total de KB enviados	cmd y la métrica se mantienen e imprime el valor de métrica solicitado
Agregar usuario	cmd se envía como '0x03' y requiere como argumento un username y una password.	cmd se mantiene e imprime un status el cual puede ser: -'0x00': OK -'0x01': Se alcanzó la cantidad máxima de usuario -'0x03': El usuario ya existe
Eliminar usuario	cmd se envía como '0x04' y requiere como argumento un username.	cmd se mantiene e imprime un status el cual puede ser: -'0x00': OK

		-'0x01': Usuario invalido
Configuración	cmd se envía como '0x05' y requiere como dos argumentos, config y segundos o buff_size según corresponda En el caso de config='0x00' se refiere al timeout y requiere de los segundos como argumento. En cambio en el caso de config='0x01' se refiere al tamaño de buffer a modificar y requiere el buff_size	cmd y config se mantienen e imprime un status el cual puede ser: -'0x00': OK, configuración actualizada -'0x01': Fallo, no se puedo actualizar la configuración.

Se diseñó un cliente el cual se encarga de probar todas las capacidades del protocolo previamente descrito y el protocolo *socks* (soportando requests del tipo curl y pop3) con un manual de ayuda al usuario para simplificar el uso del mismo.

Problemas encontrados durante el diseño y la implementación

El mayor problema encontrado durante el diseño e implementación del trabajo fue el flujo de trabajo del proxy server. Dado que hay tantos estados distintos con diferentes acciones necesarias para cada estado sabíamos que iba a ser un problema si las bases del comportamiento no estaban claras y de fácil entendimiento para cuando debíamos inicializar o liberar espacios de memoria.

Para solucionar este problema nos concentramos en hacer uso de una máquina de estados que nos permita trabajar de forma más granular con las operaciones a realizar en cada etapa del flujo de comunicación entre cliente y servidor. De todas formas, esta implementación también trajo sus propios inconvenientes al momento de manejar la alocación y liberación de recursos según fuera necesario.

Por otro lado, también se nos presentaron complicaciones al momento del parseo de los mensajes entre cliente y servidor, principalmente en cuanto al aspecto de tener que manejar los mismos como un stream de datos, lo cuál nos obligo a mantener estados dentro de cada parser para que la información no se pierda.

Por suerte este problema no duró mucho tiempo al implementar un parser que consuma el buffer de a un byte, guardando el estado en el que estábamos y chequeando si habíamos terminado. Al fin y al cabo, otra máquina de estados.

Además, tuvimos algunos problemas a la hora de determinar nuestro protocolo YAP (Yet Another Protocol) dado que en un inicio lo intentamos hacer demasiado general. Por suerte esto fue corregido previo a la implementación con ayuda del profesor en una de las clases de consulta.

Finalmente, pero no menos importante, este trabajo no se podría haber completado sin la ayuda del debugger y Wireshark.

Limitaciones de la aplicación

- Capacidad máxima de clientes: 500
- Capacidad máxima de admins: 1
- Al cambiar el timeout mediante un cambio de configuración en tiempo de ejecución no se cambian los timeouts de clientes que ya estén conectados, solo de clientes que entren nuevos.
- En la request de SOCKS, CMD acepta únicamente a CONNECT

Posibles extensiones

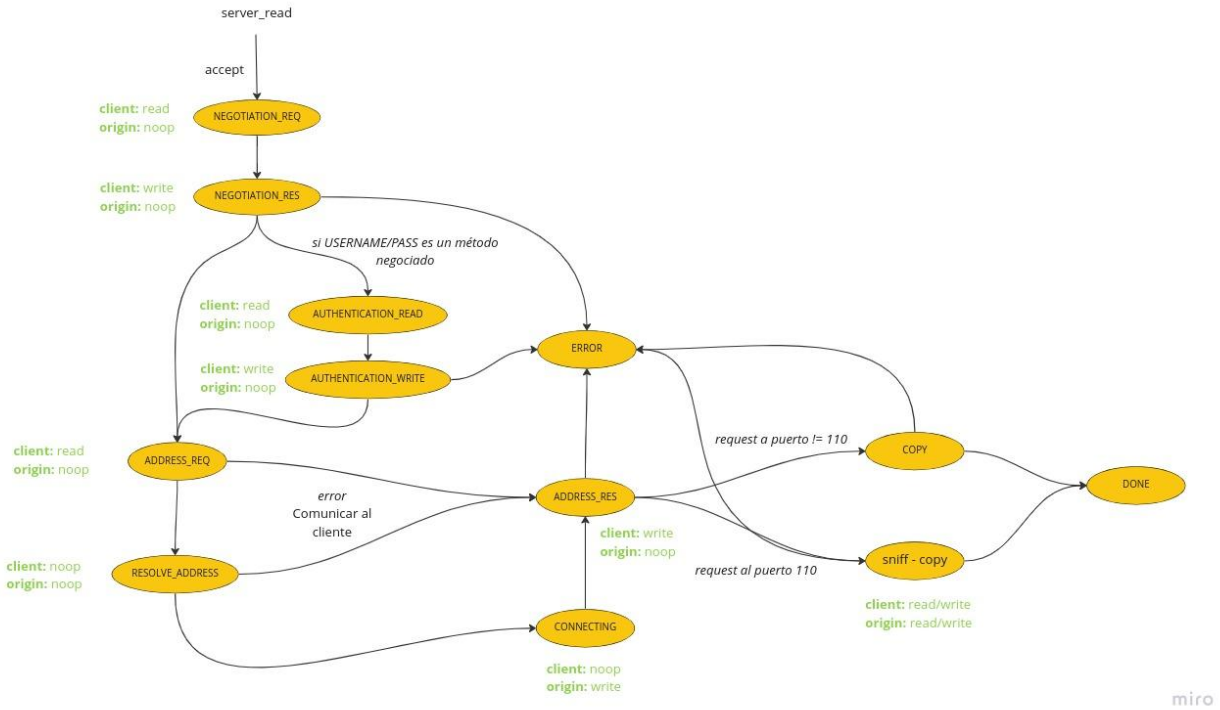
- Uso de *epoll* en la implementación para poder soportar mayor cantidad de clientes.
- Extensión del protocolo YAP para poder editar usuarios y mayor cantidad de configuraciones.
- Almacenar los usuarios, valores de métricas y logs archivos externos para mantener su persistencia y que no se elimine esa información al apagar el servidor.
- Sistema que elimine usuarios que no estén realizando peticiones, es decir, que estén inactivos.
- Agregado de mayor cantidad de administradores con capacidad de alteración de las configuraciones de manera concurrente.

Conclusión

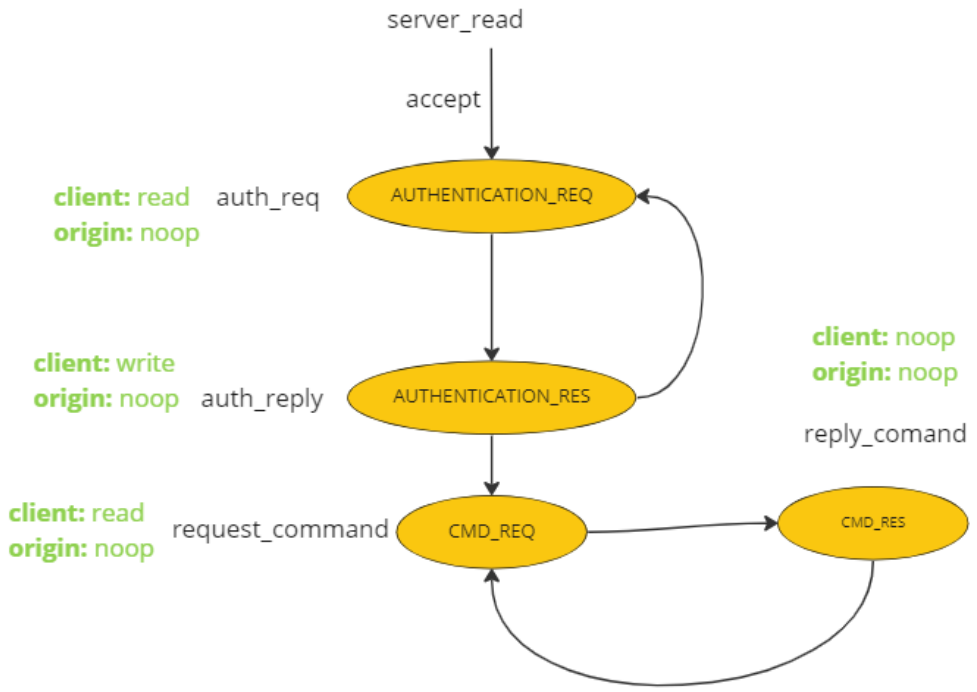
El desarrollo del trabajo fue un gran desafío para el equipo ya que requirió de una investigación más allá de lo aprendido en la cursada de la materia. Más allá de esto se logró cumplir un servidor que atiende de manera no bloqueante y multiplexada, un protocolo de alta complejidad y un cliente que corrobora y asegura el funcionamiento de lo que tanto el protocolo como el servidor se proponen.

Documento de diseño del proyecto

Socks server:



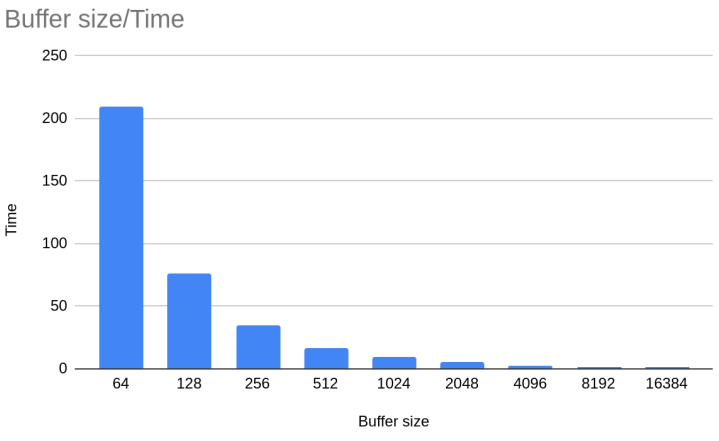
Admin server



Ejemplos de prueba

Veamos cómo varía la velocidad de transferencia dependiendo del tamaño del buffer y compararlo con el uso con otro proxy.

Nuestro proxy	
Buffer(bytes)	Tiempo(s)
64	210
128	76
256	35
512	17
1024	10
2048	5
4096	2
8192	1
16384	1



Tiempo utilizando un proxy socks5 con un túnel dinámico: ***3s***

Instrucciones para la configuración

Primero y principal se procede a listar los requerimientos de compilación:

- GCC (>=9.4.0)
- Make (>=4.2.1)

Luego hay que dirigirse al root del proyecto y correr:

```
make
```

Lo cual generará las carpetas build/ y bin/, esta última con los binarios del servidor (socks5d) y del client (socks5c). A partir de ese momento ya puede realizar la compilación del servidor y/o del cliente como se explicará a continuación en la ***Guía de instalación.***

Guía de instalación

Compilación

Para poder compilar el servidor se debe ejecutar:

```
make server
```

Para poder compilar el cliente se debe ejecutar:

```
make client
```

Para poder compilar el servidor y el cliente se debe ejecutar:

```
make all
```

En el root del proyecto.

Ejecución

Servidor

```
./bin/socks5d <opciones>
```

Opciones:

-h	→ Imprime la ayuda y termina
-p <SOCKS port>	→ Puerto entrante conexiones SOCKS
-P <conf port>	→ Puerto entrante conexiones admin
-u <name>:<pass>	→ Usuario y contraseña de usuario que puede usar el proxy. Hasta 10.
-v	→ Imprime la información sobre la versión y termina

Cliente

```
./bin/socks5c <opciones>
```

Opciones:

-Y indica protocolo YAM

-h → Ayuda

-L <address> → Especifica la dirección a conectarse. REQUERIDO

-P <port> → Especifica el puerto a conectarse.

[-4 | -6] → Especifica versión a utilizar

-S indica SOCKS

-h → Ayuda

-P <port> → Especifica el puerto a conectarse. REQUERIDO

-a <type> → Tipo de dirección.
<4> para IPv4;
<DN> para nombre de dominio;
<6> para IPv6

-u <user>:<password> → Autenticacion de usuario

En caso de no indicar version/tipo de dirección se toma a IPv4 como default