

72.07 - Protocolos de Comunicación

Proyecto Especial

Instituto Tecnológico de Buenos Aires



Grupo 3

Integrantes:

- Garrós, Celestino (64375)
- Pedemonte Berthoud, Ignacio (64908)
- Ruckauf, Federico Ignacio (64356)
- Weitz, Leo (64365)

Fecha de Entrega: 15/07/2025

Tabla de Contenidos

1. Abstracto.....	3
2. Protocolos y Aplicaciones.....	3
2.1 Servidor Proxy SOCKSv5.....	3
2.2 Protocolo de Monitoreo.....	3
2.3 Aplicación Cliente.....	4
3. Proceso de Desarrollo.....	4
3.1 Echo Server Bloqueante.....	5
3.2 Echo Server No Bloqueante.....	5
3.3 Implementación de SOCKSv5.....	5
3.4 Desarrollo del Protocolo de Administración y Aplicación Cliente.....	6
4. Problemas Encontrados.....	6
5. Limitaciones.....	7
6. Posibles Extensiones.....	7
7. Conclusiones.....	8
8. Guía de Instalación.....	8
9. Guía de Configuración.....	8
10. Ejemplos de Prueba.....	9
11. Ejemplos de Configuración y Monitoreo.....	10
12. Documento de Diseño del Proyecto.....	11
13. Bibliografía.....	12

1. Abstracto

El presente documento detalla el proceso de desarrollo de un servidor proxy que implementa el protocolo SOCKSv5 con sus correspondientes funcionalidades junto con otro protocolo adicional de administración del mismo, dentro del marco de la materia Protocolos de Comunicación en el Instituto Tecnológico de Buenos Aires.

2. Protocolos y Aplicaciones

A continuación se describen los protocolos y aplicaciones desarrolladas para el proyecto especial de la materia, con sus funcionalidades y descripciones correspondientes. Este consta de un servidor proxy que implementa el protocolo SOCKSv5, así como un protocolo de monitoreo y una aplicación cliente. Este servidor se desarrolló bajo el lineamiento de atender llamados y realizar conexiones de manera no bloqueante, lo que guió en muchos casos las decisiones frente a inconvenientes.

2.1 Servidor Proxy SOCKSv5

El servidor implementa crea al inicializarse un socket pasivo para escuchar conexiones entrantes en el puerto 1080 por default. Para cada una, realiza el proceso correspondiente de handshake tal como se especifica en el [RFC 1928](#), recibiendo a través de los buffers la información enviada por el cliente, y luego enviando por el mismo medio la respuesta.

Una vez completado el proceso anterior, se pasa a un estado de espera del servidor para que el usuario envíe el request para un origen determinado, nuevamente con formato acorde al especificado en el RFC. El recibir el pedido, se encarga de realizar la resolución correspondiente, iniciar la conexión y responder al usuario el estado. En este punto se tomaron los mismos valores de retorno del protocolo, aprovechando para darle información detallada al cliente.

Luego de establecer la conexión, se pasa a un estado de forwarding entre cliente y origen, donde se toma provecho de los códigos de buffers circulares provistos, junto con el selector y la state machine, para realizar el traspaso de información de manera no bloqueante. En caso de acabar o surgir un error, se cuenta con dos estados más correspondientes a la finalización de manera correcta o incorrecta, donde se cierra la conexión y se libera memoria.

Para la autenticación de usuarios se decidió persistirlos en un archivo binario que serializara los pares usuario-contraseña. Para la implementación del mismo, y teniendo en cuenta que el servidor podría ser utilizado por un gran número de usuarios, se desarrolló un hashmap y se utilizaron los algoritmos djb2 y closed hashing, buscando así aumentar lo máximo posible la eficiencia.

2.2 Protocolo de Monitoreo

El protocolo para el monitoreo y la configuración del servidor es un protocolo de tipo texto, orientado a conexión y desarrollado sobre TCP. Los comandos y los parámetros utilizan sólo caracteres a-z, A-Z, 0-9, _, y sus nombres deben tener entre 1 y 255 caracteres

inclusive. El flujo de la interacción de un cliente con un servidor utilizando este protocolo es el siguiente comienza con un handshake de la siguiente manera:

>S: [PROVIDE_CREDENTIALS_BANNER] [\n]

>C: [USERNAME]:[PASSWORD] [\n]

>S: [VALID/INVALID_BANNER] [\n]

En el caso que las credenciales sean válidas, el servidor pasa a aceptar comandos y actuar en base a ellos. Caso contrario, el servidor corta la conexión. Cada comando es una línea enviada por el cliente. El servidor responde según el comando:

- **STATS:** El servidor responde con estadísticas generales (ejemplo: conexiones históricas, bytes transferidos, etc.).
- **CONNECTIONS:** El servidor devuelve la lista de conexiones actuales y su estado.
- **USERS:** El servidor devuelve la lista de usuarios activos.
- **CONFIG <param> <value>:** El servidor responde OK si el cambio fue exitoso, ERR si hubo un error.
- **ACCESS_LOG <username>:** Si NO se especificó el parámetro username, el servidor retorna la lista de sitios a los que accedió cada usuario a través del servidor. Si se especificó el parámetro username, el servidor retorna la lista de sitios a los que accedió el usuario a través del servidor.
- **QUIT:** Cierra la conexión.

2.3 Aplicación Cliente


Se desarrolló adicionalmente una aplicación cliente, es decir una herramienta de línea de comandos diseñada para facilitar visualmente la interacción con el servidor de monitoreo del proxy SOCKS5.

Al conectarse, se le solicita al cliente que se autentique, permitiendo (en caso de ser exitosa) la ejecución de comandos. Estos últimos coinciden con los explicitados en la sección 2.2 *Protocolo de Monitoreo*. Esta aplicación se puede correr ejecutando el archivo `monitor_client` luego de correr el proyecto con `make`.

3. Proceso de Desarrollo

Para comenzar con el desarrollo del proyecto, se decidió tomar un enfoque progresivo, implementando pequeñas funcionalidades en su forma más sencilla en búsqueda de contar en todo momento con un servidor funcional. Teniendo en cuenta, se separó el proceso en las siguientes etapas.

3.1 Echo Server Bloqueante

La primera etapa se basó en desarrollar un Echo Server Bloqueante, el cual recibiera datos del usuario y los devolviera, en un principio, de forma bloqueante, aceptando conexiones TCP. Se tomó como punto de partida el código provisto por la cátedra en la clase teórica de programación en sockets ( 10 Sockets).

Como se puede observar en el código provisto, se tienen funciones generadas que el servidor se bloquee, ya sea esperando por la información (`recv()`) o realizando el envío (`send()`). Si bien esto no cumple con la consigna, fue un importante punto de partida, ya que sirvió para tener una base de servidor para comenzar a modificar. Por ejemplo, de allí posteriormente se reutilizó la función de creación del socket TCP `setupTCPServerSocket()`.

3.2 Echo Server No Bloqueante

Una vez logrado el servidor funcional, el siguiente paso fue la modificación del mismo para que la interacción con el usuario no fuera bloqueante y que pudiera atender múltiples conexiones. Para esto, fue fundamental el patch provisto por la cátedra conteniendo un selector. El mismo permite atender los sockets cuando estos estuvieran listos para realizar alguna operación.

Esta etapa fue crucial para el desarrollo del servidor, ya que la configuración del selector se mantuvo incluso hasta en la versión entregada del proyecto, con las correspondientes modificaciones según la etapa. Además, permitió comenzar a entender la implementación provista, lo que sería muy importante para los siguientes pasos en el desarrollo, ya que requerirían en gran parte de su uso.

3.3 Implementación de SOCKSv5

El siguiente paso en el desarrollo del servidor proxy, fue la adaptación para comunicación mediante el protocolo SOCKSv5, para lo que primero se debió leer el [\[RFC 1928\]](#). Luego de implementar lo descrito en el RFC del protocolo, se incorporó el manejo del método de autenticación por usuario/contraseña, en base a lo especificado en el [\[RFC 1929\]](#). En base a eso, se tomó el siguiente comando de prueba, para validar que la implementación sea correcta y se pueda llevar a cabo la comunicación en dicho protocolo.

```
curl -4 --proxy socks5h://user:pass@127.0.0.1:1080 http://example.org
```

En esta etapa, se tomó también un enfoque progresivo, haciendo primero toda la lógica del handshake planteada en el RFC de SOCKSv5, luego la conexión y finalmente el estado de forwarding de datos entre el origen y el cliente. Para todas ellas, fue fundamental el uso la state machine provista en un patch por la cátedra, ya que facilitó mucho el manejo de estas distintas etapas de comunicación y se acopló perfectamente con el selector.

Por otro lado, fue de gran utilidad la utilización del buffer provisto en otro patch por la cátedra, ya que permitió un mejor manejo de la información entre el cliente y el servidor, y entre el servidor y el origen en las distintas etapas del proceso de comunicación. Se optó por asignar un buffer de escritura y uno de lectura tanto para el cliente como para el origen, para la información que fuera llegando de cada uno.

Nuevamente, esta etapa fue fundamental para la comprensión de utilidades de los patches provistos, que luego serían fundamentales en los siguientes pasos de desarrollo del servidor, como la state machine y el buffer circular.

3.4 Desarrollo del Protocolo de Administración y Aplicación Cliente

Una vez establecida la base del servidor SOCKSv5, se procedió con el desarrollo del protocolo de administración. Este permite conectarse al servidor a través de un puerto dedicado y consultar información de estado, además de realizar acciones de administración como la adición o eliminación de usuarios autorizados.

En pos del diseño del protocolo de administración, se desarrolló un documento de diseño en el cual se fue especificando la estructura del protocolo, sus cualidades y los comandos que ofrecería. A su vez se desarrollaron ejemplos de interacciones entre clientes y servidor, para luego en base a esto implementar un servidor que hable este protocolo.

Finalmente, se desarrolló una aplicación cliente simple que interactúa con el protocolo de administración. Esta aplicación permite enviar comandos desde consola para consultar métricas, listar usuarios, y realizar operaciones administrativas básicas.

4. Problemas Encontrados

Muchos de los problemas encontrados durante el proceso de desarrollo del servidor proxy fueron debido a la incorrecta incorporación de los patches provistos por la cátedra, o directamente por la falta de ellos. Por ejemplo, el incorrecto uso de la state machine llevó a tener en un principio dos estados para el handshake inicial del protocolo SOCKSv5, siendo estos HELLO_READ y HELLO_WRITE, que posteriormente se simplificaron en el uso correcto de la stm como se muestra en el siguiente fragmento:

```
{
    .state = STATE_HELLO,
    .on_read_ready = hello_read,
    .on_write_ready = hello_write,
},
```

Luego, se encontraron dificultades en los manejos de estados pero desde el lado de la lógica, llevando a cometer errores en distintos fragmentos del código, principalmente en el estado de forwarding entre cliente y origen y los seteos de intereses para el selector, lo que se pudo solucionar utilizando los logs, principalmente para debugging.

Otro problema fue el manejo de los close con la función `socks5_stm_close()` para la state machine, ya que se llamaba múltiples veces y se intentaba liberar memoria que ya no se encontraba ocupada. Para solucionarlo, se agregó el campo de referencias en el `struct socks5_connection` para llevar un conteo y hacer un correcto cierre y liberación.

Por otra parte, como fue mencionado en la corrección del simulacro de entrega, el flujo de la interacción con el servidor SOCKSv5 era ineficiente, al siempre hacer la secuencia `select`, `read`, `select` y `write`. Se intentó resolverlo haciendo que las funciones de lectura, en vez de devolver el mismo estado pero con el interés en `OP_WRITE`, devolvieran un llamado a la función de

escritura. Esto trajo race conditions entre los intereses y el selector. Finalmente se tuvo que hacer un intento de escritura al final de la función read en vez de directamente llamar a la función write, y en el caso de que este intento no fuera exitoso, ahora si volver al estado en OP_WRITE.

En el proceso también se detectó una posible vulnerabilidad a los ataques de tipo Denial of Service, ya que las conexiones abiertas inicialmente no se cerraban nunca. Esto dejaba abierta la posibilidad de realizar repetidas solicitudes a un servidor que jamás retorne y corte la conexión, saturándolo y bloqueando el servicio. La solución encontrada a esto fue un timeout implementado dentro del selector provisto por la cátedra, mediante el agregado de un timestamp de último uso para cada ítem del mismo y un borrado de las conexiones expiradas al momento de realizar el select.

5. Limitaciones

La primera limitación con la que cuenta el proyecto se presenta en la cantidad máxima de conexiones en simultáneo, dada la cantidad de file descriptors disponibles. El número se encuentra en 511, ya que se cuenta con 1024 file descriptors con 3 reservados, pero al no necesitar leer nada de stdin, se libera al inicializar el servidor.

Otra limitación importante es que no se implementó el soporte para el comando UDP ASSOCIATE del protocolo SOCKSv5. Se decidió no agregar esta funcionalidad ya que es necesario gestionar una asociación entre el cliente y un socket UDP que el servidor debe mantener activo, reenviar paquetes sin conexión previa, y resolver correctamente direcciones y puertos de destino, lo que agrega dificultad y no es fundamental para el funcionamiento.

En la implementación del servidor, se tomó como máximo de pares usuario:contraseña aceptados por parámetro el número especificado por la cátedra en la consigna, el cuál es diez. Esto se dejó según el requisito, pero es una limitación que se puede cambiar en el código del servidor.

6. Posibles Extensiones

En caso de continuar con el desarrollo del trabajo y la extensión de su funcionalidad resultaría de gran utilidad la presencia de una suite de testing. Permitiría, por ejemplo, evitar regresiones al cambiar detalles de funcionamiento interno (como podría ser el uso de poll o epoll para reemplazar el selector). Además, facilitaría que el agregado de nuevas funcionalidades siga principios de TDD, permitiendo organizar mejor el proyecto a medida que este crezca.

Otra funcionalidad que se podría agregar sería el soporte para el comando UDP ASSOCIATE del protocolo SOCKSv5, mencionado previamente en la sección de limitaciones. Esto permitiría al servidor actuar como proxy para datagramas UDP, habilitando el uso de aplicaciones que se basan en este protocolo.

También se podría implementar soporte para el comando BIND del protocolo SOCKSv5, que permite al servidor actuar como intermediario para conexiones entrantes hacia el cliente. Se decidió que, para el alcance del proyecto, esta funcionalidad quedara como una posible extensión en lugar de desarrollarla.

Por último, una posible extensión sería ampliar el protocolo de monitoreo y la funcionalidad de la aplicación cliente asociada. Se podrían incorporar comandos adicionales que permitan observar estadísticas detalladas del funcionamiento interno del servidor, como la cantidad de bytes transferidos por conexión, tiempos de sesión promedio, o el número de errores registrados. También se podrían agregar opciones de configuración remota, como modificar credenciales de usuarios o activar/desactivar logs en tiempo real.

7. Conclusiones

Como conclusión del proyecto, se pudo desarrollar el servidor proxy con sus respectivos protocolos, lo que permitió un mayor entendimiento y estudio de muchos de los conceptos estudiados durante la cursada de la materia. Se pudo realizar la práctica de la lectura de un RFC para la producción de un sistema que pudiera comunicarse correctamente en la parte de SOCKSv5 del servidor.

Por otro lado, se destaca la práctica recomendada por la cátedra de realizar un proceso de desarrollo progresivo, buscando siempre obtener un servidor o una aplicación funcional. Esto ahorró mayores frustraciones y permitió una mayor comprensión de cada funcionalidad agregada.

Por último, los códigos provistos por la cátedra fueron de gran ayuda para el desarrollo del proyecto, ya que aportaron funcionalidades fundamentales y permitieron un ahorro significativo de tiempo, ya que la única tarea que se debió realizar fue la del estudio de su correcta aplicación. Si bien muchos errores y conflictos surgieron de su mal uso, estas situaciones trajeron consigo mayores ventajas que desventajas.

8. Guía de Instalación

Para instalar el programa, primero se deberá clonar el repositorio git mediante el siguiente comando:

```
git clone git@github.com:ipedemonteb/tpe-protos.git
```

Una vez clonado el repositorio, se deben compilar los archivos para obtener los ejecutables del servidor y aplicación cliente ejecutando el comando `make` (correspondiente al Makefile incorporado). Para ejecutar el servidor y/o la aplicación, se tienen los siguientes comandos con los flags correspondientes especificados en la siguiente sección.:

```
./bin/server
```

```
./bin/monitor_client
```

9. Guía de Configuración

El servidor puede configurarse, como se mencionó en la sección anterior, mediante distintos flags al momento de su ejecución:

-h: Muestra un mensaje de ayuda con el uso del programa y finaliza la ejecución.

-l <SOCKS ADDR>: Define la dirección IP en la que el servidor escuchará por conexiones entrantes SOCKS (0.0.0.0 por defecto).

-p <SOCKS PORT>: Especifica el puerto en el que se aceptarán conexiones SOCKS (1080 por defecto).

-L <CONF ADDR>: Dirección IP en la que se expondrá el servicio de monitoreo y administración (127.0.0.1 por defecto).

-P <CONF PORT>: Puerto por el cual se aceptarán conexiones al servicio de monitoreo (8080 por defecto).

-u <NAME>:<PASS>: Agrega un usuario autorizado para usar el proxy SOCKS, junto con su contraseña. Se pueden definir hasta un máximo de 10 usuarios utilizando múltiples instancias de este flag.

-v: Imprime información sobre la versión del servidor y finaliza la ejecución.

También, se cuentan con los siguientes flags para configurar la aplicación cliente de monitoreo. Los mismos se deben agregar a la hora de correr el ejecutable:

-L <CONF ADDR>: Dirección IP del servidor de administración al que se desea conectar (127.0.0.1 por defecto).

-P <CONF PORT>: Puerto correspondiente al servicio de administración del servidor (8080 por defecto).

-h: Imprime un mensaje de ayuda con los posibles flags y finaliza la ejecución.

10. Ejemplos de Prueba

Para poder evaluar el funcionamiento del servidor socks5 desarrollado, se llevaron a cabo múltiples tipos de pruebas. En primera instancia, se hicieron pruebas de funcionalidad básica utilizando, como ya se mostró previamente, el comando curl utilizando el servidor como proxy:

```
curl -4 --socks5-hostname user:pass@127.0.0.1:1080 http://example.org/
```

Posteriormente, se llevaron a cabo pruebas de estrés, como la simulación de un acceso de quinientos clientes de manera simultánea al servidor a través de un script de bash, o el uso de curl para efectuar una descarga de un archivo de 100MB usando el servidor como proxy. En este último caso, se obtuvieron los siguientes resultados:

```
curl --socks5-basic --socks5-hostname 127.0.0.2:1080 --proxy-user papapapa:pepepepe https://nbg1-speed.hetzner.com/100MB.bin --output test.download
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 100M 100 100M    0     0 6625k      0  0:00:15  0:00:15 --:--:-- 9302k
```

Figura 1. Prueba de descarga para el archivo descrito.

```
curl https://nbg1-speed.hetzner.com/100MB.bin --output test.download
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 100M 100 100M    0     0  7113k      0  0:00:14  0:00:14  --:--:-- 9313k
```

Figura 2. Prueba de descarga sin el servidor.

Finalmente, para optimizar el tamaño de buffer utilizado en el servidor, se utilizó un script que iterativamente compilaba y ejecutaba el servidor aumentando el tamaño de buffer, ejecutando en cada paso un curl de descarga de archivo e imprimiendo los tamaños de buffer junto con la velocidad de descarga en Mbps. De esta manera, comparando las velocidades, se pudo encontrar un tamaño de buffer óptimo de 16KB, que sea suficientemente grande para que no genere demasiadas llamadas al sistema, pero no de tamaño excesivo como para que se desperdicie memoria y se afecte la caché del CPU. A continuación se muestra el resultado obtenido en la ejecución de este test.

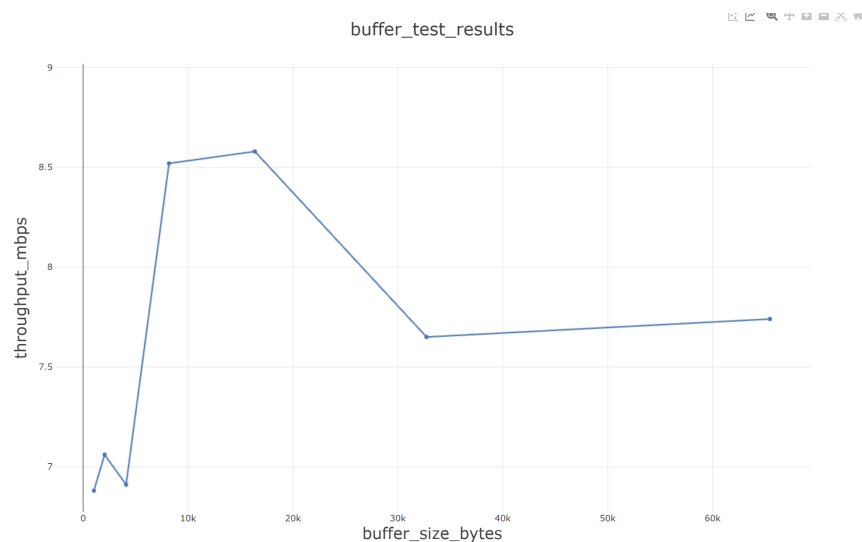


Figura 3. Testeo de Throughput según tamaño de buffer.

11. Ejemplos de Configuración y Monitoreo

A continuación, se muestran ejemplos realizados para el monitoreo y la configuración del servidor desarrollado:

```
leowk@leowk-Legion-S7-16IAH7:~/ITBA/3A-2C/Protos/tpe-protos$ ./monitor_client
Please enter Username:password (in that format)
username:password
OK Welcome username
Type commands and press Enter. Type 'QUIT' to exit.
```

Figura 4. Prueba de logueo mediante un usuario y contraseña.

```
monitor>STATS
OK
all time connections:2
bytes transferred:74312
active users:0
current connections:0
max concurrent connections:1
server uptime:276s
```

Figura 5. Prueba de comando STATS.

```
monitor>USERS
OK No active users
```

Figura 6. Prueba de comando USERS (sin usuarios activos).

```
monitor>CONNECTIONS
OK
current connections:0
max concurrent connections:1
```

Figura 7. Prueba de comando CONNECTIONS.

```
monitor>ACCESS_LOG
OK Access log for all users:
username: www.google.com:80 [2025-07-15 12:40:39] SUCCESS
Anonymous: www.google.com:80 [2025-07-15 12:41:06] SUCCESS
```

Figura 8. Prueba de comando ACCESS_LOG (sin parámetro username).

```
monitor>ACCESS_LOG username
OK Access log for user username:
www.google.com:80 [2025-07-15 12:40:39] SUCCESS
```

Figura 9. Prueba de comando ACCESS_LOG (con parámetro username).

```
monitor>CONFIG timeout 15
OK timeout set to 15
```

Figura 10. Prueba de configuración de timeout.

```
monitor>QUIT
OK Goodbye
Exiting monitor mode.
Disconnected from monitor server
```

Figura 11. Prueba de comando QUIT.

12. Documento de Diseño del Proyecto

El diseño del proyecto se encuentra representado en la siguiente figura, con todos los servicios y aplicaciones desarrolladas y las interacciones entre los mismos.

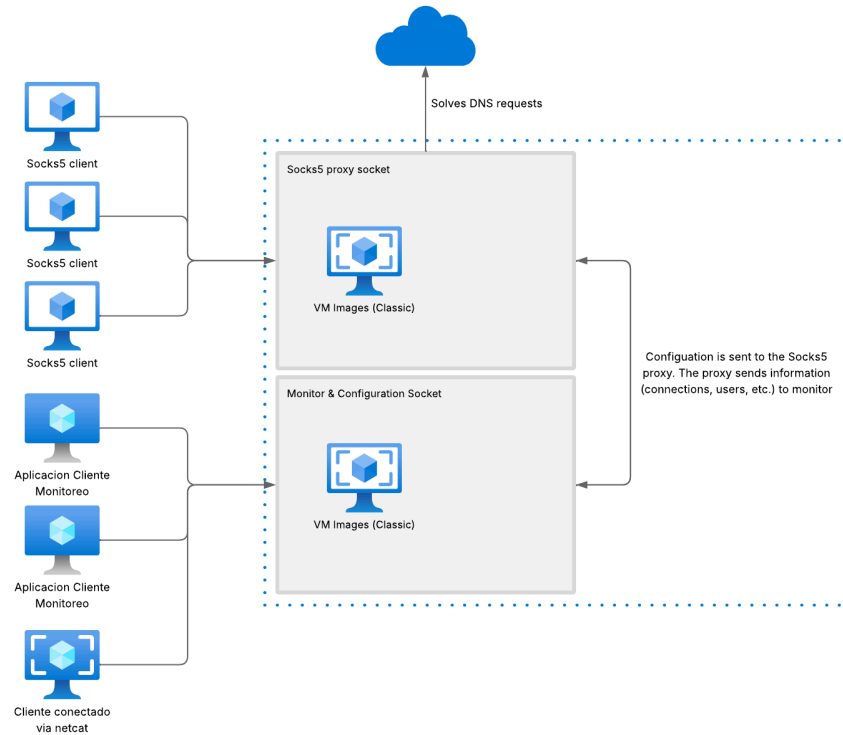


Figura 12. Diagrama de diseño del servidor con todas sus funcionalidades.

13. Bibliografía

1. RFC 1928, <https://datatracker.ietf.org/doc/html/rfc1928>
2. RFC 1929, <https://datatracker.ietf.org/doc/html/rfc1929>