Creación de una red robusta proxing p2p para utilizar libremente servicios en Internet desde redes restrictivas

Lluís Alonso Pichardo

6 de febrero de 2024

Resumen– Muchas veces nos conectamos a redes públicas abiertas, las cuales por razones de seguridad los administradores de red encargados de estas optan por restringirlas, haciendo imposible la utilización de ciertos servicios en Internet. En este proyecto, se diseñará e implementará una red proxy P2P, mostrando todo el proceso de diseño, implementación, y las funciones y roles de los diferentes participantes de la red, además se mostrará el proceso de simulación y se probará sobre una red real.

Palabras clave– Proxy, P2P, túnel, servidor, peers.

Abstract– Many times, we connect to open public networks, which, for security reasons, the network administrators in charge of these choose to restrict them, making it impossible to use certain services on the Internet. In this project, I will design and implement a P2P proxy network, showing the entire design process, implementation, and the functions and roles of the different network participants; the simulation process will also be shown and tested on a real network.

Keywords– Proxy, P2P, tunnel, server, peers

1 Introducción

MUCHOS administradores de red encargados de garantizar la seguridad en una red pública optan por restringir la red y solo permitir conexiones salientes por el puerto 80 y 443, o, por otra parte, tienen una blacklist de IP donde bloquean todo el tráfico saliente a estos.

Esto hace que utilizar determinados servicios desde estas redes sea "imposible", ya que por ejemplo, si se quiere abrir una VPN que utilice un puerto diferente, o usar un servicio de una IP baneada, no podrá efectuarse la conexión.

En el año 2021, la Universidad Autónoma de Barcelona sufrió un ciberataque[1], lo que resultó en la paralización de sus servicios, dejando a todos los estudiantes sin acceso a servidores ni internet dentro de la Universidad. Como respuesta a esta situación, una vez que se restauraron los servicios, se implementó la configuración restrictiva mencionada anteriormente en la red proporcionada a los alumnos, dificultando el acceso a según que servicios en Internet desde la red universitaria.

La motivación de este trabajo surgió por este hecho, y se tratará de implementar una red de proxys P2P que permitiera utilizar libremente todo tipo de servicios desde una red restringida, evadiendo las configuraciones de los administradores de red.

En el capítulo II se muestra el estado del arte y se explicará en que se diferencia el servicio a desarrollar en este proyecto a los existentes.

En el capítulo III se exponen los objetivos a seguir en este trabajo, subdividiendo estos en objetivos más pequeños para facilitar la obtención de estos.

En el capítulo IV se explica la planificación y la metodología, para designar las pautas a seguir, y garantizar el buen desarrollo del proyecto.

En el capítulo V se muestra como se diseñó el sistema, además de una primera implementación de este diseño, detallando los aspectos, e indicando las librerías utilizadas.

En el capítulo VI se demuestra como se asegura el servicio, y qué tecnologías se utilizaron para conseguirlo.

En el capítulo VII se prueba que el servicio es funcio-

• E-mail de contacte: Lluis.AlonsoP@autonoma.cat
• Mención realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Miguel Carpio Miranda (departament d'Enginyeria de la Informació i de les Comunicacions)
• Curs 2023/24

nal, y mediante una prueba de concepto, donde se simula una restricción por un administrador, se consigue igualmente acceder a un servicio previamente inaccesible.

En el capítulo VIII se hace un análisis breve del bypass, y se analizan las diferencias entre una conexión HTTPS normal, y un túnel SSH establecido en el mismo puerto.

En el capítulo IX se muestran las conclusiones del proyecto, comparando los resultados obtenidos con los objetivos iniciales presentados, y valorando que se ha aprendido durante la realización del trabajo.

En el capítulo X se exponen posibles mejoras para este trabajo, dirigidas a futuros proyectos que deseen continuar con esta labor.

2 Estado del arte

La práctica utilizada para hacer el bypass del firewall es comúnmente conocida[2] y utilizada por varias herramientas, como por ejemplo Chisel[3], una herramienta que sirve para hacer túneles en firewalls restrictivos. Aun así, esta herramienta funciona solo end-to-end[4] y, por tanto, no te permite redireccionar puertos.

El tema de tener una red para redistribuir el tráfico tampoco es nada nuevo, se podría parecer a lo que hace una VPN. De hecho, existen diversos software que ofrecen este servicio que se denominan proxys P2P[5], como por ejemplo Smartproxy[6], o Iproyal[7].

Aun así, en este trabajo he querido empezar desde cero una red P2P proxy, y, por tanto, encontrarme con las problemáticas que pueda suponer el diseño e implementación de estas.

3 Objetivos

El objetivo principal de este proyecto consiste en conseguir implementar una red para redirigir el tráfico, mediante el uso de la aplicación proxing P2P. Para lograr este objetivo, se tendrán que cumplir los siguientes objetivos específicos.

3.1. Diseñar e implementar los diferentes participantes en la red

Para el diseño de la red, intentar distribuir lo máximo posible el carácter distribuido de la red. Ya que desde un inicio de partida se optará por una solución centralizada, y progresivamente se irá descentralizando.

En cuanto a la implementación, el enfoque inicial será la simulación o emulación de una red con el propósito de verificar la funcionalidad de la aplicación P2P. Una vez se haya confirmado su eficacia en este entorno simulado, se procederá a evaluar su funcionamiento en una red real.

3.2. Asegurar los aspectos de comportamiento de la red

En este apartado se mostrarán los objetivos a realizar para mantener el buen funcionamiento y comportamiento de la red, garantizando la seguridad, las tolerancias a fallas, y la optimización de recursos.

3.2.1. Asegurar el servicio

Conseguir que la seguridad de la red sea robusta y, por tanto, garantice la confidencialidad, la integridad, la autenticación de los mensajes enviados en esta red, lo que comúnmente se conoce como CIA[8].

3.2.2. Controlar las fallas

Al ser una red P2P, conseguir que esta sea tolerante a fallas, es decir, que sea tolerante a que los diferentes agentes de la red se desconecten de esta. Esto significa que, por ejemplo, si un cliente tiene asignado un proxy y este falla o deja de estar en línea, se tendrá que asignar automáticamente otro proxy sin que el cliente se dé cuenta.

3.2.3. Optimizar los recursos

Adicionalmente, es importante conseguir temporalidad de los agentes en la red. Esto significa que si un cliente tiene un proxy asignado, pero no lo utiliza durante un período determinado, dicho proxy se liberará para que otros clientes que lo necesiten puedan usarlo. El cliente original solo volverá a tener un proxy asignado cuando manifieste nuevamente su intención de utilizar la red con proxy, evitando así la asignación innecesaria de recursos de proxy cuando no son utilizados.

3.3. Garantizar las caracterísitcas en los clientes

En este apartado se mostrarán los objetivos a realizar para brindar y facilitar el uso y la información de los clientes en el servicio, informando en todo momento a los peers, proveerles de una interfaz gráfica, y un script de instalación para facilitar la incorporación de estos.

3.3.1. Informar en todo momento a los peers

Conseguir que todos los peers tengan información sobre su estado en la red, y sepan la información de las acciones que hacen, y también sepan la posibilidad de que alguna excepción haya ocurrido. Para esto, han de tener algún tipo de lugar donde poder ver estas trazas o logs.

3.3.2. Proveer de una interfaz a los clientes

Implementar una interfaz gráfica de usuario (GUI) para que los clientes puedan realizar de manera más sencilla todas sus acciones en la red. A través de esta GUI, los clientes deben poder solicitar la asignación y desasignación de un proxy, además de la posibilidad de ofrecerse y retirarse como proxy.

3.3.3. Crear un script para los peers

Desarrollar un script de instalación destinado a los clientes que deseen unirse a la red, con el propósito de simplificar su acceso a la misma. Esto permitirá lograr el objetivo de proporcionar una capa de abstracción al cliente, haciéndole más sencillo el proceso de incorporación.

3.4. Análisis y comprensión del bypass

Para lograr este objetivo, analizaré el tráfico y trataré de entender las diferencias entre el tráfico web por el puerto 443 y un túnel establecido por este puerto, y, por tanto, comprender la dificultad de un firewall de detectar este bypass.

4 Planificación y metodología

A causa de que en este proyecto se va a desarrollar bastante software, (se ha de desarrollar tanto el código del servidor como de los clientes) y las limitaciones que conlleva un trabajo de fin de grado (recursos y tiempo limitado), se ha pensado que la metodología más adecuada es la de Prototyping.

A primera vista suena extraño utilizar este tipo de metodología en un TFG, donde se suele tener un cliente y a medida que se va desarrollando el proyecto se le pregunta al cliente por feedback. Pero esta manera de desarrollar

software, cuando no se conocen a priori las dificultades que pueden surgir, es bastante útil, y encaja perfectamente en este trabajo.

Por tanto, empezaré desarrollando un prototipo sencillo y viendo qué funcionalidades ir agregando, a medida que vaya creciendo la complejidad me vestiré de "cliente" y dirigiré el proyecto hasta los objetivos especificados anteriormente. Esto también es útil, porque como se quiere llegar a un sistema seguro, es mejor primero centrarse en la funcionalidad, aplicando medidas de seguridad básicas, y en el momento en el que el prototipo funcione, mejorarlas.

Por tanto, el proyecto tendrá diferentes fases, las cuales están representadas mediante el diagrama de Gantt en la Figura 1,2 y 3 del Apéndice 1. Primero se tratará de hacer las pruebas en el entorno de simulación más pertinente para poder probar la funcionalidad del código que se vaya implementando. Luego se hará un prototipo simple de servidor centralizado, clientes y proxys.

Una vez logrado esto, se intentará garantizar la seguridad del sistema, haciendo los cambios pertinentes al prototipo diseñado.

Cuando se demuestre la seguridad de este sistema, entonces se podrá probar en una red real, teniendo que adaptar el código ante las dificultades que esto conlleva. Todo esto se puede ver en la Figura 2 del Apéndice 1.

Finalmente, para lograr los últimos objetivos se intentará distribuir lo máximo posible el comportamiento de la red, intentando eliminar el rol de servidor centralizado. Esta etapa del proyecto es bastante complicada de definir, ya que en un sistema distribuido lograr la máxima distribución es complicado. Esta fase se puede ver en la Figura 3 del Apéndice 1.

Con todo esto, la duración del proyecto estimada es de 160 días.

5 Desarrollo

En este apartado se explica de manera detallada en que ha consistido todo el diseño e implementación de la red, describiendo las funcionalidades de los peers, los errores y las soluciones encontrados durante este proceso.

5.1. Primer diseño de la red

Para facilitar la implementación en este proyecto, se optó por un diseño centralizado, donde un servidor se encargaría de gestionar las asignaciones de otros peers, almacenar el estado de estos, gestionar las claves para los túneles, entre otras funcionalidades que se explican en el apartado 6.2 Los peers en esta primera versión tendrán dos tipos de roles. Podrán ofrecerse como servidores proxy, para otros usuarios de la red que quieran utilizar el servicio de la red, y, por otro lado, pueden ser clientes que quieran utilizar el servicio. Cabe destacar, que en esta primera versión no se va a controlar la malignidad de los peers, se va a suponer que un cliente proxy, no se encuentra en una red restringida, o que una vez se puede hacer un túnel de un cliente a un proxy, este no va a intentar conectarse por SSH para hacer maldades. En una futura versión se resolverán estas problemáticas.

Por tanto, para describir todo el Workflow de la red, en la siguiente Figura se puede ver resumido todo el proceso.

Primeramente, en un caso ideal, cuando al servidor centralizado le llega una petición de un cliente que quiere ofrecerse como proxy, el servidor le ofrece su clave pública para posteriormente probar la conexión y verificar la buena intencionalidad del proxy. Una vez esta comprobación es efectuada, y se valida la disponibilidad del peer, se almacena en una base de datos la dirección IP de este.

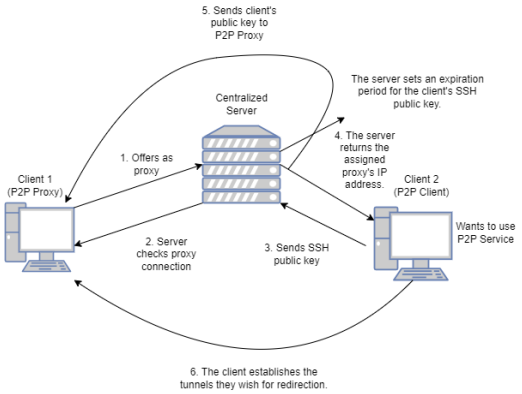

Fig. 1: Workflow del primer diseño

Cuando una petición de un cliente pidiendo un proxy llega, el servidor consulta en la base de datos proxys disponibles, y si hay alguno, le devuelve la IP del proxy para que este pueda realizar la conexión. En este primer diseño la profundidad de los peers a la hora de hacer los túneles es solo de 1, es decir, no hay más de un redireccionamiento en la conexión, para facilitar el trabajo a realizar en esta versión.

En la petición del cliente, este le manda la clave pública, para que el servidor se encargue de enviarla al proxy. Así, cuando el cliente quiera establecer un túnel, se podrá efectuar.

5.2. Implementación del servidor

En el Servidor se optó por NodeJs, más concretamente, se utilizó el framework de Express[9]. Para la base de datos se escogió MongoDB, ya que hay bastante documentación para construir una REST API con Express[10]. A nivel de código, para este servidor, se han definido 2 modelos:

- ProxiesModel.js:
  Este modelo se encarga de gestionar las funcionalidades relacionadas con los proxies en el lado del servidor. Las funciones más importantes y destacables son la de verificar la conexión ssh del proxy, la de almacenar la IP del proxy, la de asignar un proxy libre a un cliente, y la de desasignar un proxy asignado, todo esto actualizando siempre la base de datos de Mongo.
- sshKeyModel.js:
  Este modelo se encarga de gestionar las funcionalidades relacionadas con la gestión de las claves. La función más importante es la de la asignación de una expiración de las claves públicas, y la eliminación de estas en la base de datos cuando expiren.

En referencia a la API Rest esta tiene diversos endpoints, de entre los cuales se encuentran:

- /get-server-public-key
  Esta ruta permite descargar la clave pública del servidor centralizado. Esta se encuentra en la ruta `./.ssh/id_rsa.pub`, y se manda en la respuesta en formato Json, esto gracias a la librería BodyParser[11].

- /subscribe-proxy :
  Esta ruta permite a un peer ofrecerse como proxy, como se ha mencionado antes, es necesario que este ya tenga la clave pública ssh del servidor para validar su conexión.
- /unsubscribe-proxy
  Esta ruta permite que un proxy que está ofreciendo sus servicios pueda de forma voluntaria desofrecer sus servicios.
- /assign-proxy
  Esta ruta permite a un cliente que se le asigne un proxy, para poder posteriormente crear los túneles con este. En la petición el cliente envía su clave pública, y en la respuesta se le devuelve la IP del proxy, en caso de que haya uno disponible.
- /unassign-proxy
  Esta ruta permite a un cliente de forma voluntaria que le desasignen el proxy asignado por el servidor.

5.3. Base de datos

Como se ha comentado anteriormente, la base de datos escogida ha sido MongoDB. Se han definido dos bases de datos. La base de datos proxies, y la base de datos sshkeys.

- proxies: Esta base de datos tiene una única tabla, la cual almacena la IP del proxy, la IP del cliente, y el estado de la asignación. Este último parámetro es para encontrar de manera más eficiente un proxy sin asignar.
- SSHkeys: Esta base de datos almacena también una única tabla, la cual almacena la clave pública de los clientes, además del tiempo de expiración de estas.

5.4. Implementación del cliente

Para el código de los peers, se optó por utilizar el lenguaje C++, utilizando la librería FLTK[12] para la implementación de la GUI. Se decidió utilizar esta solución a causa de que una programación orientada a objetos iba a ayudar mucho en la escalabilidad del código del proyecto, y esta librería es conocida por su fácil integración y sencillez.

Cabe destacar que debido a la gran cantidad de funciones y código que hay en la parte de los clientes, solo se explicará las clases implementadas por encima, y no tan en detalle. En la Figura 2 se puede observar los diferentes elementos del menú principal de la GUI.

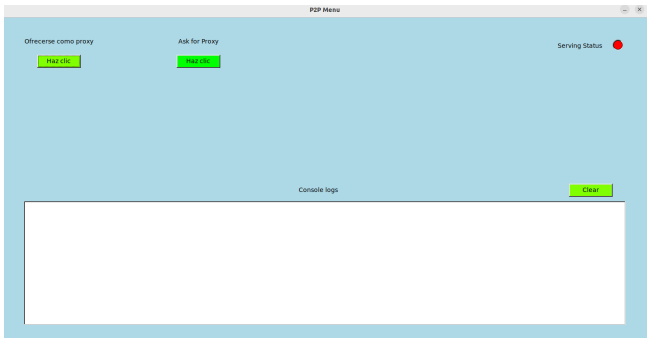

Fig. 2: Menú principal GUI peers

La GUI tiene diversos elementos, los cuales son:

- Botón de ofrecerse como proxy. En el momento que el usuario hace clic, se envía una petición al servidor, en la ruta mencionada anteriormente. En el caso de que se ofrezca correctamente, el botón se reemplaza por un botón de parar de ofrecerse como proxy, para que el peer pueda dejar de estar activo en la red.
- Botón de ask for proxy. Cuando este botón es seleccionado, se llama al endpoint correspondiente en el servidor centralizado, y cuando se le asigna un proxy al peer, el usuario puede decidir que túneles establecer como se puede ver en la Figura 3. Al principio solo hay una sesión para conectarse, pero el usuario puede, mediante los botones de añadir o quitar, modificar el número de túneles posibles simultáneos con un máximo de 4.

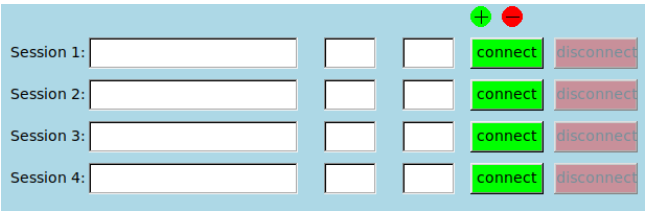

Fig. 3: Sesiones que puede elegir el cliente

  Cada sesión tiene un botón para establecer la conexión del túnel. En un principio se quería utilizar libssh[13] para el establecimiento de los túneles, el problema es que esta librería no permitía acceder a los túneles desde fuera de la aplicación, y, por tanto, se decidió utilizar el sistema del host para ejecutar el comando SSH del túnel.
- Un icono de estado, que representa si el peer está sirviendo o no. En el momento en el que se le da al botón de ofrecerse como proxy, y el servidor le contesta con que se ha suscrito correctamente, el icono aparecerá en verde, en cambio, si no está sirviendo, el icono aparece en rojo.
- Una consola de debugación, para informar de todos los mensajes recibidos en la red, además de posibles errores no esperados, para que el usuario sepa en todo momento que está sucediendo. Además de esto, existe un botón de crear, para limpiar la consola de mensajes ya leídos.

A nivel de código, en los peers se han definido 4 cpp:

- P2PMenu.cpp: Esta clase gestiona todas las funcionalidades de la interfaz gráfica. Esta contiene objetos del resto de las clases, ya que es la clase principal.
- ConnectionManager.cpp : Esta clase se encarga de las peticiones web, y de las conexiones SSH. Para las peticiones web se ha utilizado la librería libcurl[14].
- RoundButton.cpp : Esta clase gestiona los botones más complejos que se muestran en la GUI.
- main.cpp: Esta clase es la más sencilla de todas, y solo llama al constructor de la clase P2PMenu, la encargada de la interfaz.

5.5. Script de instalación para los peers

Para facilitar la creación de peers a la hora de la simulación, se ha diseñado un script de instalación, que configura los peers para el uso en la red. Este consiste en:

- Instalación de librerías mencionadas anteriormente, a causa de que las librerías están dynamically linked[15]. Esto se ha decidido así para minimizar el peso del ejecutable.
- Instalación del servidor SSH, configuración de escucha en el puerto 443 y autenticación por clave, modificando el archivo `/etc/ssh/sshd_config`, y lanzamiento del servidor.
- Creación del usuario para los túneles, y creación de directorios y archivos necesarios, como el directorio `.ssh` y el archivo `authorized_keys`, además de establecer los permisos y evitar errores en la conexión.

6 Asegurar el servicio

Antes de mostrar el funcionamiento del servicio, era necesario proceder a mejorar la seguridad del servicio, para poder evitar ciertos ataques. En este apartado se mostrará como se ha migrado las conexiones HTTP a HTTPS tanto en el lado del servidor como del cliente. Gracias a esto se evitan ataques del tipo Man-In-The-Middle[16].

6.1. En el servidor

Para el servidor centralizado, se ha utilizado el módulo https[17], y la generación de un certificado autofirmado para el dominio de este. Se ha escogido el dominio ”p2pProxyService.com”. La problemática de tener un dominio radicaba en tener que pagar por este en un principio, pero se optó por una solución práctica y segura. En el repositorio de GitHub mencionado anteriormente, existe la IP del servidor y el dominio de este, para que los clientes al hacer la instalación, descargarán la entrada del archivo, y agregarán esta al directorio `/etc/hosts`, además del contenido del certificado público. La modificación en el código consistió en modificar el propio server.js, y en vez de usar app.listen directamente, se invoca a https.createServer, pasándole la clave privada y pública.

También se tuvo que actualizar la actuación del servidor como cliente cuando se conecta al puerto 12345 explicado anteriormente. Para esto se tuvo que utilizar la librería tls[18], y la implementación de este fue más complicada. Esto se hizo modificando ligeramente la ruta /subscribe-proxy, obligando al proxy proporcionar la clave pública de la conexión de escucha de este. Con esta modificación se comprueba que el proxy es válido intentando la conexión SSH y la del socket en escucha. Si una de estas falla, el servidor no añadirá al proxy como válido. Por tanto, se tuvo que modificar la base de datos, ahora además de todo lo anterior, almacenando la clave pública del proxy.

6.2. En el cliente

Para asegurar las conexiones del cliente, primero se comprueba en el código que existan las claves pública y privada de este, y en caso de no tenerlas, generarlas. El directorio de estas claves se decidió en `/home/ClientP2P/utils/`, separando las claves del peer y del servidor por los subdirectorios client y server respectivamente.

Para las conexiones salientes hacia el servidor centralizado se utilizó la librería que se utilizaba anteriormente para conexiones HTT, la librería libcurl, modificando la clase ConnectionManager, específicamente la función postRequest, y además de esto, se procedió a enviar al servidor en la petición de /subscribe-proxy, la clave pública del proxy.

Para migrar de HTTP a HTTPS el socket del puerto 12345,se utilizaron las librerías atomic[19] y openssl[20], y se modificó la función listenForConnections, utilizando las dos librerías anteriores.

Se tuvo que modificar también el Makefile, añadiendo las opciones -lcrypto, -pthread y -lssh, para que el compilado se generara.

6.3. Interacciones entre clientes y servidor

Existía un problema importante que no se tuvo en cuenta cuando se implementó el código del servidor. Este consistía en que la autenticación de los clientes se hacían mediante la IP de la petición. En un entorno local esto no supone un problema, pero sí será un problema cuando se lance el servicio en Internet, a causa de que por culpa de NAT (Network Adress Translation) múltiples clientes que accedieran al servicio podrían interferir entre ellos.

Una de las opciones que se tuvo en cuenta, era que los clientes se registrarán en el servidor con un usuario y contraseña, pero me parecía perder la oportunidad de identificar a los clientes por su clave pública.

Otra opción que se pensó, fue que los clientes utilizarán su clave pública cada vez que hicieran peticiones, pero en frente a esta posibilidad, se prefirió utilizar JWT[21] para identificar de manera equivoca a los clientes. Con esta modificación, los clientes tendrán dos tipos de JWT, uno para cuando actúan de proxys, y otro para cuando actúan de clientes, para evitar que un cliente que esté actuando como proxy pueda actuar como cliente con su JWT.

Para la implementación del JWT, se tuvo que modificar la base de datos, para poder almacenar los tokens emitidos por el servidor, el servidor en sí, emitiendo los tokens en los endpoints `/subscribe-proxy` y `assign-proxy`, para más tarde verificar los proporcionados de los clientes en las rutas `/unsubscibe-proxy` y `/unassign-proxy`.

Otra mejora que se tuvo que hacer consistió en la autenticación por parte del servidor al enviar los mensajes a los proxys, ya que en la versión anterior no había ningún tipo de autenticación, y cualquiera podía suplantar a este.

Para solucionar esta problemática se decidió que el proxy antes de ofrecer sus servicios generará un número entero random de 256 bits, y que se lo enviará al servidor. Más tarde, cuando el servidor tenía que enviar un mensaje al cliente, este le devolvería él random de 256 bits para verificar su autenticación. De esta manera se evita que los mensajes de control que envía el servidor, los pueda suplantar un atacante.

7 Prueba de concepto

En este apartado se mostrará un caso de uso de este servicio, demostrando de que funciona en un entorno local, para más tarde poder implementarlo en la red. Para hacer esta prueba, se lanzan dos máquinas virtuales en modo bridge[22], los dos con imágenes de Ubuntu 22.04.3 LTS[23].

En la Figura 4, se puede observar como no hay ningún

problema al abrir el archivo VPN:

```
client
dev tun
proto tcp
remote edge-eu-starting-point-1-dhcp.hackthebox.eu 443
resolv-retry infinite
nobind
persist-key
persist-tun
remote-cert-tls server
comp-lzo
verb 3
data-ciphers-fallback AES-128-CBC
data-ciphers AES-256-CBC:AES-256-CFB:AES-256-CFB1:AES-256-CFB8:AES-256-OFB:AES-256-GCM
tls-cipher "DEFAULT:@SECLEVEL=0"
auth SHA256
key-direction 1
```

Fig. 4: Archivo de conifguración VPN

Si se abre el archivo de configuración no hay ningún problema, pero si se ejecuta el comando de la Figura 5 no se puede acceder a la VPN, simulando una especie de blacklist por un administrador de red.

```
lluis@lluis-VirtualBox:~/Documents/POC$ sudo iptables -A OUTPUT -d edge-eu-starting-poi
lluis@lluis-VirtualBox:~/Documents/POC$ sudo openvpn ./BACKUP.ovpn
2024-01-21 20:25:04 WARNING: Compression for receiving enabled. Compression has been us
ed unless "allow-compression yes" is also set.
2024-01-21 20:25:04 OpenVPN 2.5.5 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOL
2024-01-21 20:25:04 library versions: OpenSSL 3.0.2 15 Mar 2022, LZO 2.10
2024-01-21 20:25:04 Outgoing Control Channel Encryption: Cipher 'AES-256-CTR' initializ
2024-01-21 20:25:04 Outgoing Control Channel Encryption: Using 256 bit message hash 'SH
2024-01-21 20:25:04 Incoming Control Channel Encryption: Cipher 'AES-256-CTR' initializ
2024-01-21 20:25:04 Incoming Control Channel Encryption: Using 256 bit message hash 'SH
2024-01-21 20:25:04 TCP/UDP: Preserving recently used remote address: [AF_INET]23.19.62
2024-01-21 20:25:04 Socket Buffers: R=[131072->131072] S=[16384->16384]
2024-01-21 20:25:04 Attempting to establish TCP connection with [AF_INET]23.19.62.79:44
```

Fig. 5: Bloqueo de accesso a la VPN

Entonces, se procede a lanzar el servidor centralizado conectado a la base de datos de Mongodb, como se puede observar en la Figura 6.

```
lluis@lluis-VirtualBox:~/Documents/WebApp$ node server.js
Conexión a MongoDB establecida.
Servidor Express en ejecución en el puerto 3000
lluis@lluis-VirtualBox: ~/Documents/client
lluis@lluis-VirtualBox:~/Documents/client$ sudo netstat -tlnp
[sudo] password for lluis:
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Pro
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      16660/s
tcp        0      0 127.0.0.1:27017         0.0.0.0:*               LISTEN      1002/mo
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      32755/c
tcp        0      0 0.0.0.53:53             0.0.0.0:*               LISTEN      662/sys
tcp6       0      0 :::443                  :::*                    LISTEN      16660/s
```

Fig. 6: Lanzamiento del servidor centralizado

Ahora que ya se tiene el servidor lanzado, desde la otra máquina virtual en modo bridge, después de proceder con la instalación, se ejecuta la aplicación, y se ofrece como proxy, obteniendo el mensaje de que se ha suscrito por consola, como se puede observar en la Figura 7. El servidor centralizado se encarga de comprobar si es posible conectarse al proxy, y como la prueba es efectiva, en la base de datos se agrega la información de este proxy, como se puede observar en la Figura 8. Este paso se correspondería al número 1 y 2 del Workflow de la Figura 1.

Ahora se procede a pedir un proxy desde la máquina

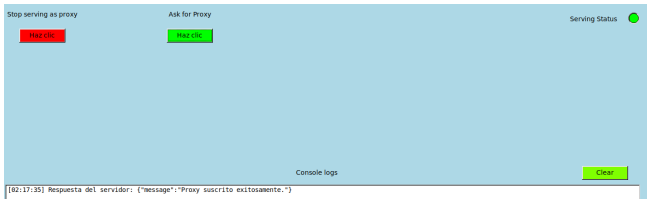

Fig. 7: Proxy se ha ofrecido correctamente

```
> db.proxies.find()
{ "_id" : ObjectId("65ad5d6a7118217f2a9b4676"), "ip" : "192.168.137.240", "certificateP
oZIhvcNAQEL\nBQAwADAeFw0yMzEyMTUyMjQ3MjVaFw0yNDEyMTQyMjQ3MjVaMAAwggIiMA0GCSqG\nSIb3DQEB
642FWR1KvlOJASHoKxLeDsDynhWKEOfdt\n9JaqulWn2QB1mNs7SBdTXJRJoHEZTyPkKu0zUlkMCdn24SGlpy8j
sntdw8JDaZcassP4P9hqI1GOep2qCL2ns81vimiBeqmvh1FkRCBEml\n6eqYoW4UhE3F8us1MI3zFa1blcRbfL
6bL3jNuc21F\nn4fH4mbe7vslkqtgWG1HuzW2nLcRHzP75s9Q0+G7YdL6aJLS8Xwz4eYlJnYit7O+\niOGxtdWk
uBkxn4fvQNH3WKwqt4w1KzyQ9hew60wUf\nbEJHzrKDpPqhayuL8lj1sqbi3trp4hG9+h0zmhApD8JrU6+6iYpE
BgwFoAUy7nsgGBbKKwUJaqb0ieUTOsjujwwDwYDVR0TAQH/BAUwAwEB\n/zANBgkqhkiG9w0BAQsFAAOCAgEA06
NTzzRYmcdXT\nj9U9vz5UJeSOjRlwjZgR5JlIm7O1xTUQm7WebHEIhsWzsxsTxK8MyLMTD46d3E5K\nLtTCSyhZ
EHjuhUUNtDWRIQSNNSrkTsnA5dO/rSELK\nxLhBj3MernE+f+2Z22SEm1B2SJcWacdcjfT9OtboOqKC5egX6ngG
85oOxEXAnNXm8DQNnIbvo8k7K7faIi8f3iril+YyYJu1dfArcwQ3EOE\ndHRaZOzOgekQQvWYpdxTAnmR97jtgx
zbqptWvMt8a\nLcNduM+QhXBDSuY74o/clwmBfj7SF6mwse0A6ntEC5Y7CfA1I7SmZJJSKJqdxQjH\nB4THs28=
61f12d41ca7d9ee737b2", "clientip" : "", "assigned" : false, "__v" : 0 }
>
```

Fig. 8: Información actualizada en la base de datos

restringida, ofreciendo una clave pública. Como en este momento hay un proxy sin asignar, se le asigna este, y se procede a poner el puerto e IP de la configuración de la VPN, además de un puerto local al cual haremos el túnel, como se puede observar en la Figura 9. Esto correspondería con el paso 3 y 4 del Workflow de la Figura 1.

Antes de que el cliente pueda establecer un túnel, el servidor ha enviado la clave pública del cliente al proxy, para que este la almacene. Una vez el cliente le da al botón de conectar, se establece el túnel como se puede ver en la Figura 10. Este paso corresponde al número 5 y 6 del Workflow de la Figura 1.

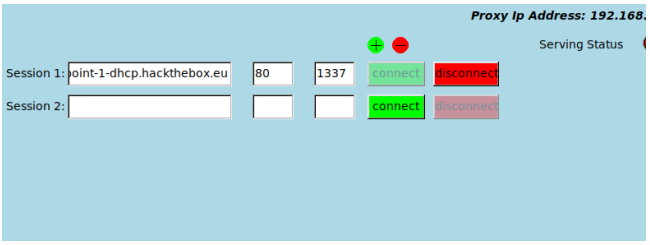

Fig. 9: Túnel establecido desde la aplicación

```
lluis@lluis-VirtualBox:~/Documents/POC$ netstat -tlnp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Pro
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:27017         0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:1337          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp6       0      0 :::443                  :::*                    LISTEN      -
tcp6       0      0 :::3000                 :::*                    LISTEN      31894/n
tcp6       0      0 ::1:1337                :::*                    LISTEN      -
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
lluis@lluis-VirtualBox:~/Documents/POC$
```

Fig. 10: Comprovación del túnel

Ahora, cambiando el archivo de configuración de la Figura 4 por el de la Figura 11, en vez de apuntar directamente al servicio, se modifica edge-eu-starting-point-1-dhcp.hackthebox.eu[24] por localhost o 127.0.0.1, y cambiando el puerto 443 por el 1337, se consigue abrir la VPN desde la máquina restringida, como se puede ver el resultado en la Figura 12.

Por tanto, como se ha podido observar, el servicio funciona correctamente en un entorno local.

Tal y como está el código estructurado, para ser lanzado en Internet, los usuarios de los peers, tendrían que gestionar NAT en sus routers para utilizar el servicio.

8 Análisis del bypass

En este apartado se muestra analizando los datagramas de la posibilidad de intentar detectar si alguien está estable-

```
lluis@lluis-VirtualBox:~/Documents/POC$ sudo cat vpnfile.ovpn
[sudo] password for lluis:
client
dev tun
proto tcp
remote 127.0.0.1 1337
resolv-retry infinite
nobind
persist-key
persist-tun
remote-cert-tls server
comp-lzo
verb 3
data-ciphers-fallback AES-128-CBC
data-ciphers AES-256-CBC:AES-256-CFB:AES-256-CFB1:AES-256-CFB8:AES-256-OFB:AES-256-GCM
tls-cipher "DEFAULT:@SECLEVEL=0"
```

Fig. 11: Modificación de parámetros remote en el archivo VPN

```
lluis@lluis-VirtualBox:~/Documents/POC$ sudo openvpn vpnfile.ovpn
[sudo] password for lluis:
2024-01-21 21:01:37 WARNING: Compression for receiving enabled. Compression has been us
low-compression yes" is also set.
2024-01-21 21:01:37 OpenVPN 2.5.5 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOL
2024-01-21 21:01:37 library versions: OpenSSL 3.0.2 15 Mar 2022, LZO 2.10
2024-01-21 21:01:37 Outgoing Control Channel Encryption: Cipher 'AES-256-CTR' initializ
2024-01-21 21:01:37 Outgoing Control Channel Encryption: Using 256 bit message hash 'SH
2024-01-21 21:01:37 Incoming Control Channel Encryption: Cipher 'AES-256-CTR' initializ
2024-01-21 21:01:37 Incoming Control Channel Encryption: Using 256 bit message hash 'SH
2024-01-21 21:01:37 TCP/UDP: Preserving recently used remote address: [AF_INET]127.0.0.
2024-01-21 21:01:37 Socket Buffers: R=[131072->131072] S=[16384->16384]
2024-01-21 21:01:37 Attempting to establish TCP connection with [AF_INET]127.0.0.1:1337
2024-01-21 21:01:37 TCP connection established with [AF_INET]127.0.0.1:1337
2024-01-21 21:01:37 TCP_CLIENT link local: (not bound)
2024-01-21 21:01:37 TCP_CLIENT link remote: [AF_INET]127.0.0.1:1337
```

Fig. 12: Conexión VPN establecida a través del bypass

```
5: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOW
    link/none
    inet 10.10.16.94/23 scope global tun0
       valid_lft forever preferred_lft forever
    inet6 dead:beef:4::105c/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::49cd:8916:9e48:a89c/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
lluis@lluis-VirtualBox:~/Documents/POC$
```

Fig. 13: Interfaz de la VPN levantada correctamente

ciendo un túnel SSH en un puerto diferente al 22.

En la Figura 14, se muestra la captura de una petición web a Google, pudiéndose observar el uso de TLS para la conexión HTTPS. En comparación a la Figura 15 se muestra la captura del túnel establecido mediante SSH.

La principal diferencia que se ve en las dos capturas de Wireshark es que detecta la conexión HTTPS como TLSv1.2, mientras que la otra la detecta como SSL. Al estar cifradas las dos conexiones son totalmente privadas, pero a priori parece ser posible diseñar un firewall que detecte los túneles.

26	2.122636352	127.0.0.53	127.0.0.1	DNS	103	Standard query
27	2.122693921	127.0.0.53	127.0.0.1	DNS	115	Standard query
28	2.123016336	192.168.137.160	142.250.200.132	TCP	76	41462 → 80 [SY
29	2.139489109	142.250.200.132	192.168.137.160	TCP	76	80 → 41462 [SY
30	2.139525782	192.168.137.160	142.250.200.132	TCP	68	41462 → 80 [AC
31	2.139711623	192.168.137.160	142.250.200.132	HTTP	414	GET / HTTP/1.1
32	2.159937309	142.250.200.132	192.168.137.160	TCP	68	80 → 41462 [AC
33	2.393269937	142.250.200.132	192.168.137.160	HTTP	1165	HTTP/1.1 302 F
34	2.393336610	192.168.137.160	142.250.200.132	TCP	68	41462 → 80 [AC
35	2.397828674	192.168.137.160	142.250.200.132	TLSv1.2	401	Application Da
36	2.413798600	142.250.200.132	192.168.137.160	TCP	68	443 → 60494 [A
37	2.553769500	142.250.200.132	192.168.137.160	TLSv1.2	1106	Application Da
38	2.554859061	142.250.200.132	192.168.137.160	TLSv1.2	1468	Application Da
39	2.554891903	192.168.137.160	142.250.200.132	TCP	68	60494 → 443 [A
40	2.554859486	142.250.200.132	192.168.137.160	TLSv1.2	1468	Application Da
41	2.554859553	142.250.200.132	192.168.137.160	TLSv1.2	2180	Application Da

Fig. 14: Conexión HTTPS a Google

9 Conclusiones

Como conclusiones a los objetivos mencionados anteriormente, en general se han cumplido todos los objetivos. El más importante y principal era el de implementar el servicio que permitiera el acceso libre en Internet desde una red restrictiva, el cual se ha demostrado a pequeña escala mediante la prueba de concepto mostrada en el apartado 7. Se ha conseguido implementar una interfaz para los clientes, como se ha mostrado también en las imágenes. También se ha programado un script para la instalación del software necesario para actuar como cliente. Y finalmente, se aseguró el servicio antes de probarlo en Internet, para evitar ataques, como se muestra en el apartado 6. El único objetivo que aún no se ha logrado completamente es alcanzar una distribución más amplia del servicio. En la implementación actual, se ha dependido en gran medida de la participación de un servidor centralizado para abordar los desafíos asociados con un servicio completamente distribuido

1	0.000000000	192.168.137.160	192.168.137.240	TCP	76	51432 → 443 [S
2	0.000565948	192.168.137.240	192.168.137.160	TCP	76	443 → 51432 [S
3	0.000599255	192.168.137.160	192.168.137.240	TCP	68	51432 → 443 [A
4	0.000790479	192.168.137.160	192.168.137.240	TCP	109	51432 → 443 [P
5	0.001162160	192.168.137.240	192.168.137.160	TCP	68	443 → 51432 [A
6	0.010303383	192.168.137.240	192.168.137.160	TCP	109	443 → 51432 [P
7	0.010329296	192.168.137.160	192.168.137.240	TCP	68	51432 → 443 [A
8	0.010641504	192.168.137.160	192.168.137.240	SSL	1604	Continuation D
9	0.011280204	192.168.137.240	192.168.137.160	SSL	1180	Continuation D
10	0.012477592	192.168.137.160	192.168.137.240	SSL	116	Continuation D
11	0.016734383	192.168.137.240	192.168.137.160	SSL	592	Continuation D
12	0.020732301	192.168.137.160	192.168.137.240	SSL	84	Continuation D
13	0.061982365	192.168.137.240	192.168.137.160	TCP	68	443 → 51432 [A
14	0.062005811	192.168.137.160	192.168.137.240	SSL	112	Continuation D
15	0.062357164	192.168.137.240	192.168.137.160	TCP	68	443 → 51432 [A
16	0.062357216	192.168.137.240	192.168.137.160	SSL	112	Continuation D

Fig. 15: Conexión del túnel realizado mediante SSH

Por tanto, se puede decir que se han cumplido la mayoría de objetivos siguiendo el planteamiento y metodología que se definió al principio del proyecto.

Como conclusiones más generales de este trabajo, he podido aprender las dificultades que suponen diseñar un sistema distribuido, en este caso peer to peer, y sobre todo la dificultad de implementar a nivel de código el diseño. En toda la carrera hemos estudiado tecnologías distribuidas, donde solo veíamos el enfoque teórico o de diseño, sin entrar en la gran dificultad que tiene desarrollar el código de estos, y si no fuera complicado implementar el código de un sistema de estas características, la dificultad que conlleva crear un sistema que sea seguro ante ataques. Aunque el sistema implementado en este trabajo no sea un producto final que se pueda utilizar, me he tenido que ver con todo este tipo de problemáticas en pequeña escala.

Además de que este trabajo me ha servido para desarrollar mis capacidades de full stack, al estar en todo momento programando tanto para el servidor como para el front-end del cliente, potenciando mi conocimiento en lenguajes como JavaScript y C++.

10 Posibles mejores

10.1. Mayor distribución

Intentar conseguir el carácter distribuido más alto posible en la red implementada. Por ejemplo, intentar dividir la carga del servidor centralizado, en diversos servidores, o también se podría migrar la base de datos mongo a sqlite, y, por tanto, tener el archivo de base de datos compartido y distribuido por P2P.

También se podría intentar, mediante una regla de consenso similar a Etherium, que hubiera unos verificadores que se encargaran de monitorear el comportamiento de los proxys.

10.2. Diseño de un firewall

Ver y comprobar si existe algún tipo de firewall que consiga vetar esta técnica, sin comprometer el tráfico web, es decir, desarrollar un firewall que pueda bloquear túneles SSH en puertos diferentes al 22, si es que se puede evitar esta técnica.

AGRADECIMIENTOS

Para finalizar este proyecto, deseo expresar mi más sincero agradecimiento a diversas personas que han brindado un valioso respaldo a lo largo de su desarrollo.

En primer lugar, agradezco a Miguel Carpio por su continuo seguimiento del proyecto durante estos meses, brindándome orientación y sugerencias valiosas para mejorar cada detalle del informe final.

Además, quiero reconocer el apoyo moral brindado por mi familia y amigos.

El constante apoyo y la compañía de mi familia y amigos han sido fuentes invaluables de motivación a lo largo de este viaje, desempeñando un papel significativo en el éxito de este proyecto. Su respaldo emocional ha sido tan valioso como cualquier guía técnica, y estoy profundamente agradecido por contar con ellos a mi lado.

REFERENCIAS

[1] Pñerez, E. (2021) El ciberataque a la uab afectará hasta finales de Año: La Difícil Gestión de una Universidad sin acceso a su sistema informático durante meses, Xataka. Available at: https://www.xataka.com/seguridad/ciberataque-a-uabafectara-finales-ano-dificil-gestion-universidad-accesoa-su-sistema-informatico-durante-meses (Accessed: 19September 2023).

[2] Nek, D. (2022) What is a P2P proxy? things to know: Web hosting geeks' blog, Web Hosting Geeks Blog. Available at: https://webhostinggeeks.com/blog/what-isp2p-proxy/ (Accessed: 19 September 2023).

[3] Jpillora, JPILLORA/Chisel: A fast TCP/UDP tunnel over HTTP, GitHub. Available at: https://github.com/jpillora/chisel (Accessed: 23 September 2023).

[4] academy, A.A. (2020) Incident response case: From SSH tunnel to endpoint analysis, Medium. Available at: https://alparslanakyildiz.medium.com/incidentresponse-case-from-ssh-tunnel-to-endpoint-analysisa4a7c9d0b67d (Accessed: 19 September 2023).

[5] Sharma,R.(2023).7 best P2P proxy providers 2023 Available at: https://www.bloggersideas.com/es/bestp2p-proxy/ (Accessed: 23 September 2023).

[6] The fastest residential proxies, Smartproxy. Available at: https://smartproxy.com/lp/fastestresidential-proxies (Accessed: 23 September 2023).

[7] IPRoyal: Premium quality proxies, unbeatable prices, IPRoyal.com. Available at: https://iproyal.com/ (Accessed: 23 September 2023)..

[8] Hashemi-Pour, C. and Chai, W. (2023) What is the CIA triad?: Definition from TechTarget, WhatIs. Available at: https://www.techtarget.com/whatis/definition/Confidentiality-integrity-and-availability-CIA (Accessed: 30 September 2023).

[9] Infraestructura de Aplicaciones Web Node.js Express. Available at: https://expressjs.com/es/ (Accessed:30 September 2023).

[10] Express.js and Mongodb Rest Api Tutorial MongoDB. Available at: https://www.mongodb.com/languages/expressmongodb-rest-api-tutorial (Accessed: 1 October 2023).

[11] Body-parser, npm. Available at: https://www.npmjs.com/package/body-parser (Accessed:5 October 2023).

[12] Fast light toolkit, Fast Light Toolkit - Fast Light Toolkit (FLTK). Available at: https://www.fltk.org/ (Accessed: 13 October 2023).

[13] The SSH library!, libssh. Available at: https://www.libssh.org/ (Accessed: 25 October 2023).

[14] The LIBCURL API libcurl - API. Available at: https://curl.se/libcurl/c/ (Accessed: 28 October 2023).

[15] Yan, D. (2020) C++ development tutorial 4: Static and dynamic libraries, Medium. Available at: https://domiyanyue.medium.com/c-developmenttutorial-4-static-and-dynamic-libraries-7b537656163e (Accessed: 5 November 2023).

[16] team, I. editorial (2023) Man-in-the-middle attack, IONOS Digital Guide. Available at: https://www.ionos.com/digitalguide/server/security/man-in-the-middle-attack-an-overview-of-attack-patterns/ (Accessed: 16 December 2023).

[17] Node.js V21.4.0 documentation HTTPS — Node.js v21.4.0 Documentation. Available at: https://nodejs.org/api/https.html (Accessed: 16 December 2023).

[18] Node.js V21.4.0 documentation TLS (SSL) — Node.js v21.4.0 Documentation. Available at: https://nodejs.org/api/tls.html (Accessed: 16 December 2023).

[19] Std::Atomic, cppreference.com. Available at: https://en.cppreference.com/w/cpp/atomic/atomic (Accessed: 17 December 2023).

[20] OpenSSL Foundation Inc. OpenSSL. Available at: https://www.openssl.org/ (Accessed: 17 December 2023).

[21] auth0.com Jwt.io, JSON Web Tokens. Available at: https://jwt.io/ (Accessed: 20 January 2024).

[22] Chapter 6, Virtual Networking. Available at: https://www.virtualbox.org/manual/ch06.html (Accessed: 17 December 2023).

[23] Canonical, Ubuntu 22.04.3 Lts (Jammy Jellyfish). Available at: https://releases.ubuntu.com/jammy/ (Accessed: 17 December 2023).

[24] Hacking training for the best, Hack The Box. Available at: https://www.hackthebox.com/ (Accessed: 17 December 2023).

APÉNDICE

A.1 DIAGRAMA DE GANTT

DIAGRAMA DE GANTT

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
1		**Pruebas entorno de emulación, creación de red**	**7 días?**	**mar 19/09/23**	**mié 27/09/23**		
2		Prubas en CORE Network	2 días	mar 19/09/23	mié 20/09/23		
3		Pruebas GNS3	3 días	mié 20/09/23	vie 22/09/23		
4		Pruebas VirtualBox (Bridge)	2 días	vie 22/09/23	lun 25/09/23		
5		**Creación primera versión servidor centralizado**	**10 días**	**mié 27/09/23**	**mar 10/10/23**	**1**	
6		Creación APIs	8 días	mié 27/09/23	vie 06/10/23		
7		Creación de Base de Datos, y implementación en el servidor	2 días	vie 06/10/23	lun 09/10/23		
8		Gestión de claves	2 días	lun 09/10/23	mar 10/10/23		
9		**Creación primera versión clientes**	**21 días?**	**mar 10/10/23**	**mar 07/11/23**	**5**	
10		Implementación GUI	4 días	mar 10/10/23	vie 13/10/23		
11		**Implementación peer proxy**	**8 días**	**vie 13/10/23**	**mar 24/10/23**	**10**	
12		Implementación ofrecerse de proxy	4 días	vie 13/10/23	mié 18/10/23		
13		Implementación dejar de ofrecerse como proxy	4 días	mié 18/10/23	lun 23/10/23		
14		**Implementación peer cliente**	**9 días**	**mar 24/10/23**	**vie 03/11/23**	**11**	
15		Implementación asignación de proxy	5 días	mar 24/10/23	lun 30/10/23		
16		Implementación desasignación de proxy	4 días	lun 30/10/23	jue 02/11/23		
17		Modificación API servidor	10 días	mié 25/10/23	mar 07/11/23		

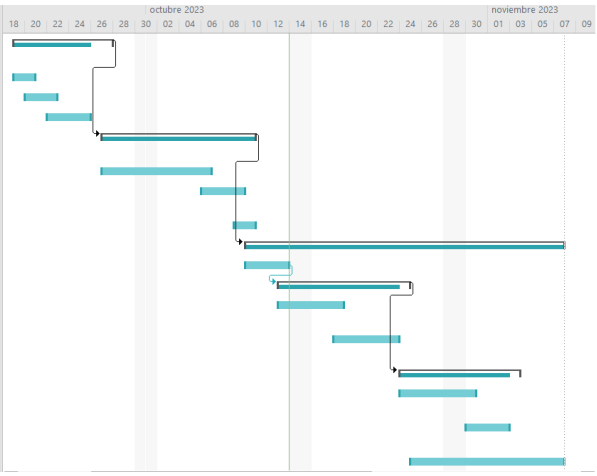

Fig. 1: Diagrama de Gantt Septiembre 2023 - Noviembre 2023

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
18		**Implementación de Seguridad del Sistema**	**19 días**	**vie 03/11/23**	**mié 29/11/23**	**14**	
19		Actualización código servidor	10 días	vie 03/11/23	jue 16/11/23		
20		Actualización código peers	10 días	jue 16/11/23	mié 29/11/23		
21		**Implementación en redes reales**	**29 días**	**vie 01/12/23**	**mié 10/01/24**		
22		Investigación Servicios a través de NAT	5 días	vie 01/12/23	jue 07/12/23		
23		Modificación código servidor	10 días	jue 07/12/23	mié 20/12/23		
24		Modificación código peers	10 días	mié 20/12/23	mar 02/01/24		
25		Uso de la red sobre la red real	7 días	mar 02/01/24	mié 10/01/24		

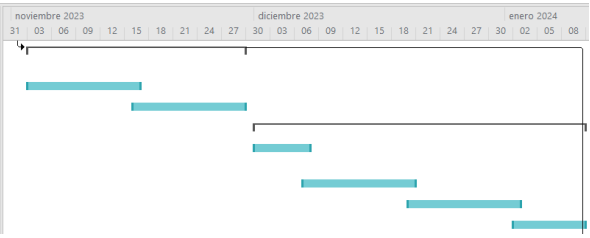

Fig. 2: Diagrama de Gantt Noviembre 2023 - Enero 2024

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
26		**Lograr máxima distribución**	**33 días**	**jue 11/01/24**	**lun 26/02/24**	**18**	
27		Investigar metodos de indexación	10 días	jue 11/01/24	mié 24/01/24		
28		Distribuir la base de datos	10 días	mié 24/01/24	mar 06/02/24		
29		Cambiar codigo peers	15 días	mar 06/02/24	lun 26/02/24		

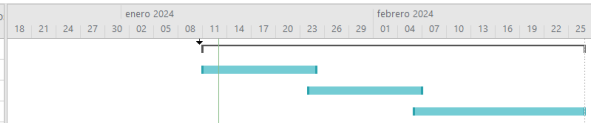

Fig. 3: Diagrama de Gantt Enero 2023 - Febrero 2024