UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Estudio estadístico sobre los tipos de NAT

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Pujol Villanueva, Jaime

Tutor/a: Oliver Gil, José Salvador

CURSO ACADÉMICO: 2022/2023

Resum

Gran part dels dispositius dels usuaris es troben hui dia darrere d'un encaminador amb NAT (tant en les xarxes a casa, com en les connexions 4G/5G dels telèfons, etc). Per tant, NAT és una temàtica interessant amb una gran implicació en moltes aplicacions, des de jocs en xarxa fins a l'ús de xarxes P2P. Aquest projecte té com a objectiu identificar i avaluar els diferents tipus de NAT existents en la xarxa. Per a això s'utilitzarà un programa creat amb l'objectiu de manar i rebre paquets que posteriorment seran analitzats. Una vegada preparat el banc de proves, es faria proves amb diversos encaminador casolans (NAT) i amb diverses xarxes mòbils. Amb això, es busca obtindre uns resultats estadístics que mostren les diferències en els tipus de NAT i quina presència té cadascun d'ells en l'actualitat. Per a finalitzar, es donaran unes conclusions del treball.

Paraules clau: NAT, P2P, resultats estadistics

Resumen

Gran parte de los dispositivos de los usuarios se encuentran hoy en día detrás de un router con NAT (tanto en las redes en casa, como en las conexiones 4G/5G de los teléfonos, etc.). Por tanto, NAT es una temática interesante con una gran implicación en muchas aplicaciones, desde juegos en red hasta el uso de redes P2P. Este proyecto tiene como objetivo identificar y evaluar los diferentes tipos de NAT existentes en la red. Para ello se utilizará un programa creado con el objetivo de mandar y recibir paquetes que posteriormente serán analizados. Una vez preparado el banco de pruebas, se haría pruebas con varios router caseros (NAT) y con varias redes móviles. Con esto, se busca obtener unos resultados estadísticos que muestren las diferencias en los tipos de NAT y qué presencia tiene cada uno de ellos en la actualidad. Para finalizar, se darán unas conclusiones del trabajo.

Palabras clave: NAT, P2P, resultados estadísticos

Abstract

Today, a large part of user devices are behind a router with NAT (both in home networks and in 4G/5G phone connections, etc.). Therefore, NAT is an interesting subject with a great implication in many applications, from network games to the use of P2P networks. This project aims to identify and evaluate the different types of NAT existing in the network. To do this, a program created with the aim of sending and receiving packets that will later be analyzed will be used. Once the test bench has been prepared, tests will be carried out with various home routers (NAT) and with various mobile networks. With this, we seek to obtain statistical results that show the differences in the types of NAT and the presence of each of them at present. Finally, some conclusions of the work will be given.

Key words: NAT, P2P, statistic outcome

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En el tejido de la era digital, donde la interconexión y la comunicación desempeñan un papel esencial en nuestra vida cotidiana, conceptos que enmarcan la infraestructura de la red a menudo pasan desapercibidos para el usuario común. Uno de estos conceptos fundamentales es Network Address Translation (NAT), una técnica que actúa como un conductor silencioso detrás de la cortina digital, orquestando la distribución de direcciones en la vasta red global. A medida que los dispositivos se multiplican y la tecnología evoluciona, NAT ha mantenido su posición central, permitiendo la comunicación fluida y segura entre dispositivos, tanto en el hogar como a nivel empresarial [1].

En su esencia, NAT es un componente indispensable en la administración de direcciones IP, permitiendo una distribución eficiente de direcciones limitadas y actuando como barrera protectora entre las redes internas y las amenazas externas. Desde su concepción y desarrollo inicial hasta su papel en la actualidad, NAT ha evolucionado para abordar desafíos cambiantes en el agotamiento de direcciones IPv4, la creciente adopción de IPv6 y la seguridad de red [2].

Aunque NAT ofrece ventajas significativas, presenta desafíos en la comunicación bidireccional. Dado que múltiples dispositivos comparten una única dirección pública, la comunicación iniciada desde el exterior de la red puede ser complicada. Esto se debe a que, sin configuraciones adicionales, los dispositivos internos no pueden ser abordados directamente desde el exterior.

En un panorama digital en constante evolución, donde las amenazas cibernéticas adoptan nuevas formas, la capacidad de identificar y rastrear dispositivos en una red se vuelve esencial para la detección temprana de intrusiones y para la gestión eficiente de la red. Aquí es donde los detectores de NAT entran en juego. Estas herramientas se centran en reconocer patrones y comportamientos que indican la presencia de la traducción de direcciones de red en un dispositivo, permitiendo la identificación precisa de nodos[1] en una red que anteriormente estaban ocultos detrás de la NAT.

A medida que la tecnología avanza y las demandas de conectividad siguen creciendo, la traducción de direcciones de red sigue siendo un área de investigación y desarrollo. La transición a IPv6, con su abundancia de direcciones IP, cambiará el panorama de NAT. Sin embargo, aún existen desafíos en la gestión eficiente de direcciones y en la habilitación de la comunicación transparente en un mundo cada vez más interconectado.

Al sumergirse en la esencia de la NAT, este estudio ilumina su importancia subyacente en el tejido mismo de la comunicación moderna. A medida que continuamos explorando las maravillas de la conectividad global, es imperativo comprender y apreciar el

[1] Nodo: Punto de conexión, unión o interacción de varios elementos. En este caso son ordenadores conectados a la red.

papel de la traducción de direcciones de red en la construcción y protección de nuestras redes digitales interconectadas.

1.1 Motivación

Realizar un proyecto de estudio estadístico sobre Network Address Translation (NAT) se motiva por varios factores cruciales. En primer lugar, brinda una comprensión profunda sobre cómo se distribuyen las direcciones IP en una red y cómo esto impacta la conectividad. Esto resulta especialmente útil para administradores de redes y profesionales de TI que buscan entender cómo NAT influye en la dinámica de una red. Además, el análisis estadístico puede optimizar la gestión de recursos al identificar patrones en el uso de direcciones IP y el tráfico a través de NAT. Esto conduce a una utilización más eficiente de recursos como direcciones IP y ancho de banda.

En términos de seguridad, el estudio estadístico puede detectar patrones de tráfico sospechoso o anormalidades, señalando posibles amenazas cibernéticas. Esto empodera a los profesionales de seguridad para reconocer actividades maliciosas y fortalecer las defensas de la red. Planificar para la escalabilidad se vuelve más sencillo con un estudio estadístico, ya que proporciona información sobre cómo cambian los patrones de tráfico con la expansión de la red. Esto facilita la planificación para futuras expansiones.

El trabajo de final de grado es una gran oportunidad de asentar las bases de nuestro futuro profesional, ya que se muestran nuestras inquietudes dentro del vasto mundo de la informática. En este caso, está claro que el área de conocimientos se centra en la administración de redes y se acerca a la ciberseguridad, un sector profesional muy interesante. Como profesional en estos ámbitos, se debe conocer los entresijos de las redes informáticas y NAT es un claro ejemplo de ello. Es por eso que entender el funcionamiento de este método y el estado actual del mismo ha supuesto un gran aliciente a la hora de elegir este proyecto. También ha influido en la elección el deseo de aportar un grano de arena a la evolución de NAT, intentando mostrar todas las opciones que ofrece y dando una visión más o menos amplia de su presencia en la red.

1.2 Objetivos

En este proyecto se plantea el objetivo principal de identificar y evaluar los diferentes tipos de NAT existentes en la red. Para ello, se desarrollará una arquitectura de red con esa función y se procesarán los resultados obtenidos. La arquitectura será creada y explicada de forma que pueda ser replicable para futuros proyectos. En general, este proyecto contribuirá a la comprensión más profunda de cómo se gestionan la traducción de direcciones en las redes, cómo afectan a diferentes aspectos y cómo las organizaciones pueden tomar decisiones informadas para optimizar su red en función de sus necesidades específicas.

Estos objetivos se lograrán abordando una serie de subobjetivos:

1. Estudiar y analizar la traducción de direcciones de red.
2. Mostrar todos los tipos de NAT y sus diferencias.
3. Crear un conjunto de servidores y desplegarlos en diferentes entornos.
4. Desarrollar una aplicación móvil que interactúe con la arquitectura de servidores.

5. Examinar los paquetes que transitan por la arquitectura de red mediante un analizador de protocolos.

6. Interpretar los resultados obtenidos en las pruebas mediante estadísticas.

Para finalizar, los servidores y la aplicación móvil son un medio para alcanzar el objetivo principal, pero que conllevarán gran parte del desarrollo. Una vez hayamos obtenido los resultados y se expliquen, llegaremos a unas conclusiones definitivas.

1.3 Estructura de la memoria

En el próximo capítulo, Estado del arte, se pretende informar al lector de la situación actual de los conceptos abordados en este proyecto. Los dos principales objetos de estudio serán IP y NAT, se definirán ambos conceptos teóricamente y se profundizará en sus variantes. Como último punto en este capítulo, se mencionará qué puede mejorar nuestro TFG en estos aspectos, justificando su desarrollo.

En el capítulo 3 se pasará a la fase de diseño de la solución, donde el primer paso será aproximar al lector el problema que se pretende solucionar y la forma de abordarlo. A continuación, se explicará la solución propuesta a este conflicto además de dar a entender el porqué de las opciones escogidas para solventar el problema (arquitectura y herramientas elegidas).

El cuarto capítulo, como el nombre indica, es un desarrollo de lo mencionado en el anterior capítulo. Se explican en profundidad todos los elementos de la red, cómo se han creado con sus respectivas herramientas y cómo interactúan entre ellos, dando ejemplos de errores que se han superado en el proceso. En esta sección también explicarán con ejemplos visuales los casos de uso de la aplicación.

El quinto y penúltimo capítulo consiste en mostrar diversas pruebas de validación de la arquitectura mediante capturas del tráfico de red y la terminal.

Por último, el capítulo 6 se aportarán las conclusiones a las que hemos llegado realizando este proyecto y posibles trabajos futuros relacionados con el mismo.

CAPÍTULO 2
Estado del arte

El estado de arte introduciremos ciertos conceptos relacionados con este trabajo. Entre ellos está el protocolo IP, NAT y sus tipos. Además, se hará una crítica al estado del arte actual.

2.1 Internet Protocol y sus versiones

Comenzaremos explicando qué es este protocolo, sus funciones y versiones más utilizadas. Según el RFC 791 [1], documento gestionado por consorcio de colaboración técnica más importante de Internet (Internet Engineering Task Force), IP es un protocolo que proporciona los medios necesarios para la transmisión de bloques de datos llamados datagramas en sistemas interconectados de redes de comunicación de ordenadores[3].

IP tiene varias funciones muy conocidas y utilizadas. Entre ellas encontramos el direccionamiento, el enrutamiento y la fragmentación. La primera se refiere a la forma de asignar una dirección IP y cómo se dividen y se agrupan subredes de equipos. Por supuesto, esta asignación se ve afectada por el tema principal del trabajo, NAT, pero ya lo veremos más adelante.

El enrutamiento es la forma en la que los paquetes se hacen llegar al destino a través de la red. Es la función que busca un camino, el más rápido normalmente, en la topología de la red en ese instante. Típicamente el encaminamiento es una función implantada en la capa 3 (capa de red) del modelo de referencia OSI [2].

Por último, la fragmentación es un mecanismo que permite separar un paquete IP entre varios bloques de datos, si su tamaño sobrepasa la unidad máxima de transferencia (Maximum Transfer Unit - MTU) del canal. Los módulos internet usan campos en la cabecera internet para fragmentar y reensamblar los datagramas internet cuando sea necesario para su transmisión a través de redes de *trama pequeña*.

2.1.1. IPv4

En la actualidad, la gran mayoría de dispositivos tienen una IP privada suministrada por un servidor DHCP (protocolo de configuración dinámica de hosts [3]) y utilizan

[1] RFCs o Request for Comments: Son documentos numéricos en los que se describen y definen diferentes protocolos, conceptos, métodos y programas de Internet.

[2] OSI: Modelo de interconexión de sistemas abiertos. Se trata de siete capas de abstracción que pretenden estandarizar el intercambio de información entre dispositivos diferentes.

[3] Host: Término adoptado del inglés que significa anfitrión y hace referencia a computadoras u otros dispositivos (tabletas, móviles, portátiles) conectados a una red que proveen y utilizan servicios de ella.

el protocolo IPv4 para el intercambio de paquetes. Este protocolo se diseñó para asignar direcciones individuales a los dispositivos para poder comunicarse con ellos a través de la red. Lamentablemente, estas direcciones se volvieron cada vez más limitadas hasta agotarse por completo (un total de 2 elevado a la potencia de 32 direcciones) hace aproximadamente doce años.

Una dirección IPv4 es un número de 32 bits [4] que identifica de forma exclusiva una interfaz de red de un sistema. Se divide en cuatro campos de ocho bits, dándole la famosa forma X.X.X.X. Por supuesto, este protocolo utiliza las funciones vistas anteriormente y, además existen particularidades en la asignación de direcciones. El motivo es que ciertas autoridades han restringido el uso general de varias direcciones y se han reservado para funciones especiales [4]. Un caso muy común y que veremos en la solución del trabajo es la dirección 127.0.0.0/8, que se reservan para direcciones loopback, es decir, una dirección especial que los hosts utilizan para dirigir el tráfico hacia ellos mismos.

2.1.2. IPv6

El último apartado de esta sección está dedicado al protocolo IPv6, la actualización al protocolo anterior. Dado el problema que apareció hace bastantes años de escasez de direcciones, la solución natural era aumentar el número de las mismas. Esto comenzó a desarrollarse hace veinticinco años y se puso operativo hace siete años aproximadamente.

A diferencia del protocolo anterior, IPv6 proporciona 2 elevado a 128 (o 340 sextillones) direcciones [5], lo cual resolvía con creces el problema. Conseguir este número tan alto obviamente conlleva aumentar el tamaño de cada dirección. Cada una tiene ocho campos de dieciséis bits, pudiendo llegar a 65536 valores en cada campo o como suele ejemplificarse en informática, el valor FFFF en hexadecimal. Gracias a las direcciones más largas, la seguridad aumenta. Se calcula que, un ataque básico que hasta ahora tardaba 5 minutos, tardará ahora varios miles de millones de años en llevarse a cabo. Además, este protocolo también simplifica las arquitecturas de direccionamiento y de red, lo que permite una seguridad más robusta y una mayor escalabilidad de las aplicaciones.

Su objetivo, obviamente, era suplantar al menos óptimo IPv4 que para entonces ya estaba ampliamente normalizado. Por desgracia, hemos visto una falta de impulso a la hora de migrar al nuevo protocolo en la mayoría de países. El mapa posterior es una prueba de ello.

[4] Bit: Dígito binario que almacena la unidad mínima de información. Tiene dos estados: 0 y 1

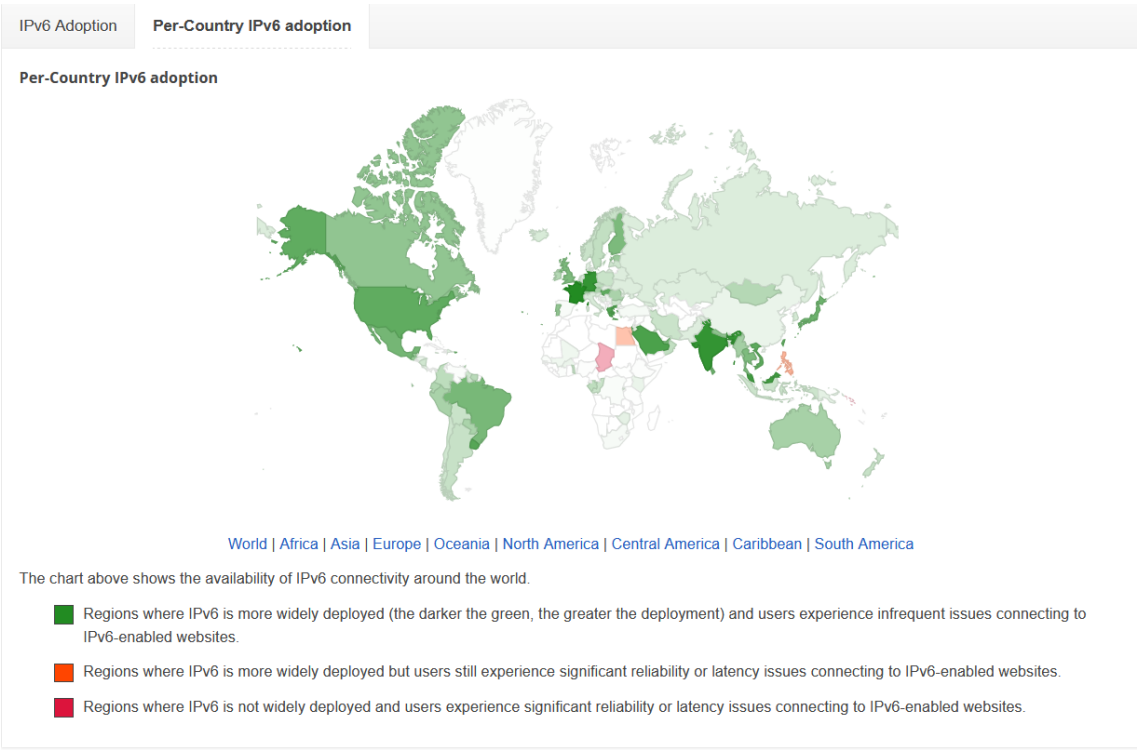

Figura 2.1: Adopción de IPv6 mundial

En esta imagen (2.1) podemos ver como ciertos países (EEUU, Francia o India) han sabido hacer una conversión eficaz teniendo alrededor de un 60 % de dispositivos usando IPv6 en Google. Pero también está la otra cara de la moneda, en la próxima imagen veremos como España en concreto no llega al 3 % actualmente.

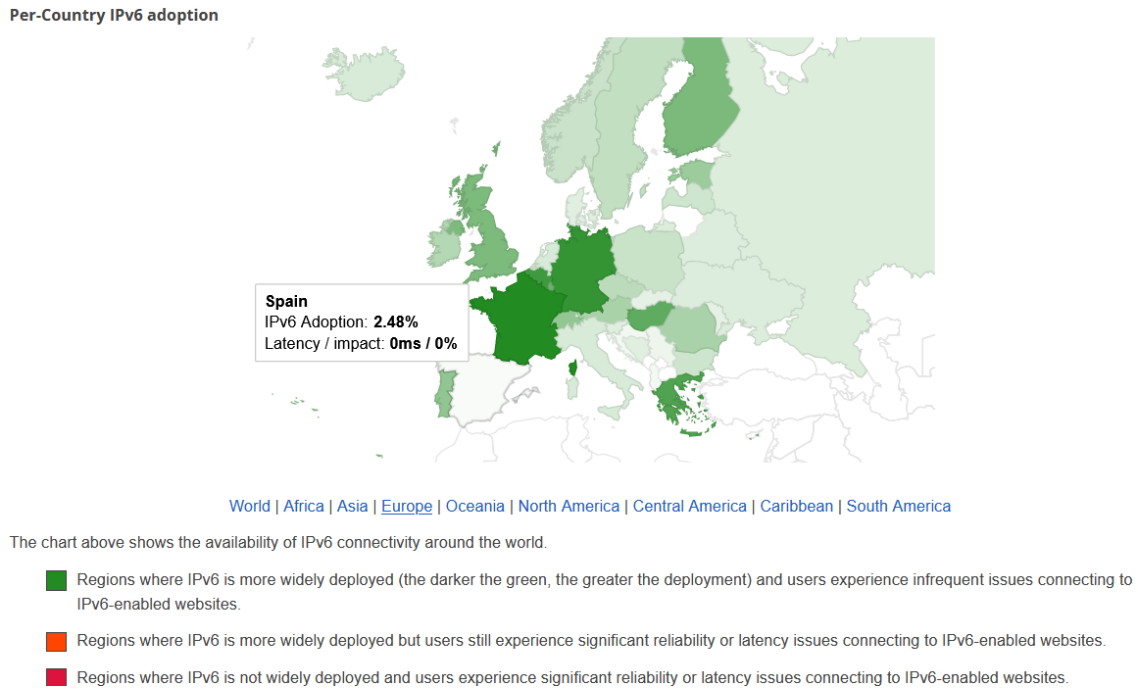

Figura 2.2: Adopción de IPv6 en Europa

Aunque los datos no son del todo prometedores después de tantos años, el cambio a IPv6 no ha parado de crecer. Según otra estadística de Google [6], ahora mismo estamos

en una adopción del 43 % mundial. En los tres últimos años, se ha aumentado en más de un 10 % y el crecimiento podría ser exponencial si todos empiezan a hacerlo compatible.

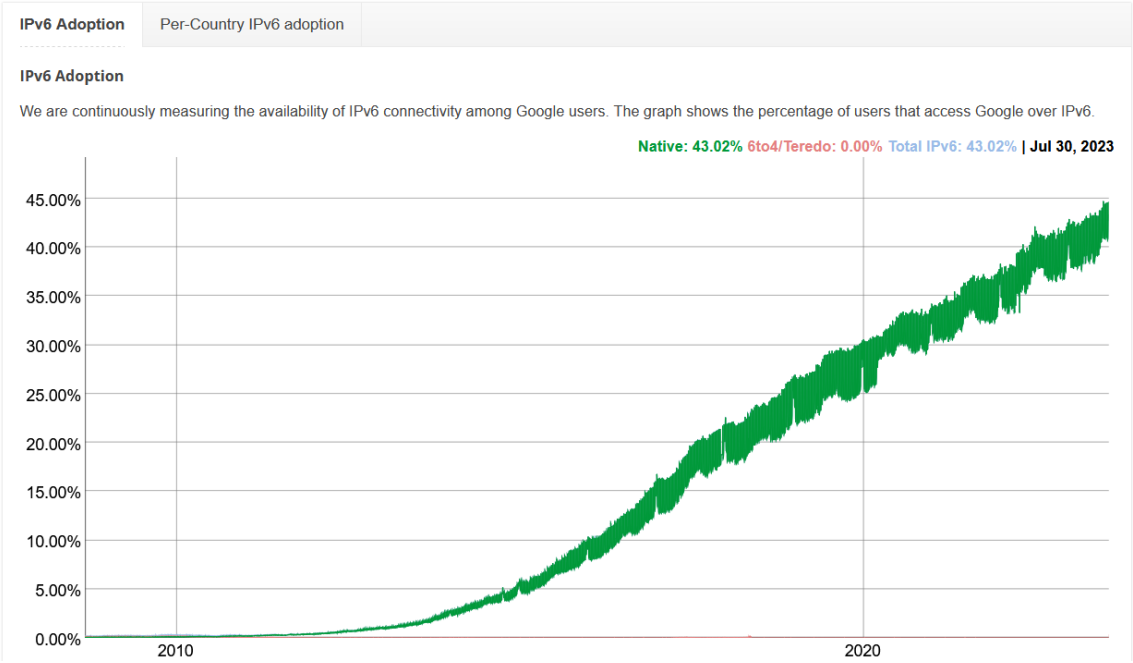

Figura 2.3: Crecimiento de la adopción de IPv6

Por último, se debe mencionar que, durante el periodo de desarrollo de este trabajo, ha habido ciertos avances en este ámbito. El primero es que, en junio de 2023, Movistar se ha convertido en el primer operador en España en direccionar con IPv6 el tráfico de Internet de sus usuarios móviles con el despliegue totalmente completado [7]. El segundo avance ha sido perpetuado por Amazon AWS en agosto del mismo año. Esta plataforma ha decidido empezar a cobrar el uso de direcciones IPv4 a 3,6 dólares al mes [8]. Los motivos principales son el incremento sostenido de los precios que han sufrido las viejas direcciones IP en los últimos años y la intención de empujar al público a ser más austero con el uso de estas direcciones y que planifiquen su migración a IPv6.

2.2 Traducción de direcciones de red (NAT)

Cuando nos referimos a IP, la mayoría piensa en IPv4 por su aplastante presencia en la red. Aunque IPv6 lleva bastante tiempo en el mercado, no se ha hecho un traslado masivo a este protocolo incluso sabiendo las ventajas que tiene. Uno de los motivos es precisamente que la traducción de direcciones de red ha proporcionado cierta comodidad o, visto de otra forma, la gran inversión que requiere migrar al nuevo protocolo.

Network Addres Translation, o NAT, es un método por el cual se convierten las direcciones IP en los paquetes transportados desde un dominio a otro, en un intento de proporcionar encaminamiento transparente a las máquinas de dos redes que asignan mutuamente direcciones incompatibles [9]. El uso más común de este mecanismo es conectar redes internas con direcciones privadas no registradas con la red externa de direcciones públicas globalmente registradas.

Además de la conservación de direcciones IP, el NAT también proporciona cierto nivel de seguridad al actuar como un cortafuegos básico. Al mantener ocultas las direcciones

IP privadas detrás de la dirección IP pública, los dispositivos internos se vuelven más difíciles de detectar y atacar desde el exterior.

Existen varios tipos de NAT, cada uno con sus propias características y aplicaciones. Cada tipo tiene sus propias ventajas y desventajas según el contexto de uso y los requisitos de la red. Si el lector está familiarizado con NAT, puede conocer los tipos de NAT estático, dinámico y de sobrecarga. Estos no son los únicos tipos, ya que existe NAT para el capeado independiente del extremo (los mencionados anteriormente) pero también existe NAT para el filtrado dependiente del extremo.

Los tipos de NAT mencionados anteriormente (estático, dinámico, de sobrecarga, etc.) son categorías que describen cómo se realiza la traducción de direcciones IP y cómo se asignan las direcciones privadas y públicas. Sin embargo, los tipos de NAT expuestos a continuación (full-cone, restricted-cone, restricted-port, symmetric) se refieren a las características específicas del comportamiento de NAT en términos de cómo establecen las conexiones y cómo gestionan los flujos de datos. Estos tipos de NAT se basan en las respuestas que un dispositivo NAT da a las solicitudes de conexión entrantes y salientes.

2.2.1. Capeado independiente del extremo

Existen varios tipos de NAT (Network Address Translation) que se utilizan en redes de computadoras para permitir la comunicación entre dispositivos en diferentes redes y con diferentes direcciones IP. Estos son los tipos más comunes de NAT:

NAT Estática (Static NAT)

En este tipo de NAT, se asigna de manera fija una dirección IP privada a una dirección IP pública específica. Cada vez que un dispositivo con la dirección IP privada se comunica con el exterior, la dirección IP pública asignada se utiliza en la comunicación [10]. Es comúnmente utilizado cuando se necesita acceder a un recurso interno (como un servidor web) desde Internet.

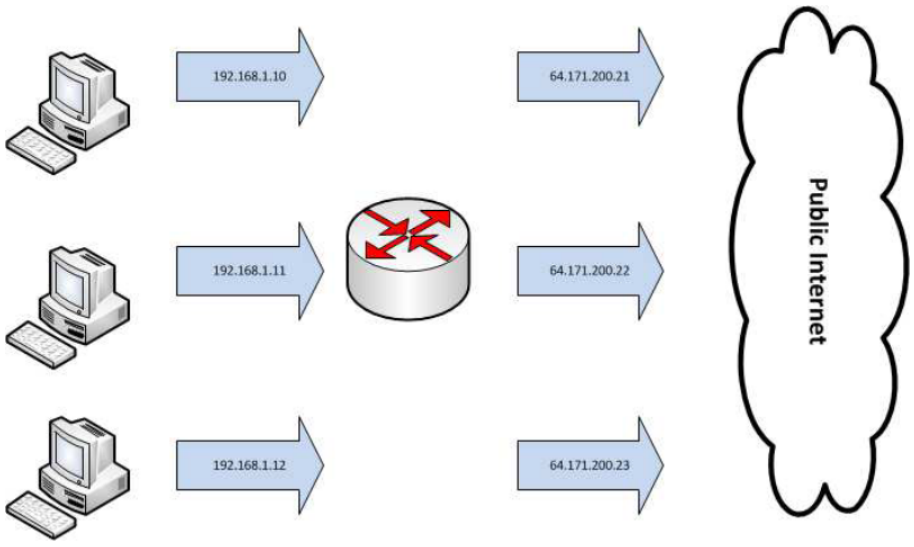

Figura 2.4: Ejemplo de NAT estática

NAT Dinámica (Dynamic NAT)

En la NAT dinámica, un grupo de direcciones IP privadas se asigna a un conjunto de direcciones IP públicas disponibles. A medida que los dispositivos internos inician comunicaciones, se les asigna una dirección IP pública disponible del conjunto. Una vez que la comunicación se completa, esa dirección IP pública vuelve al conjunto para ser utilizada por otro dispositivo [11]. En la práctica, podríamos comprobar este fenómeno comparando las IPs de varios paquetes enviados por el mismo nodo. Si alguno o varios de estos paquetes tienen diferente dirección IP (estando completamente seguros de que provienen del mismo nodo), entonces se está aplicando una NAT dinámica.

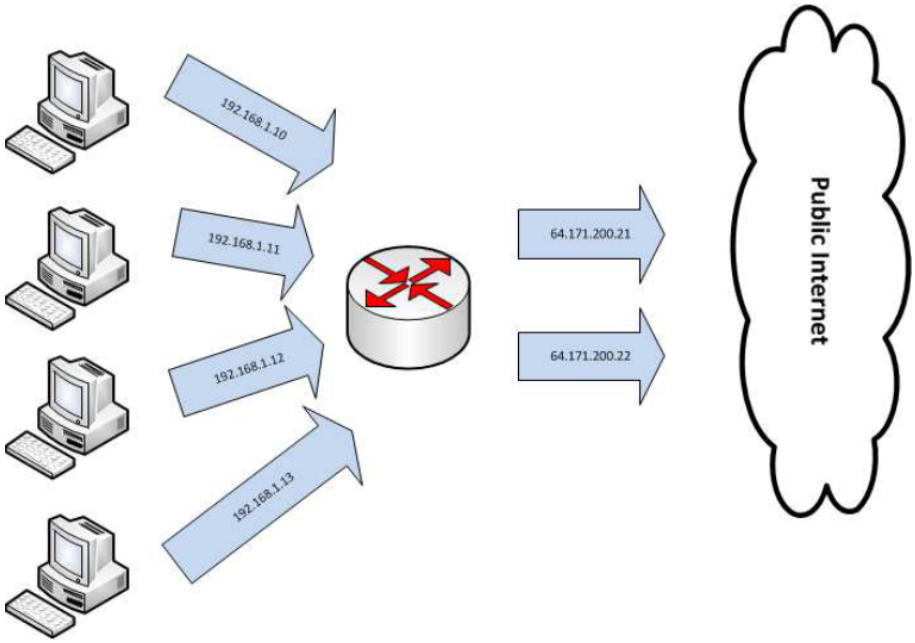

Figura 2.5: Ejemplo de NAT dinámica

NAT de Sobrecarga (Port Address Translation, PAT o NAT Overloading)

Este tipo de NAT permite que varios dispositivos internos compartan una única dirección IP pública. Cada dispositivo utiliza una dirección IP privada única, pero comparten la misma dirección IP pública y se distinguen por los números de puerto. Esto se logra mediante la modificación de los números de puerto en la dirección IP y el puerto de origen de los paquetes [11]. El NAT de sobrecarga es comúnmente utilizado en redes domésticas y pequeñas empresas.

Estos son solo algunos de los tipos de NAT más comunes. La elección del tipo de NAT dependerá de los requisitos de la red, la cantidad de dispositivos y la forma en que se desea administrar y asegurar la comunicación entre dispositivos internos y externos.

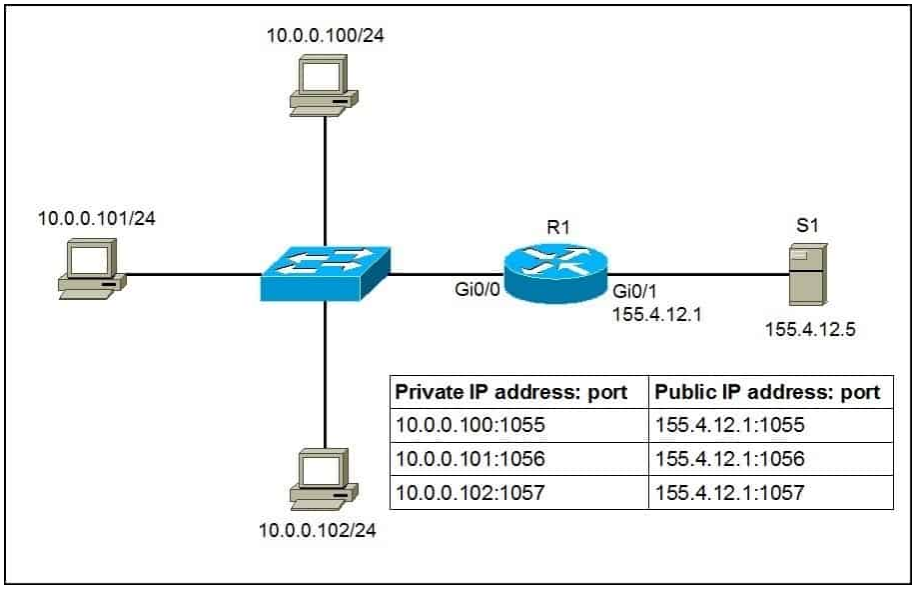

Figura 2.6: Ejemplo de NAT de sobrecarga (PAT)

2.2.2. Filtrado dependiente del extremo

Estos tipos de NAT se basan en las respuestas que un dispositivo NAT da a las solicitudes de conexión entrantes y salientes [12].

Full-Cone NAT

Full-Cone NAT mapea una dirección IP interna y un puerto a una dirección IP pública y un puerto en forma unidireccional. Es más, cualquier host externo puede enviar datos a esa dirección IP y puerto públicos, y serán entregados al dispositivo interno. Es el tipo más permisivo y sin duda puede conllevar problemas de seguridad ya que el dispositivo interno está totalmente expuesto.

En la imagen aclaratoria 2.7 podemos ver como el puerto 4702 y una IP cualquiera ha sido asociado al nodo interno, que tenía la IP 10.0.0.8 y el puerto 2050. Por tanto, cualquier nodo externo, en este caso dos servidores han podido enviar paquetes al nodo interno a través de estos puerto e IP externos.

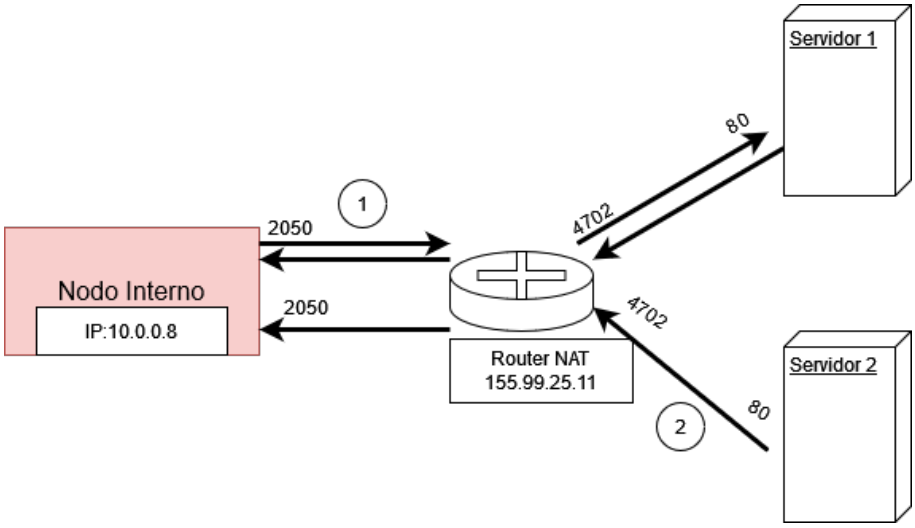

Figura 2.7: Ejemplo de Full-Cone NAT

Restricted-Cone NAT

Este tipo de NAT es similar al full-cone NAT, pero con una restricción adicional. Solo los hosts externos a los que el dispositivo interno se ha conectado previamente pueden enviar datos a la dirección IP pública asignada. Decimos solo IP porque puede recibir paquetes de otros puertos siempre que lleguen con la IP a la que ha enviado el nodo interno. El resto de los envíos que reciba este host interno serán rechazados. De esta forma, se garantiza al menos que el nodo interno tiene el control (a priori) de elegir qué hosts van a pasar el filtro de NAT.

La imagen aclara esta variante NAT 2.8, mostrando primero todo el intercambio de paquetes entre el servidor 1 y el nodo interno y el intento del servidor dos que es rechazado. Como el host interno envía previamente información al servidor 1, lo habilita a responder e incluso mandar paquetes desde otro puerto.

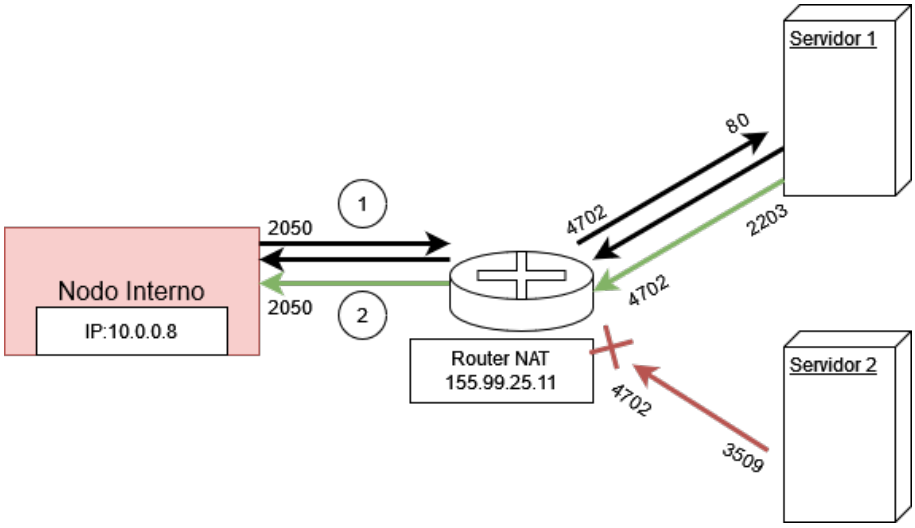

Figura 2.8: Ejemplo de Restricted-Cone NAT

Restricted-Port NAT

En este caso, el NAT restringe no solo la dirección IP del host externo sino también el número de puerto, que debe ser el mismo que el puerto por el cual el host interno realizó la conexión con el externo. Por tanto, solo un host externo que haya recibido paquetes de un dispositivo interno podrá comunicarse a través de la misma IP y puerto que haya usado el nodo interno para enviar paquetes.

La diferencia con restricted-cone se muestra en el esquema posterior 2.9. La única diferencia con la imagen de restricted-cone es que el paquete enviado desde el servidor 1 con un puerto diferente al 80 (que es el puerto al que envía el nodo interno) es rechazado.

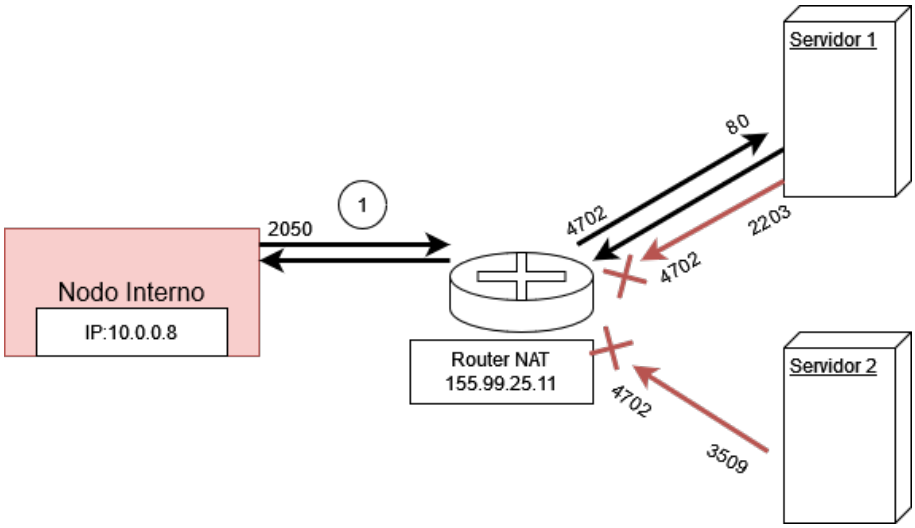

Figura 2.9: Ejemplo de Restricted-Port NAT

Symmetric NAT

Este tipo de NAT es el más restrictivo de todos, pero también se podría argumentar que es el más seguro. Aun así, es sin duda el que más problemas de compatibilidad incluye.

Cada solicitud de conexión desde un host interno genera una asignación de dirección IP y puerto únicos. Esto significa que las respuestas solo se dirigen a la dirección IP y puerto que hizo la solicitud, y no se pueden reutilizar para otras conexiones. Todas las solicitudes de los mismos puertos y direcciones IP internos y hacia un puerto y una dirección IP de destino específicos, se asignan a los mismos puerto y dirección IP externos.

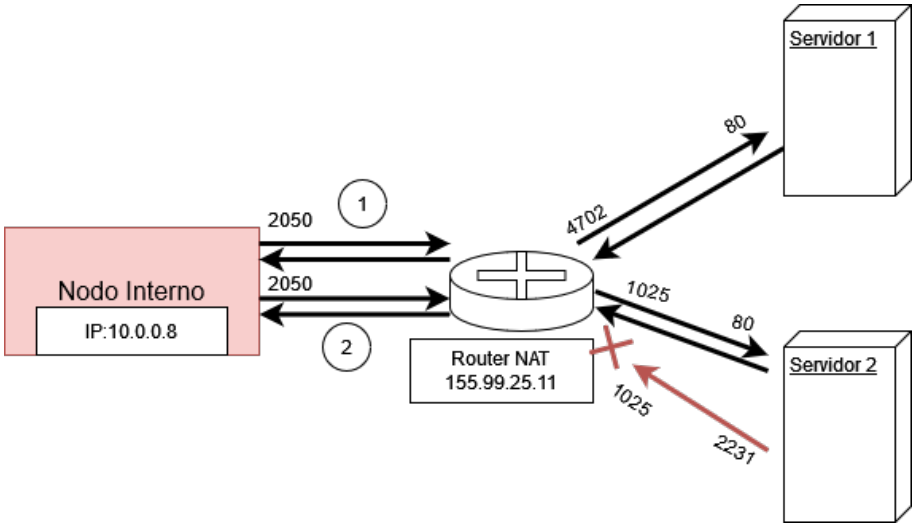

Figura 2.10: Ejemplo de NAT simétrica

Hole-Punching

Vamos a introducir ahora una técnica llamada Hole-Punching que permite establecer conexiones directas entre dos nodos que no se conocen y que utilizan NAT [13]. Para realizar la conexión entre ambos dispositivos, estos se conectan previamente a un servidor temporal no restringido por NAT y le indican sus direcciones y puertos. Luego, el servidor transmite la información de cada cliente al otro y usando esa información cada cliente intenta establecer una conexión directa. Como resultado de las conexiones que usan números de puerto válidos, los cortafuegos o enrutadores restrictivos aceptan y reenvían los paquetes entrantes en cada lado. Esto es muy común en protocolos como torrent.

Ahora explicaremos cuán restrictivo es cada tipo de NAT, teniendo en cuenta esta técnica que teóricamente facilita la conexión UDP [5] entre nodos. Lo haremos a través de un esquema que muestra la compatibilidad de cada tipo con los demás tipos 2.1.

Según el esquema, cono completo, cono restringido y cono de puerto restringido conforman las NAT permisivas. Esto se refiere a la capacidad que tienen las tres de poder utilizar la técnica Hole-Punching sin problemas de enrutamiento siempre y cuando los firewalls o routers de ambos nodos tengan uno de estos tres tipos de NAT. El problema llega con NAT simétrica, que al cambiar de dirección y puerto con cada conexión, es muy complicado si no imposible establecer una dirección y puerto para futuras conexiones. El motivo es simple, la primera conexión se realiza con el servidor no restringido, por lo que NAT asigna al nodo A interno una IP y puerto externos para dicha comunicación. Pero una vez tenga la dirección y puerto del otro nodo B e intente transmitir paquetes, NAT asignará un nuevo par de IP y puerto al nodo A.

Esto es contradictorio porque el otro nodo B interno, que también se ha conectado con el servidor, ha recibido el par IP y puerto del nodo A que se ha utilizado con el servidor y no el que el nodo A pretende utilizar para comunicarse con el nodo B. Por tanto, cuando se trata de cono de puerto restringido y simétrico, no puede funcionar la técnica Hole-Punching porque siempre va a haber discrepancias en la IP y puerto que utilizan.

La razón por la que existen estos tipos de NAT es que ofrecen niveles variables de seguridad, funcionalidad y eficiencia en diferentes escenarios.

[5] UDP o User Datagram Protocol: Protocolo del nivel de transporte basado en la transmisión sin conexión de datagramas.

		Site B			
		Permissive NAT			
	NAT Type	Normal (Full Cone)	Restricted Cone	Port Restricted Cone	Symmetric
Site A: Permissive NAT	Normal (Full Cone)	Routable	Routable	Routable	Routable
	Restricted Cone	Routable	Routable	Routable	Routable
	Port Restricted Cone	Routable	Routable	Routable	Not Routable
	Symmetric	Routable	Routable	Not Routable	Not Routable

Tabla 2.1: Compatibilidad con Hole-Punching

2.3 Crítica al estado del arte

Una revisión a fondo de los trabajos realizados en la UPV nos ha mostrado que no existe prácticamente ningún trabajo que abarque el método NAT de la forma que lo hace este proyecto. Por desgracia, no he podido utilizar los pocos trabajos relacionados con el tema ya que son privados y muchos otros proyectos simplemente mencionan de pasada el método NAT. Visto lo visto, decidimos buscar otros trabajos fuera de la UPV con algo más de éxito ya que existen ciertos trabajos que tienen un objetivo parecido.

Lo más similar que hemos podido encontrar es un proyecto que pretende descubrir los tipos de NAT de forma pasiva en vez de activa [14], como hace este trabajo, y otra página web que detecta si la NAT que se utiliza es simétrica o no. Este último tiene el objetivo de ayudar en la medida de lo posible a evitar los problemas de compatibilidad que hemos visto con la técnica Hole-Punching anteriormente.

Por otro lado, las herramientas utilizadas en este trabajo son muy conocidas y de gran relevancia en la actualidad. Por ejemplo, Android Studio o Visual Studio Code son dos grandes aplicaciones que tienen bastante presencia en los trabajos de la ETSINF. También podemos encontrar muchos proyectos que utilizan una arquitectura basada en servidores para alcanzar los objetivos descritos en sus trabajos.

En definitiva, este proyecto no es completamente nuevo a ojos de Internet, donde se puede encontrar infinidad de proyectos, pero sí lo es para la UPV. Se pretende con este trabajo dar más reconocimiento a un método ampliamente utilizado y que ha aportado gran estabilidad al esquema de red que tenemos actualmente. Es por eso que creemos que puede ayudar a futuras investigaciones.

CAPÍTULO 3
Diseño de la solución

En este capítulo se describirá el problema y se mencionarán las decisiones tomadas con el fin de llevar a cabo la solución del problema. Además, se introducirán algunas tecnologías utilizadas con este propósito.

3.1 Análisis del problema

Como se ha explicado anteriormente, las direcciones de IPv4 son claramente inferiores al número de puntos finales que requieren de una. En consecuencia, es muy difícil ver direcciones públicas asignadas a dispositivos individuales actualmente. Aquí es donde entra NAT. Gracias a este método, es posible mapear direcciones IP internas a una o varias direcciones IP externas registradas. Aunque no es su única función, es una forma muy efectiva de aprovechar las escasas IPs públicas.

Aunque el método NAT aporta muchas ventajas tecnológicas y solventa gran variedad de problemas, también tiene inconveniencias. Una de ellas, por ejemplo, es la posible pérdida de paquetes según el filtrado NAT que tenga un host. Si se desconoce el tipo de filtrado de una dirección, es posible que algunos paquetes no lleguen o produzcan errores una vez llegado al destino. Por otro lado, se producen retrasos de reenvío al perder tiempo en utilizar el método NAT en cada paquete y esto afecta en entornos vastamente utilizados como protocolos VoIP.

Dadas las circunstancias, sería de gran ayuda conocer previamente el método de traducción que posee cada host para facilitar la comunicación posterior.

3.2 Solución propuesta

Teniendo en cuenta las desventajas del método Network Address Translation, se propone diseñar una solución que servirá para detectar el tipo de NAT de un dispositivo. Tanto el capeado independiente del extremo como el filtrado dependiente del extremo se podrán a prueba en esta solución.

Para comenzar, se implementará una aplicación que servirá para mandar mensajes desde un dispositivo móvil u ordenador (según cual sea cambiará ligeramente el código). Esta aplicación se utilizará como medio de comunicación entre el dispositivo cliente y dos servidores que estarán ejecutando un programa a la espera de los primeros paquetes enviados por el cliente.

Después, utilizando los dos servidores mencionados, que estarán detrás de una IP pública cada uno, se iniciará un programa de escucha en un determinado puerto donde

llegarán los paquetes UDP. Cada uno tendrá una forma de gestionar estos paquetes, pero serán bastante parecidos.

El último dispositivo que conforma esta solución es un servidor maestro al que los otros dos servidores tendrán que responder. Este también ejecutará un programa de escucha, pero solo recibirá información de los servidores esclavos. Su función consiste en comparar los datos recibidos para adivinar el tipo de NAT del cliente. Para ello, se utilizará la IP y el puerto de los paquetes enviados al principio por el cliente a ambos servidores esclavo.

Aunque el intercambio de información entre dispositivos no es muy amplio, es necesario poner cierto orden y así evitar errores de interpretación de mensajes. Para ello se han ideado tres tipos de mensajes que tendrán funciones diferentes y dependen del estado de la comunicación.

Tipo 1

En primer lugar, los mensajes *tipo 1* serán enviados por los clientes. Su propósito principal es llegar a dos servidores diferentes a la vez con la información necesaria para detectar el capeado independiente del extremo. Serán los únicos mensajes utilizados con este objetivo.

Tipo 2

En segundo lugar, los mensajes *tipo 2* serán utilizados para notificar el inicio de la segunda prueba (la detección de filtrado dependiente del extremo). En este tipo también se incluye el intento de comunicación con el cliente por parte de los servidores. La gracia de este intento es que uno de los servidores esclavo es desconocido para el cliente, de esta forma se puede comprobar todo el espectro de filtrado NAT que existe.

Tipo 3

El último tipo de mensaje es obviamente *tipo 3*. La única función de estos paquetes es notificar cual es el filtrado que se utiliza en el cliente. Será una comunicación realizada desde y hacia servidores, sin pasar por el cliente.

En la tabla inferior 2.1, podemos ver las diferencias entre los tipos mencionados, según sus funciones en la solución.

	Tipo 1	**Tipo 2**	**Tipo 3**
Capeado independiente del extremo	**X**		
Filtrado dependiente del extremo		**X**	**X**
Comunicación cliente-servidor	**X**	**X**	
Comunicación servidor-cliente		**X**	
Comunicación servidor-servidor		**X**	**X**

Tabla 3.1: Tipos de mensajes

3.3 Arquitectura del sistema

La arquitectura del proyecto ha variado según la fase de pruebas en la que se encontrara porque se han hecho pruebas en la red de la universidad y en la red exterior. Los

cambios más notables son las IPs usadas, pero también hay otros detalles menores que son dignos de mención.

Como se ha comentado anteriormente, la solución está compuesta por tres niveles: cliente, servidores esclavos y servidor maestro. Como se puede ver en la imagen inferior, el intercambio de paquetes se realiza por la interfaz Wi-Fi en todos los casos menos en la comunicación entre servidor maestro y servidor esclavo 1, que al estar en el mismo dispositivo se hace por loopback (IP 127.0.0.1). La interfaz loopback podría ser reemplazada por otra interfaz Wi-Fi si se añadiera otro router con IP pública, pero en este caso no ha sido posible disponer de otro dispositivo y por suerte no altera el resultado.

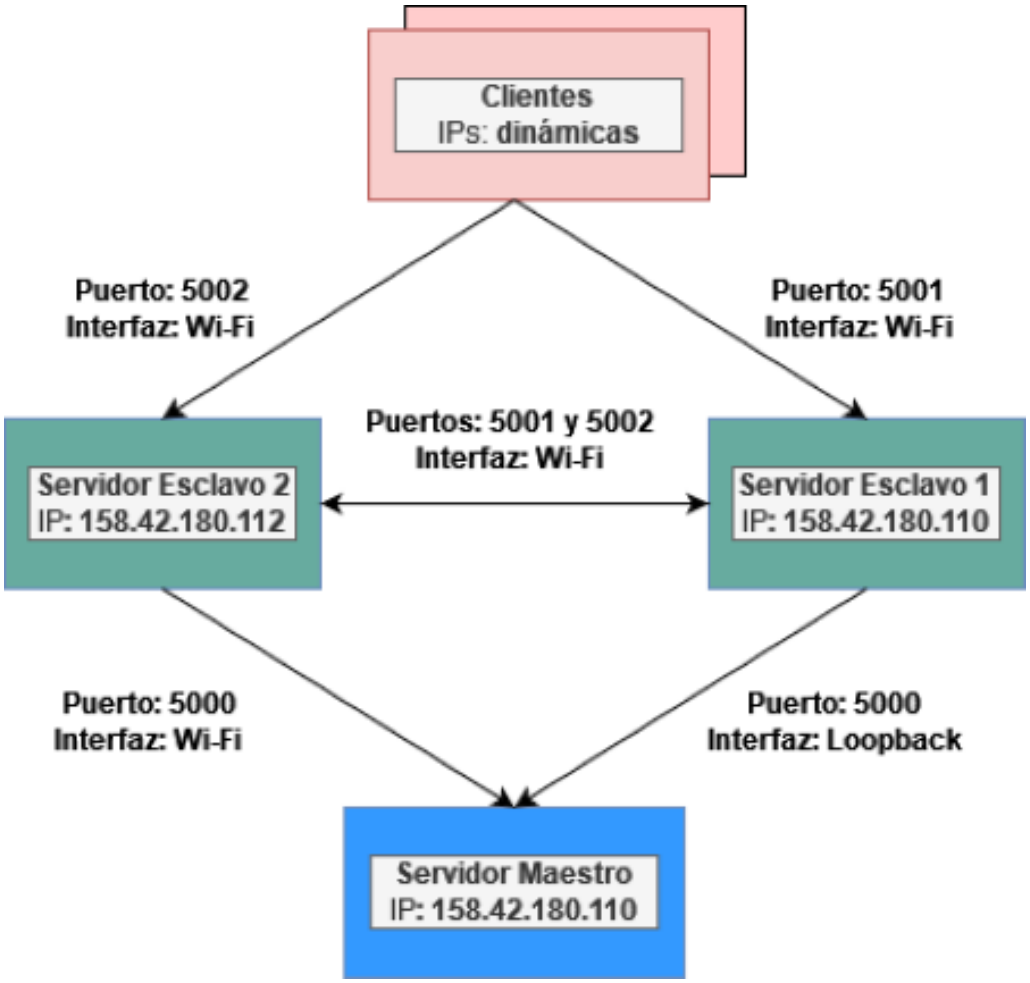

Figura 3.1: Topología de la UPV

La inclusión de dos servidores esclavo con IP pública es esencial porque es ahí donde se aplican las reglas de NAT en el router o firewall del cliente. Si solamente se utilizase uno, el único tipo de NAT que podríamos detectar es el estático por el lado del capeado independiente del extremo. En cuanto al filtrado dependiente del extremo, no se podría detectar el tipo de NAT simétrico. Sería una solución incompleta.

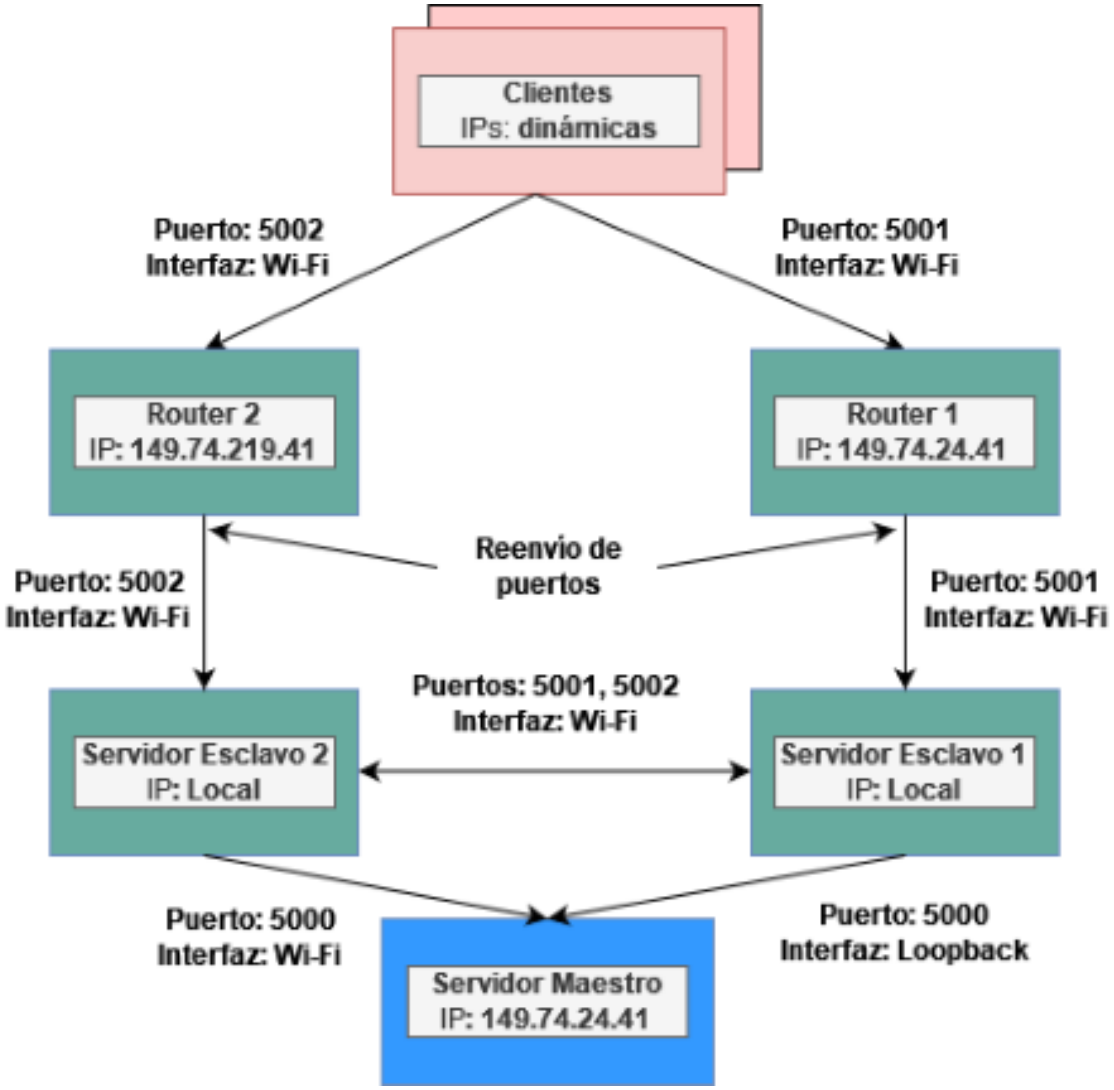

Figura 3.2: Topología bridge externa

En la imagen anterior, se puede observar que las IP no son propias de la red de la UPV. Las IP mostradas son dos routers privados que han sido ligeramente modificados para gestionar los paquetes UDP del sistema. Estos routers son Livebox 6 de Orange y tienen la opción *reenvío de puertos* que es precisamente lo que necesita el sistema. La forma de comunicarse con un dispositivo detrás de un router en la red es establecer una redirección de paquetes en el router. De esta forma, cuando reciba un paquete cuyo destino tenga el puerto del programa (5000, 5001 o 5002), se redirigirá al dispositivo que ejecuta dicho programa.

Cabe destacar que la comunicación entre el servidor esclavo 2 y el resto de los dispositivos no es directa en este caso. Como ya se ha mencionado, si el servidor maestro tuviera su propia IP pública así sería, pero al estar en el mismo dispositivo que el servidor esclavo 1, todos los paquetes deben pasar por el router 1 antes de llegar al servidor maestro o esclavo. El motivo por que aparece así en la imagen es únicamente para que sea comprensible.

3.4 Tecnologías utilizadas

En esta sección se hará mención de las herramientas y lenguajes utilizados en el desarrollo de la solución, justificando también el porqué de la elección de dichas tecnologías. En caso de tener que usar un lenguaje o entorno nuevo, debe ser uno que tenga un tiempo de aprendizaje y utilidad viable.

3.4.1. Entornos de desarrollo

Una parte esencial en cualquier trabajo que requiera escribir código para un programa es el entorno de desarrollo. Uno tiene que elegir aquel con el cual se siente cómodo y le ayude todo lo posible.

Visual Studio Code

Visual Studio Code o VS Code, es un editor de código desarrollado por Microsoft y apto para la gran mayoría de sistemas operativos. En nuestro caso, se ha utilizado Windows, sistema para el cual está soportado y actualizado. Aunque para este proyecto únicamente se ha utilizado para escribir código, con VS Code es posible depurar, realizar control integrado de Git [1], resaltado de sintaxis, finalización inteligente de código y muchas otras opciones. Toda esta variedad es posible gracias al apartado de extensiones que se pueden aplicar al editor.

Figura 3.3: Virtual Studio Code

En nuestro caso, se ha utilizado VS Code debido al conocimiento previo del editor y debido a las características tan útiles que posee. Una opción que permite es ejecutar el programa desde el mismo editor, lo que ahorra mucho tiempo en compilar si se quiere comprobar el funcionamiento del proyecto.

Android Studio

Android Studio es un entorno de desarrollo creado por Google que sirve para crear y compilar aplicaciones Android. Este IDE (Entorno de Desarrollo Integrado) ofrece muchas opciones y es uno de los entornos por excelencia. Resulta bastante intuitivo su uso,

[1] Git: Software de control de versiones de código abierto para aplicaciones. Su propósito es llevar registro de los cambios en archivos.

la interfaz es sencilla, fácil de aprender a utilizarla y todas las partes de la aplicación son muy accesibles a la hora de cambiar código.

Figura 3.4: Android Studio

Cuando se accede a la aplicación, pregunta en qué lenguaje se quiere escribir, da la opción de Kotlin o Java aunque C++ también lo permite. En nuestro caso, se eligió Java por mantener la consistencia dado que en Visual Studio Code se utilizó ese lenguaje.

Además, Android Studio ofrece un emulador de móvil llamado Android Virtual Devices que puede incluir la aplicación desarrollada por lo que es muy útil para realizar pruebas. Se puede crear virtualizaciones de una gran variedad de dispositivos, pero en nuestro caso con un móvil sencillo es suficiente ya que se pretende crear una aplicación aplicable a cualquier móvil.

3.4.2. Lenguaje de programación

Como se ha mencionado en el apartado anterior, el lenguaje utilizado en este proyecto es Java, un lenguaje y plataforma ampliamente reconocido. El motivo principal por el que se ha usado es el previo conocimiento del lenguaje, que ayuda a desarrollar una aplicación más compleja.

Figura 3.5: Java

Java tiene la ventaja de ser un lenguaje muy utilizado y por lo tanto contiene una vasta cantidad de librerías y soporte que muchos otros no tienen. Un ejemplo de cómo se ha

utilizado esto en nuestra solución es con la librería GSON que más adelante se explicará en detalle. Al tener Java tanto soporte se ha podido encontrar rápidamente soluciones a problemas que han aparecido en el proceso.

CAPÍTULO 4
Desarrollo de la solución

En este apartado, se explicarán los pasos seguidos para cumplir los requisitos del diseño de la solución, los problemas que han surgido y las decisiones que se han tomado para llegar a la implementación final.

Explicaremos el desarrollo de abajo hacia arriba, según la topología de la red. En primer lugar, se dará a entender cómo desarrollamos el servidor maestro, después los servidores esclavo y por último el cliente.

4.1 Solución inicial

4.1.1. DatagramSocket y DatagramPacket

La primera elección a realizar y que determina el resultado final, es la herramienta principal con la que se mandarán los paquetes en Java. La solución utilizará sockets [1], más concretamente usará las clases de DatagramSocket y DatagramPacket ya que proporcionan unos métodos sencillos pero eficaces.

DatagramSocket

DatagramSocket es una clase que extiende de la clase Object y que permite crear un objeto socket cuya finalidad es identificar un punto de envío o recepción. Sirve como conector a través del cual enviamos y recibimos paquetes UDP (datagramas).

La creación del objeto se consigue gracias a un constructor que proporciona esta clase y que tiene esta forma:

DatagramSocket socket = new Datagramsocket();

El constructor también tiene la opción de incluir como parámetro el puerto por el que se quiere recibir los paquetes, dentro de los paréntesis y con forma numérica (int).

Una vez creado el socket, se debe poner en marcha la recepción de paquetes. Esto se consigue con el método de instancia receive() que proporciona la clase DatagramSocket. Dentro de los paréntesis se pondrá como parámetro el paquete a recibir, que se creará con la clase que veremos ahora.

[1] Socket: Canal de comunicación por el cual dos procesos pueden intercambiar cualquier flujo de datos, en este caso datagramas

DatagramPacket

DatagramPacket es otra clase dedicada al manejo de datagramas que extiende de Object, pero en este caso se utiliza crear dichos paquetes. Al igual que la clase anterior, DatagramPacket tiene un constructor. Este método, en cambio, tiene varios parámetros que se pueden incluir. Entre ellos se encuentran el buffer con los datos, el tamaño del buffer, la dirección IP a la cual se enviará y el puerto (normalmente el que abre el método receive). La forma del constructor es la siguiente:

DatagramPacket paquete = new DatagramPacket(buffer, buffer.length, direccion, puerto);

Para enviar el paquete creado, se usará un método de DatagramSocket llamado send(). Dento de los paréntesis irá el paquete y al ser un método de instancia, se hará referencia al socket.

4.1.2. Gson

Gson es una librería de código abierto desarrollada por Google que permite trabajar con el formato de intercambio de datos JSON. Más concretamente, se utiliza para serializar y deserializar objetos en Java. En nuestro caso, esta librería es esencial para la comunicación entre servidores y clientes ya que JSON es el formato más efectivo a la hora de intercambiar información entre servidores de manera clara, liviana y rápida.

El funcionamiento de la serialización es relativamente sencillo. Primero se crea un objeto JsonObject con un constructor proporcionado por la librería Gson. Después se introducen los datos con otro método de instancia (referenciando al objeto Json) llamado addProperty de misma librería. JSON acepta varios tipos de datos (números, strings, booleanos...). En nuestra solución únicamente se necesitan strings y números para comunicar las IPs y puertos además de algunos datos de mantenimiento.

Como tal, la edición de datos una vez han sido introducidos no es posible, por lo tanto, se debe eliminar el dato que sea necesario e introducir uno nuevo. Para eliminar un dato existe el método remove().

Por otro lado, la deserialización es un poco más compleja y que ha llevado a errores. La idea principal es, a partir de un String, crear un objeto JsonObject que los servidores y clientes puedan manejar (ver sus valores, cambiarlos).

Para ello, lo primero que se deba hacer es crear un objeto Gson con su constructor. El siguiente paso es evitar errores en la lectura de datos ya que se suelen producir malformaciones de strings [15]. Por último, se crea un JsonObject cuyo valor es el resultado del método fromJson de la librería Gson a partir del objeto gson, los datos del string y el tipo al que se pretende convertir (en nuestro caso JsonObject). Es de gran utilidad este fragmento de código para entender dicho proceso.

```
Gson gson = new Gson();
JsonReader reader = new JsonReader(new StringReader(JSON));
reader.setLenient(lenient:true);
JsonObject jsonObject = gson.fromJson(reader, typeOfT:JsonObject.class);
```

Figura 4.1: Código de deserialización Gson

4.1.3. Servidor maestro

Como ya se ha explicado anteriormente, el encargado de realizar las comprobaciones finales, que revelarán el tipo de NAT de un dispositivo, será el servidor maestro. Este es el primer servidor a desarrollar ya que no depende del resto, únicamente escucha en un puerto por el que recibe paquetes y no responde a ninguno.

La forma de recibir constantemente paquetes es sencilla, un bucle while con la condición true que crea un paquete al principio del bucle en el que introducirá los datos que le lleguen por el puerto abierto. Con los datos recibidos, se llamará a un par de métodos externos alojados en clases específicas para ello fuera del programa principal (main) que dejarán listo el objeto json para las próximas pruebas.

La primera parte de las comprobaciones consiste en determinar el tipo del mensaje, como los que se vieron en el apartado de la solución propuesta. Si es de tipo 1 implica que es un mensaje de uno de los dos servidores con los datos (puerto, IP) del cliente. Esta información ha sido obtenida después de que el cliente mandase un paquete a cada servidor. Por tanto, si se pretende descubrir el tipo de capeado NAT, se requiere comparar la IP y puerto de los dos paquetes.

Dado que la solución utiliza un bucle para atender a los datagramas y son necesarios dos paquetes para comparar, se guardarán los datos del primero en una variable externa al bucle para poder recibir el siguiente paquete. Si el tipo del segundo paquete es 1 y además proviene del otro servidor, se puede determinar que provienen del mismo cliente. En el caso de que sea otro tipo o provenga del mismo servidor que el primer paquete guardado, es posible que los datos no sean correctos por lo que se lanzará un error y eliminará los datos guardados.

Una vez tengamos los datos de ambas comunicaciones entre cliente y servidores, podemos comparar los puertos y las IPs de ambos. Desde el main, se llamará a un método externo que realizará dicha comparación y devolverá un string con el tipo de capeado NAT (estática, dinámica, pat). Con el resultado, se calcularán los porcentajes de cada tipo de NAT identificados hasta el momento en los clientes, mostrándolo por pantalla.

El tipo de filtrado NAT tiene una forma de comprobación diferente, por lo que al servidor maestro únicamente le llegará el resultado bajo el tipo 3 de paquete. Tendrá que deserializar el JSON y recuperar el valor del resultado (full cone, restricted cone, restricted port o symmetric). Una vez hecho esto, volvemos a calcular porcentajes, pero en este caso del tipo de capeado y se mostrarán por pantalla.

Como se puede leer entre líneas, esta solución no contempla la ejecución en paralelo. Las pruebas que se han realizado y que veremos más adelante se han hecho en secuencial, por lo que no ha podido darse el caso de que paquetes de diferentes clientes se intercalen. El servidor maestro no tiene la capacidad de filtrar por cliente y sería realmente complejo teniendo en cuenta que NAT puede cambiar la IP y el puerto. La mejor opción para realizar una ejecución en paralelo sería utilizar hilos que puedan responder a diferentes clientes a la vez e identificadores aleatorios con los que determinar inequívocamente los paquetes para sus comprobaciones posteriores.

4.1.4. Servidor esclavo 1

El segundo servidor que veremos es ligeramente más complejo que el resto porque recae sobre él una parte mayor del trabajo. Su función será servir como punto de paso entre el cliente y el servidor maestro (igual que el otro esclavo) pero también estará encargado de atender al inicio de la segunda prueba, comprobar el filtrado NAT.

Esta tarea hemos decidido relegarla del maestro porque sería malgasto de recursos. Cuando comienza la prueba, el cliente informa que está listo para ello al servidor esclavo 1 pero no hay razón por la que informar también al maestro. Son paquetes que se estarían mandando innecesariamente y por tanto se generaría más tráfico en la red.

Como ocurre en el resto de los programas desarrollados, el método main exclusivamente ejecuta un bucle para capturar datagramas y el resto del procesamiento se realiza en métodos externos. Se ha elegido una parte del código que muestra este suceso, y las clases que engloban los métodos.

```
while (true) {

    byte[] buffer = new byte[1024];

    DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
    socketUDP.receive(packet);

    String JSONstring = ProcesadoPaquete.procesadoPaquete(socketUDP, packet, origen:"servidorEsclavo1");

    if (!JSONstring.equals(anObject:"")) {
        ProcesadoPaquete.envioPaquete(socketUDP, direccionServidorMaestro, puertoServidorMaestro,
                JSONstring);
    }

}
```

Figura 4.2: Bucle while del servidor esclavo 1

Para explicar brevemente este fragmento de código (4.2), vemos como se crea un byte que transportará el contenido del texto hallado en los datagramas. Posteriormente, se crea un datagrama y, con el método receive, se espera a su llegada. Cuando se dispone del paquete, la clase ProcesadoPaquete (4.3) se encarga de realizar las tareas necesarias según sea tipo 1, 2, o 3. Además, podemos ver que ProcesadoPaquete incluye un método vacío y estático que envía el paquete a las partes afectadas: envíoPaquete.

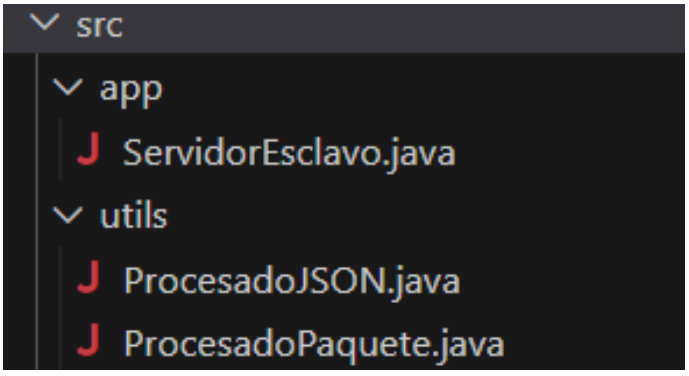

Figura 4.3: Clases del servidor esclavo 1

Haciendo referencia a lo mencionado previamente, este servidor es el más versátil porque tiene la posibilidad de enviar paquetes a los otros tres implicados en la solución. Al servidor maestro cuando le llegan mensajes de tipo 1 o 3 (para notificar el resultado de la prueba), al servidor esclavo 2 cuando le avisa del inicio de la segunda prueba y al cliente para intentar detectar su filtrado NAT mediante el envío de paquetes propios.

4.1.5. Servidor esclavo 2

Poco hay que mencionar de este servidor que no hayamos comentado antes. Es una versión simplificada del esclavo 1 que ejecuta la primera prueba de la misma forma que el otro, pero varía en la segunda. La diferencia radica en la forma de enviar paquetes al cliente para detectar el capeado. La dirección IP de este servidor no ha sido guardado en la lista de direcciones conocidas en el cliente, por lo que su única posibilidad de llegar al cliente es si tiene un capeado del tipo full-cone.

Es por eso que está preparado sólo para enviar un datagrama con un apartado en el texto, indicativo de que en efecto el cliente usa full-cone. Ni siquiera recibirá la respuesta a este paquete porque el cliente lo envía al servidor esclavo 1. Por lo tanto, la única comprobación que debe hacer el esclavo 2 es si el mensaje que ha recibido es de tipo 2, si no es así, tendrá que reenviar lo que haya obtenido al servidor maestro.

4.1.6. Cliente

El componente final de la solución y el que más tiempo ha requerido es el cliente. Consiste en un programa capaz de comunicarse con los servidores y exponer la información importante al usuario. Dado que podrá utilizarse en ordenador y móvil, ha sido necesario crear dos aplicaciones. Una con Android Studio y otra con VS Code. Las dos tienen los mismos métodos, pero se ha modificado ligeramente el código en la aplicación móvil por problemas de compatibilidad.

Aplicación en VS Code

El cliente desarrollado en Java es probablemente el programa más sencillo de todos. Esta parte se ha ideado para que sea un programa de una sola ejecución. Esto implica que, teniendo el archivo ejecutable, únicamente se requerirá un comando que realice todas las pruebas y responder a una serie de preguntas. Se trata de preguntas estándar como si va a utilizar la red de la UPV o una red externa y la solicitud de las direcciones y puertos en caso de que sea la red externa. Después de eso, obtendremos la información directamente en la consola ya que se harán las dos pruebas a la vez.

El código consiste en un archivo main con dos cláusulas try-catch para crear dos sockets aislados. Dentro de cada cláusula, además, se creará un jsonObject simplemente para definir la propiedad *tipo* y el valor que requiera la prueba. Con el primer socket se enviarán dos datagramas de tipo 1, uno a cada servidor esclavo sin ninguna información adicional. De esta forma, los servidores podrán sacar la IP y puerto de cada datagrama, que se incluyen automáticamente en el envío.

La segunda cláusula comienza enviando un paquete de tipo 2 al servidor esclavo 1 y posteriormente esperará a recibir un paquete. Esto puede resultar extraño teniendo en cuenta que los servidores mandan en total 3 paquetes, pero se puede explicar. El paquete más relevante es el primero que llegue porque indicará el tipo de NAT que utilice el cliente. El servidor esclavo 1 esperará un tiempo a que el segundo esclavo envíe su paquete, que probablemente no llegue porque muy pocas NAT utilizan full-cone. Después enviará los suyos, que como mínimo tiene la seguridad de llegar uno (el que accede por restricted-port). Cuando ha recibido uno, el cliente ya puede responder diciendo cual ha sido. Esto lo hace mediante un último envío, de tipo 3, con la información que indica el tipo de NAT.

Aplicación móvil

Esta es sin duda la parte más compleja del desarrollo, no solo por la diferencia en el código sino también por la multitud de archivos que debemos tener en cuenta para crear la aplicación. Como ejemplo, por cada pantalla que requiera, como mínimo hará falta un archivo java y otro xml, además de indicar en otro archivo de navegación cómo se accederá a dicha pantalla.

Antes de explicar las pantallas, hay que aclarar en qué consisten los archivos java que las implementan. Android Studio utiliza un componente llamado Activity, que es una clase que permite crear instancias Activity según el estado de la aplicación. La idea es que en el móvil no siempre se accede a la aplicación desde el main, ya que existen aplicaciones que pueden llamar a otras y acceder a partes del código posteriores directamente. Un ejemplo muy claro sería una aplicación de correo electrónico. Existen redes sociales que llaman a la aplicación para redactar un correo y por lo tanto se inicia en la pantalla de redactar correo en vez del main.

Una vez entendido esto, podemos desarrollar el funcionamiento de la aplicación. Previo a las pantallas que tiene, existe una clase java llamada MainActivity que se ejecuta en primer lugar [16]. Esta clase no sirve como pantalla inicial, sino que se encarga de inicializar los recursos, entre ellos se encuentran métodos que usaremos para lanzar las pruebas que nos incumben.

En cuanto a archivos xml relevantes, temenos *AndroidManifest.xml*, *nav_graph.xml* y *strings.xml*. El primero describe información esencial de la aplicación para las herramientas de creación de Android, es decir, se declaran cosas como el nombre de la aplicación, los permisos que necesita, o las funciones hardware y software. El segundo, es un archivo de recursos que contiene todos los destinos y acciones. El gráfico representa todas las rutas de navegación de la app. El último, como su nombre indica, contiene la lista de strings usados en la aplicación. Desde el título de cada pantalla hasta el contenido de los botones.

Como curiosidad, durante la ejecución inicial de la aplicación obtuvimos un error de ejecución, EPERM (Operation not permitted), y el motivo era falta de permisos de acceso a Internet. Por suerte, este problema ya estaba solucionado y la forma de evitarlo era incluir unas líneas de código en el archivo *AndroidManifest.xml* [17].

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Figura 4.4: Permisos de acceso a internet en Android Studio

La aplicación consta de tres pantallas: una en la que se explica en qué consiste el proyecto y da acceso a las pruebas en sí y otras dos que permiten ejecutar las pruebas mediante botones e informa de los resultados. Pero todas tienen en común que requieren dos archivos para su ejecución, uno de tipo java y otro xml. El primero crea la vista y los objetos que contiene, mientras que el segundo define la estructura de la interfaz.

En cuanto a diferencias con la aplicación de ordenador (teniendo en cuenta solamente la parte de pruebas), hay fragmentos de código que se han debido incluir para no producir errores de ejecución. Un ejemplo es cierta cláusula que incluimos para permitir operaciones de red en los métodos desarrollados desde el main [18].

```
if (android.os.Build.VERSION.SDK_INT > 9) {
    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build()
    StrictMode.setThreadPolicy(policy);
```

Figura 4.5: Cláusula para cambiar las políticas de red

Por último, dada la naturaleza de la aplicación móvil, no era posible informar al usuario mediante la terminal. Así que se han utilizado diálogos de alerta que informan de los resultados y ventanas emergentes simples que informan de la finalización de las pruebas. Simplemente hay que crear los objetos de alerta (AlertDialog) o informativos (Toast) y darle los datos necesarios para que los exponga. Este es el motivo por el que no se ejecutan las dos pruebas a la vez, se ha elegido dar tiempo al usuario de ver el resultado individualmente. Además, facilita la comprensión del proyecto.

4.2 Casos de uso

Esta parte tratará de hacer una demostración visual de la interfaz de la aplicación móvil mediante capturas de pantalla en el emulador que proporciona Android Studio. Se debe tener en cuenta que también funciona de la misma forma en otros dispositivos móviles. Como se ha expuesto antes, la aplicación consta de tres pantallas: Inicio, Red UPV y Red externa. Ahora procederemos a explicarlas.

4.2.1. Pantalla de inicio

La pantalla de inicio es obviamente la que aparece cuando se abre la aplicación. Lo que salta a la vista es el texto introductorio y dos botones. En el texto no solo se informa al usuario de las funciones de ambos botones como veremos en la imagen siguiente 4.6, sino que también se explica el motivo de la aplicación.

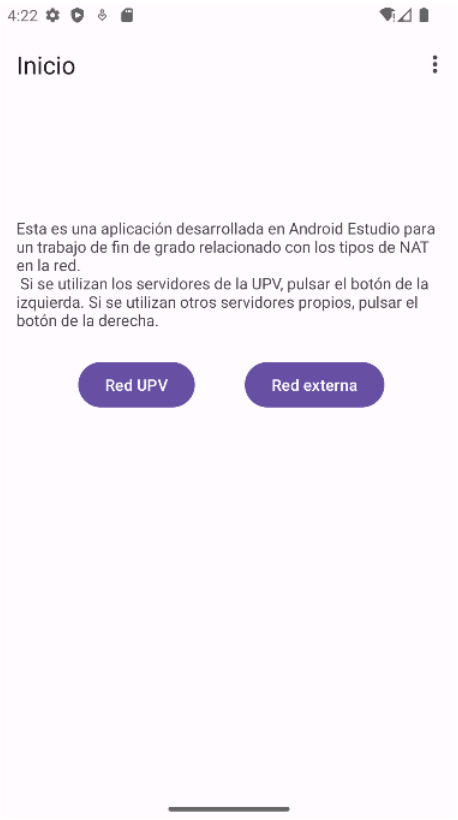

Figura 4.6: Inicio de la aplicación

Directamente desde esta pantalla se puede acceder a las otras dos que veremos, según el botón que se pulse. Empezaremos por la pantalla de la UPV pulsando el botón de la izquierda, como indica el texto del mismo.

4.2.2. Pantalla de la red UPV

Cuando accedemos a esta pantalla 4.7, podemos ver una distribución parecida a la anterior. Un texto explicativo y dos botones que permiten ejecutar las dos pruebas que se han explicado varias veces en el transcurso del documento.

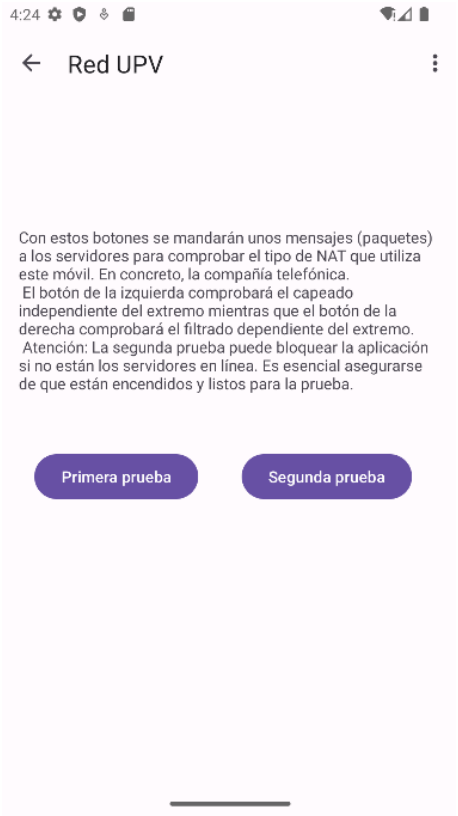

Figura 4.7: Pantalla red UPV

Es importante destacar la simplicidad de esta pantalla porque una de las opciones que podría tener esta solución es implantarla en empresas interesadas en ello. Por esto, es conveniente que sea sencillo, los empleados puedan acceder a ella rápidamente y sea intuitivo.

Vamos a utilizar la red externa para realizar las pruebas principalmente ya que tiene más detalles interesantes y porque se ve exactamente lo mismo en los resultados, a diferencia de las IPs. Pero para que se entienda a qué nos referimos con las direcciones, vemos la respuesta que se envía pulsando el botón de la izquierda 4.8.

Figura 4.8: Resultado de la primera prueba, red UPV

La ventana superior es un AlertDialog con la opción de pulsar el botón OK después de leer el texto que explica qué se ha hecho en la prueba. Además, hay un segundo comentario abajo, que únicamente está unos segundos y desaparece automáticamente. Con este se pretende informar de que ha terminado la ejecución sin errores.

4.2.3. Pantalla de la red externa

Para finalizar con los casos de uso, veremos la última pantalla, red externa, a la que se accede pulsando el botón de la derecha en el inicio (4.6).

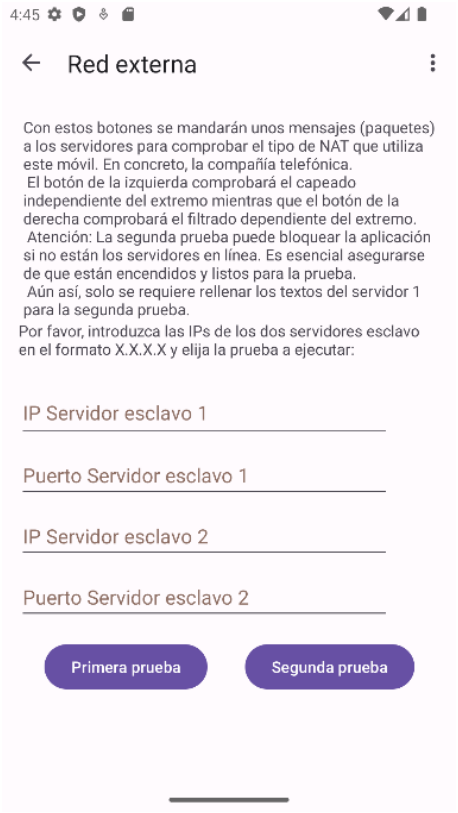

Figura 4.9: Pantalla red externa

Las diferencias son notables. En la parte superior, vemos un texto más amplio que el resto, motivado por la necesidad de explicar cómo proceder con la parte inferior. La parte inferior son cuatro cajas de texto editables en las que habrá que introducir la información solicitada por cada una. Los requisitos de cada caja se piden mediante palabras subyacentes que desaparecen cuando se empieza a escribir en ellas. Cada campo solicita un dato esencial para ejecutar las pruebas, la IP y el puerto de cada servidor.

Aunque se explica cuál es el formato para escribir en cada campo, es normal que los usuarios se equivoquen de vez en cuando. Por este motivo, se han pensado todos los errores que podrían cometerse al escribir en estas cajas y hemos creado mensajes de error que facilitan al usuario a reconducir la situación. En las imágenes siguientes veremos los errores más comunes y explicaremos cómo se advierte al usuario.

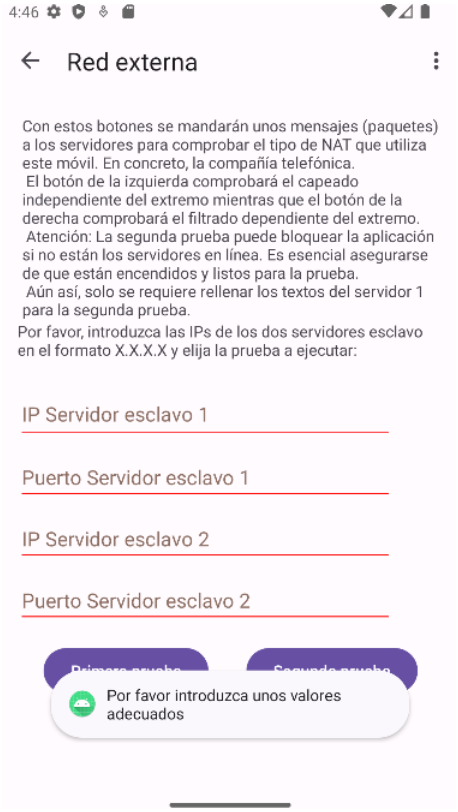

Figura 4.10: Textos vacíos en la red externa

Este primer caso es el más simple de todos, no se han introducido datos en las cajas, y muestra claramente cómo se actúa frente a los errores 4.10. En el texto superior se advierte de que no es necesario poner datos en los dos últimos campos, pero únicamente si se ejecuta la segunda prueba. En este caso se ha pulsado al primer botón por lo que todos los campos son erróneos. En la parte más baja de la pantalla aparece un *popup* solicitando nuevos valores que desaparece tras unos segundos para no entorpecer. Además, cada caja de texto que tenga un error cambiará de color a rojo. Es decir, la parte baja de la caja que tiene una línea negra normalmente, cuya utilidad es delimitar cada caja, cambiará a rojo. Esto sigue el principio de visibilidad del estado del sistema, uno de los diez principios de usabilidad de Jakob Nielsen. Más adelante, explicaremos cómo esta aplicación ha intentado abordar estos principios.

También hay errores que aparecen al escribir en los campos por ser datos inválidos. En la próxima imagen (4.11) se expondrán los cuatro fallos que hemos tenido en cuenta.

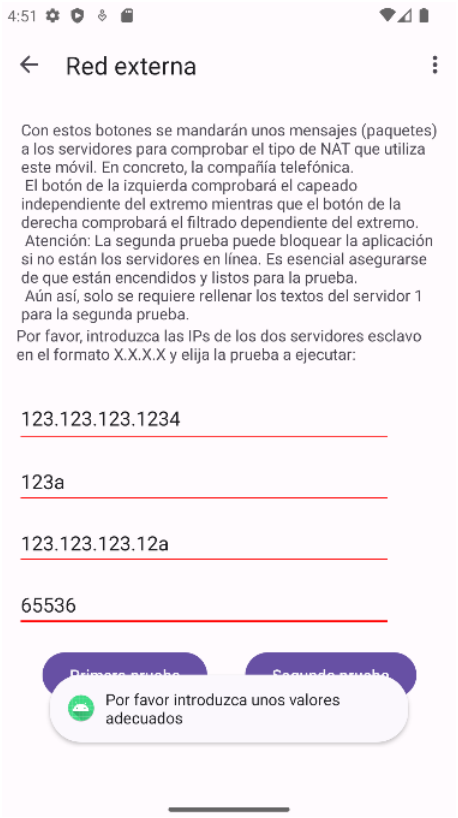

Figura 4.11: Valores incorrectos en la red externa

Por supuesto, se avisará de la misma forma que antes y no se ejecutará la prueba. En el primer campo, vemos que la última parte de la dirección tiene cuatro números en vez de 3, haciendo que sea un valor superior a 255. El segundo campo muestra un puerto con una letra, esto es simplemente un fallo de compatibilidad y lanzaría un NumberFormatException en la aplicación. Lo mismo pasa con el tercer campo, donde aparece una letra al final del string y no es una IP válida. Por último, también se ha tenido en cuenta el rango de puertos existentes, que abarca del 0 al 65535 por lo que poner un número más alto no está acogido.

Para advertir al usuario de un posible bloqueo de la aplicación, creemos que no es suficiente con una línea en el texto superior ya que podría ignorarse. Por eso se ha incluido otro cuadro de alerta que aparece antes de ejecutar la segunda prueba 4.12, causante del bloqueo si no están los servidores en línea. Es un cuadro simple que permite retroceder si se ha pulsado sin querer, además de dar al usuario una segunda oportunidad de comprobar si los servidores están en línea.

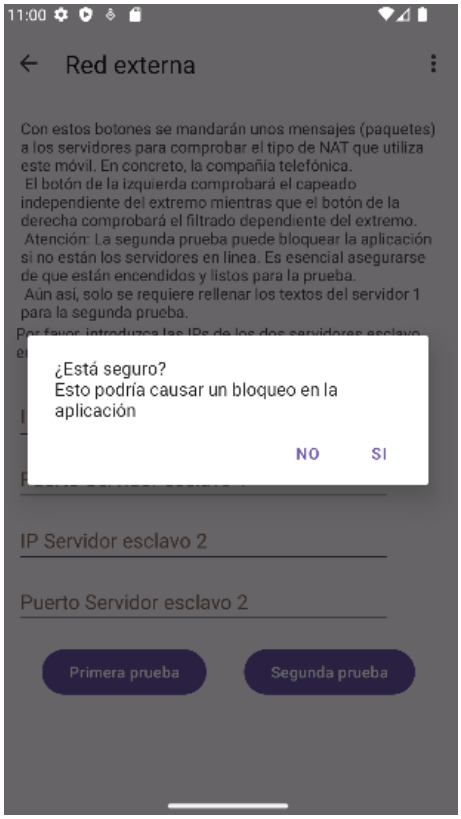

Figura 4.12: Cuadro de alerta en la red externa

Finalmente, veremos qué resultado tiene ejecutar la segunda prueba, el botón de la derecha. Es similar al botón de la izquierda en cuanto a resultado, aparece un texto AlertDialog con un botón de *OK* para confirmar que se ha entendido y un texto inferior para informar de que no ha habido errores. Esta vez, el texto inferior primero indicará que se está intentando conectar con el servidor ya que si hubiera retrasos en los paquetes podría llevar un rato y el usuario necesita saber que el proceso no se ha detenido 4.13. En este caso, el resultado ha sido inmediato y por eso aparecen las dos ventanas a la vez. Pero cuando ya ha terminado, aparecerá el mismo desplegable que en la primera prueba, avisando de que se ha ejecutado sin errores 4.14.

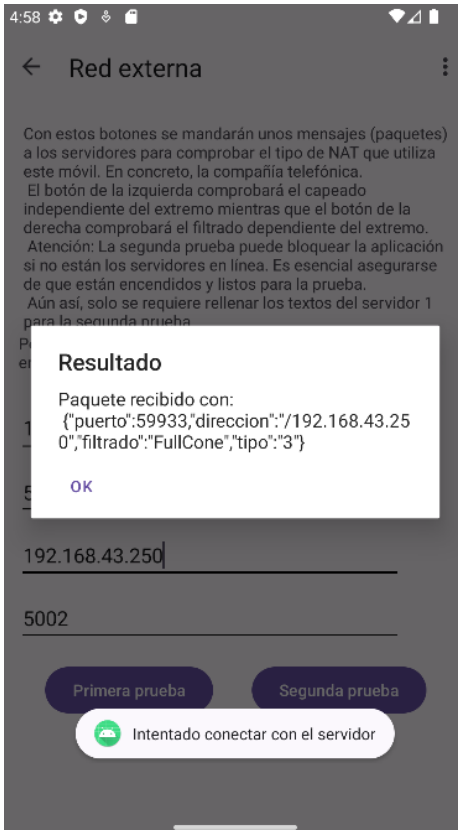

Figura 4.13: Segunda prueba en proceso, red externa

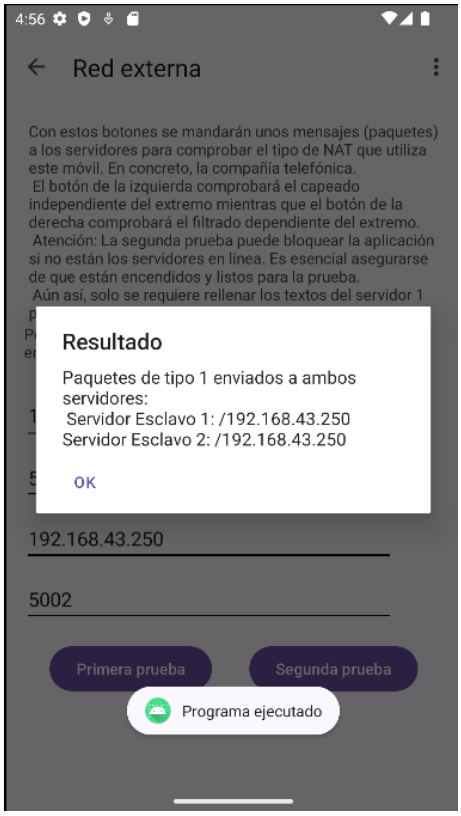

Figura 4.14: Resultado de la segunda prueba, red externa

Algo bueno que se ha mantenido, es no borrar los campos una vez ha terminado la prueba. Aunque en algunas situaciones puede ser contraproducente, hemos pensado que es más probable que un usuario utilice los mismos servidores. Además, también se mantiene el texto en caso de error, permitiendo al usuario rectificar sobre lo que ya ha escrito y que sea más fácil de reconocer los fallos.

4.2.4. Principios de usabilidad

La usabilidad en el diseño consiste en facilitar al usuario la utilización de una interfaz de la forma más fácil e intuitiva. Es la facilidad con las que las personas interactúan con una herramienta con el fin de alcanzar un objetivo concreto. Los principios de usabilidad, o principios heurísticos, están basados en reglas generales son una serie de 10 ideales y fundamentos que permiten crear productos con un mayor grado de acogida entre los usuarios debido a que se basan en sus necesidades y en su comportamiento.

Comenzaremos por el principio mencionado antes, visibilidad del estado del sistema. Con las ventanas emergentes y los cambios de color en los campos, se consigue tener informado al usuario de lo que está pasando en la aplicación.

Relacionado con el principio anterior, se le da control y libertad del usuario a la hora de cometer errores ya que se muestra el lugar en el que ha ocurrido y se avisa del formato en el texto superior. Avisar del formato, podría acercarse más al principio de prevención de errores, también tenido en cuenta. Cuando se pide al usuario confirmar el inicio de la segunda prueba se hace uso de este principio. Al no borrarse la información que han introducido, también se le da la oportunidad al usuario de subsanar el error.

Otro principio que se ha seguido es la consistencia en la navegación con botones parecidos en todas las pantallas y la misma disposición de todos los elementos. Así, el usuario podrá encontrar los elementos que necesita más rápido y será todo intuitivo.

Por último, aunque no es tan obvio por la cantidad de texto, se intenta buscar un diseño minimalista: un texto, dos botones y en el caso de la red externa los campos necesarios.

CAPÍTULO 5
Pruebas de validación

Las pruebas que enseñaremos constan de dos partes. Por un lado, demostraremos el funcionamiento de la solución con Wireshark, una aplicación consolidada que expone sin lugar a duda la validez de la aplicación. Por otro lado, presentaremos los resultados obtenidos con los mensajes que proporciona la aplicación en sí. Se verán los outputs en terminal de cada componente, incluso los del cliente creado para ordenador.

5.1 Paquetes UDP

Esta sección está dedicada a mostrar el intercambio de paquetes capturados con la aplicación Wireshark. Desde los paquetes enviados por el cliente hasta los que se envían entre servidores. La idea es poder visualizar el contenido, puertos de envío e IPs de una forma clara y definitiva.

Además, es importante entender que las capturas en Wireshark se realizan sobre una interfaz únicamente. Esto quiere decir que no se podrán ver en una sola captura todos los paquetes enviados por cada dispositivo ya que habrán diferentes interfaces involucradas. Por ejemplo, la más usada será Wi-Fi pero también veremos la interfaz loopback.

Figura 5.1: Aplicación Wireshark

Los paquetes que se mirarán más a fondo serán los capturados en la red externa a la UPV ya que tiene ciertas particularidades que la hacen más compleja y por lo tanto tiene más área que cubrir. Pero también se mostrará unas capturas con el resultado del tipo de NAT en la red de la UPV.

5.1.1. Red externa

Para estas pruebas se utiliza el esquema mostrado en la imagen 3.2, por lo que las IPs serán muy diferentes entre sí teniendo en cuenta que algunas son de teléfonos móviles (clientes). Veremos las diferentes interfaces involucradas, entre ellas estarán las interfaces Wi-Fi de ambos servidores esclavo y la interfaz loopback del servidor esclavo 1. Esta última es necesaria porque el servidor esclavo 1 y el maestro fueron desplegados en la misma máquina.

Interfaz Wi-Fi (servidor esclavo 1)

La primera imagen que se va a analizar es el envío al completo de los mensajes para comprobar ambos tipos de NAT, capeado y filtrado, entre el cliente y el servidor esclavo 1 en la interfaz Wi-Fi del servidor 5.2.

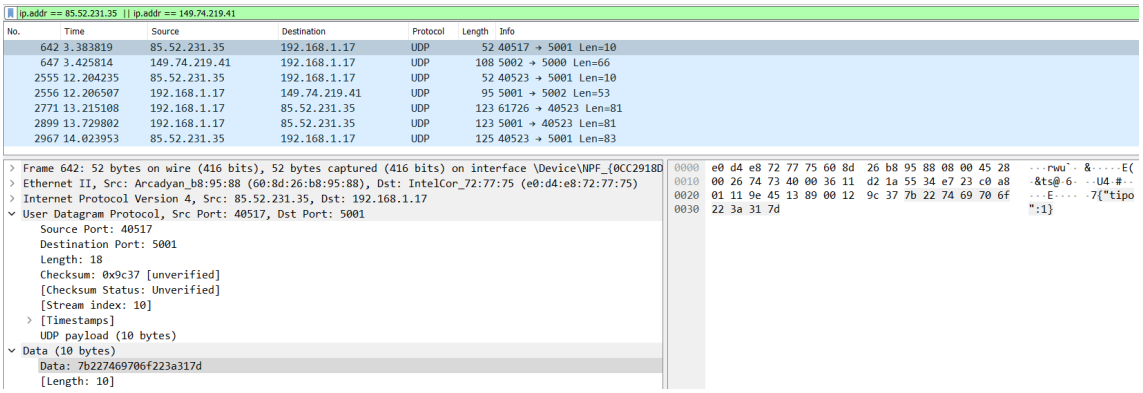

Figura 5.2: Cliente y Esclavo 1, primer paquete

Como se ha comentado ya, el primer paquete es enviado por el cliente, en este caso un teléfono móvil con IP 85.52.231.35, hacia el servidor con IP 192.168.1.17. Se ha elegido este paquete en concreto porque puede verse la información que requiere el servidor maestro para comprobar el tipo de capeado NAT y el contenido del envío que identifica a este paquete como tipo 1 en formato JSON.

La dirección del servidor expuesta en esta captura no es la dirección que el cliente ve, es la que tiene en la interfaz Wi-Fi. La dirección real es 149.74.24.41. En esta misma imagen se puede observar la IP real del servidor esclavo 2 cuando el servidor actual le envía un datagrama.

En esta misma captura de la red encontramos otro paquete muy interesante, el enviado por este cliente para notificar al servidor 1 que puede comenzar la segunda prueba (detección de filtrado NAT) 5.3. Es esencial que sea el cliente quien empiece la comunicación únicamente con el servidor 1 ya que para comprobar si utiliza un tipo de NAT full-cone no debe comunicar con el servidor 2 previamente.

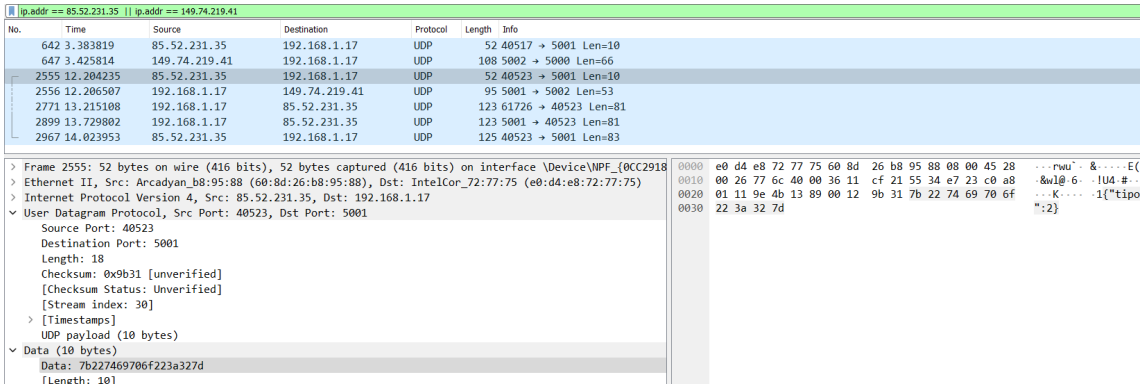

Figura 5.3: Cliente y Esclavo 1, segundo paquete

Cuando el servidor 1 recibe este datagrama, inmediatamente notifica al otro servidor esclavo la IP y puerto que tiene el cliente abiertos para esta prueba. Una vez hecho esto, procede a realizar sus propios intentos de comunicación con el cliente.

En el caso más restrictivo, recibirá únicamente la notificación de que el tipo de NAT es port-restricted y todos los demás paquetes serán rechazados. Es decir, el NAT del cliente solo deja pasar mensajes de dispositivos con los que el cliente haya contactado previamente y siempre que mantengan la misma IP y puerto.

Finalmente podemos ver en la imagen inferior (5.4) como el cliente hace saber al servidor esclavo 1 que ha recibido únicamente la notificación mencionada en el párrafo anterior. Lo hace poniendo en el texto en JSON un apartado con el nombre *filtrado* y el valor *Restricted Port*, además de indicar que es un mensaje de tipo 3 (será enviado al servidor maestro).

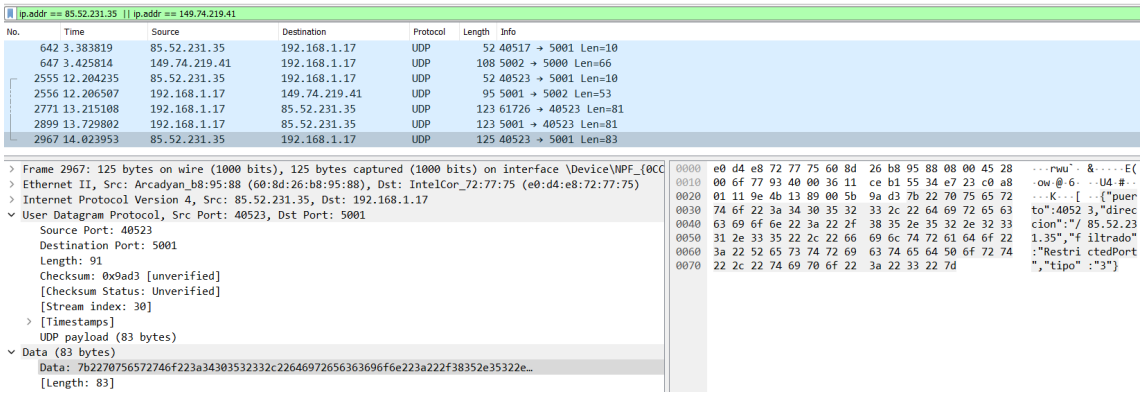

Figura 5.4: Cliente y Esclavo 1, último paquete

Interfaz Wi-Fi (servidor esclavo 2)

Aunque en esta interfaz hay menos cantidad de paquetes, son también bastante relevantes. Se sigue un esquema parecido al otro servidor, primero recibe un paquete del cliente con un mensaje de tipo 1 que debe comunicar al servidor maestro y después envía un paquete al cliente para intentar detectar el filtrado NAT.

En el primer paquete (5.5) podemos ver que ha cambiado el puerto, pero la IP no, esto implica que estamos ante un capeado pat. Como detalle a destacar, vemos que el puerto al que envía el datagrama el cliente es 5002, que es el que el servidor tiene abierto para estas transacciones.

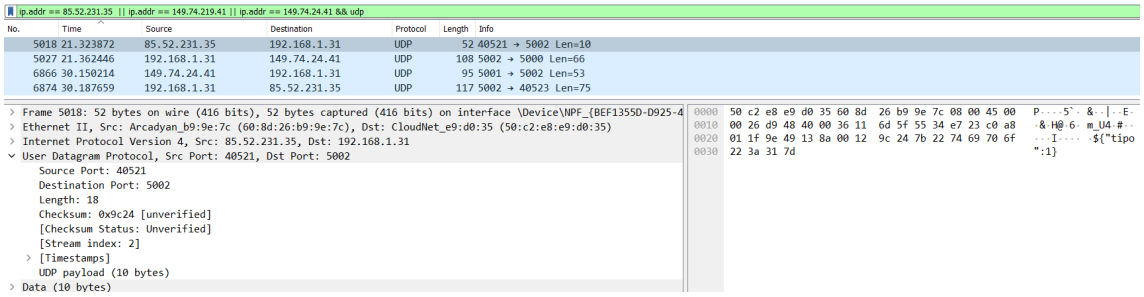

Figura 5.5: Cliente y Esclavo 2, primer paquete

También hay otro paquete interesante, el tercero. Es de recibo mencionar que el emisor de ese paquete, que tiene IP 149.74.24.41 es el servidor esclavo 1. Es un mensaje de tipo 2, es decir, que se quiere informar al servidor esclavo 2 de que puede mandar su prueba final al cliente. Como información adjunta, indica cual es la ip y puerto del cliente (5.6).

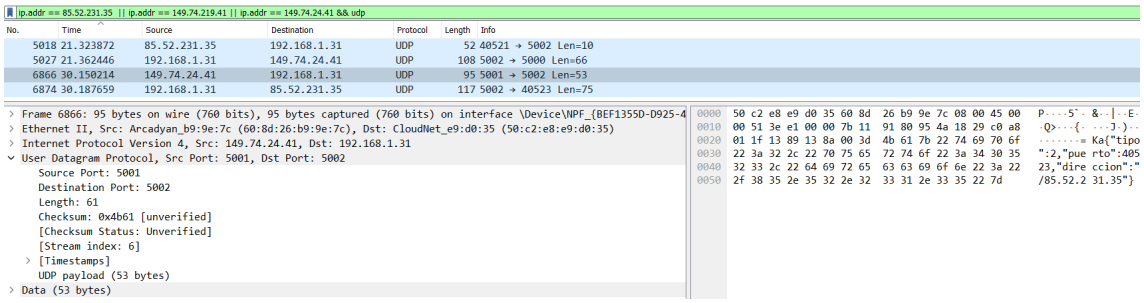

Figura 5.6: Cliente y Esclavo 2, tercer paquete

Finalmente, el esclavo 2 envía un paquete al cliente con el fin de detectar si utiliza NAT full-cone, teniendo en cuenta que el cliente no ha contactado previamente con él. No es contradictorio el hecho de que hayamos visto antes un paquete mandado por el cliente porque se hace desde un socket diferente. El puerto abierto por el cliente cambia una vez se crea otro socket y por tanto la referencia guardada por NAT no se reutiliza. Podemos ver el contenido del paquete en la captura e identificar el valor full-cone (5.7).

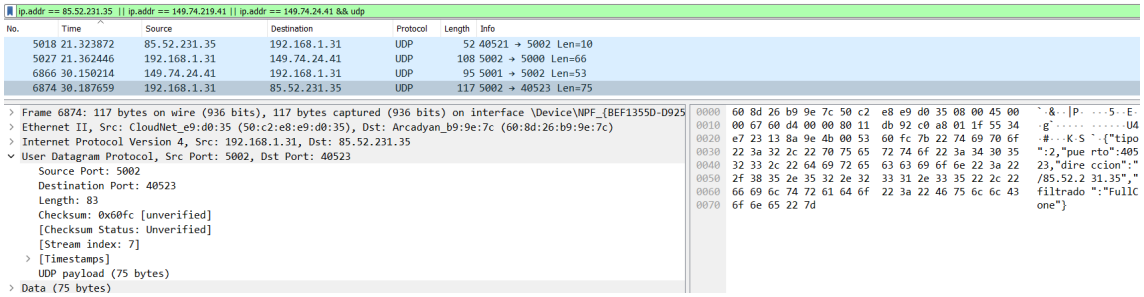

Figura 5.7: Cliente y Esclavo 2, último paquete

Interfaz loopback (servidor esclavo 1)

Esta parte es la última que necesitamos investigar de nuestra solución, sabiendo que se podría cambiar por otra interfaz Wi-Fi si se dispusiera de otro endpoint para el servidor maestro. Vamos a mostrar una imagen con dos paquetes, el primero equivale al visto en la imagen 5.2 y por tanto no es necesario explicarlo. Pero el segundo (5.8) es la última comunicación realizada en nuestra solución. es la que indica el tipo definitivo de

capeado NAT. Como podemos ver, proporciona toda la información del cliente para que el servidor maestro pueda imprimirla por pantalla después.

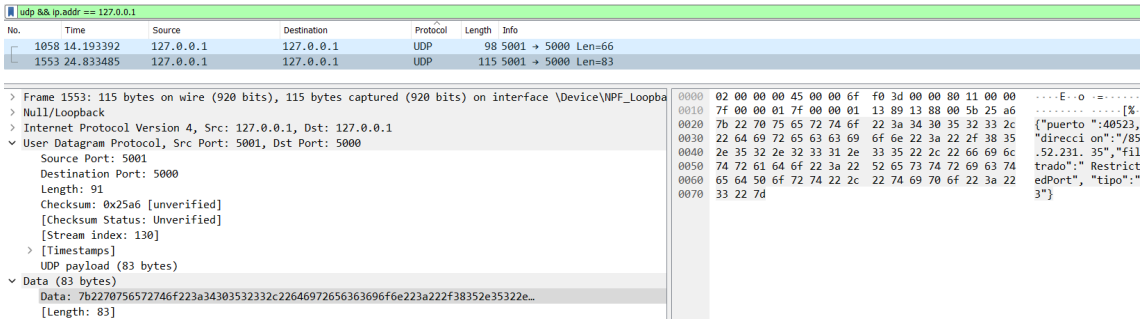

Figura 5.8: Esclavo 1 y maestro, segundo paquete

5.1.2. Red UPV

Aunque se tratan de los mismos casos que antes, en esta sección mostraremos algunas diferencias que hemos encontrado en la red de la UPV y unas situaciones particulares que podrían ocurrir en la ejecución de la solución.

Es relevante destacar que en esta configuración, todos los dispositivos sometidos a evaluación en la red exhibieron resultados consistentes en relación a los tipos de Traducción de Direcciones de Red (NAT). En la primera prueba, se observó que la totalidad de los dispositivos se vieron afectados por la implementación de NAT estática, lo que resultó en la uniformidad de las direcciones IP y los puertos en sus conexiones. En el contexto de la segunda prueba, se constató una predominancia abrumadora de dispositivos configurados con NAT de cono restringido.

Como es obvio, las IPs han cambiado en este contexto. Las que veremos son del tipo: 158.42.X.X, que pertenecen a las máquinas del laboratorio; o 10.236.43.X, para los dispositivos conectados al Wi-Fi. Normalmente, después del 42 encontramos un 180 en las máquinas del laboratorio, pero el siguiente caso es una IP diferente.

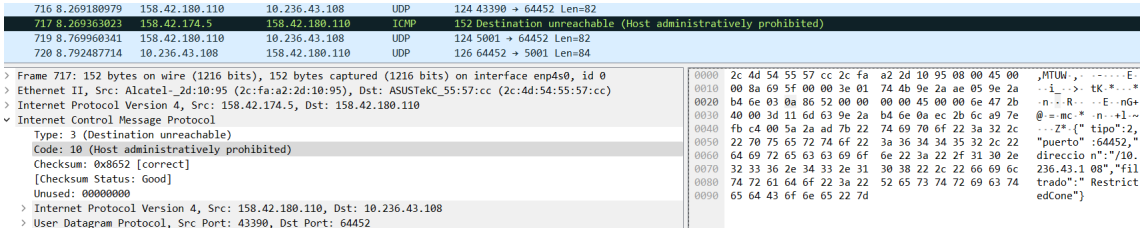

Figura 5.9: Paquete icmp del router

En la imagen anterior (5.9), vemos un paquete ICMP enviado desde la dirección 158.42.174.5 al servidor esclavo 2 en la interfaz Wi-Fi del servidor. El datagrama que lo precede es uno enviado por el servidor, que pretende descubrir si el cliente está detrás de un NAT full-cone. Como consecuencia, lo que probablemente sea el router (aunque podría ser el firewall también) responde con este paquete. La intención es clara, informar al dispositivo con IP 158.42.180.112 que su paquete (cuya información aparece a la derecha) ha sido despreciado.

En el mensaje de error aparece: Type 3: (Destination unreachable) y Code 10: (Host administratively prohibited). Como NAT está establecido a port-restricted, el router no permite que el paquete llegue a su destino y responde mediante el protocolo ICMP. En

ocasiones, no hay respuesta (es el caso de la red externa anterior) pero esto es justo lo que se pretende comprobar en la solución. Además, esto ocurre igual en el caso del servidor esclavo 1 cuando envía su primer paquete (teniendo en cuenta que NAT está establecido a port-restricted).

Por último, veremos un ejemplo de paquetes llegando a destiempo y explicaremos por qué no deberían producirse errores si esto ocurre. Antes de nada, hay que entender que para que funcione lo que vamos a comentar se debe tratar de una dupla de pruebas provenientes del mismo cliente. Es decir, pueden intercalarse paquetes que formen parte de las dos pruebas que se realizan, detectar el filtrado y el capeado, siempre y cuando sea del mismo cliente.

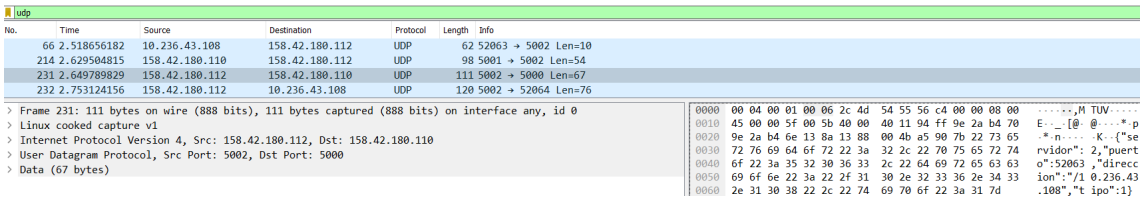

Figura 5.10: Paquetes a destiempo

Aquí, simplemente ha llegado antes un paquete del servidor esclavo 1 al servidor esclavo 2, sin que el segundo haya podido enviar el resultado de la primera prueba al servidor maestro. Por suerte, para el servidor maestro no altera la ejecución. Pero, aunque cambiara, que el único caso sería llegando primero un paquete de tipo 1, después otro de tipo 3 y finalmente el segundo de tipo 1, no daría problemas. Esto se debe a que primero se comprueba el tipo del mensaje y después la proveniencia del mismo. Si fuese al revés, sería un error de diseño porque podría llegar un paquete de tipo 3 del servidor esclavo 1 y después otro de tipo 1 del mismo servidor y daría error, cosa que es insostenible.

5.2 Capturas de terminal

Dado que en las anteriores capturas hemos visto principalmente el comportamiento de las máquinas en la red, en esta sección nos centraremos en el contenido de los paquetes y la interacción con el usuario en la terminal.

Primera prueba

Para comenzar, veremos qué información ofrece la solución una vez se ha ejecutado la primera prueba. Los tres mensajes que aparecen se imprimen prácticamente a la vez sin retrasos en la red.

En los dos servidores esclavo se muestra lo que reciben del cliente, incluyendo la información que saca del datagrama (puerto e IP) y lo que devuelve al maestro (imágenes 5.11 y 5.12. La única diferencia es la identificación del servidor esclavo, así el maestro sabe que puede comparar los dos paquetes porque vienen de diferentes servidores.

```
Paquete recibido en servidorEsclavo1 con: {"tipo":1}
Puerto: 20818
Direccion: /84.78.250.74
Paquete enviado con: {"servidor":1,"puerto":20818,"direccion":"/84.78.250.74","tipo":1}
```

Figura 5.11: Información del servidor esclavo 1 sobre la primera prueba

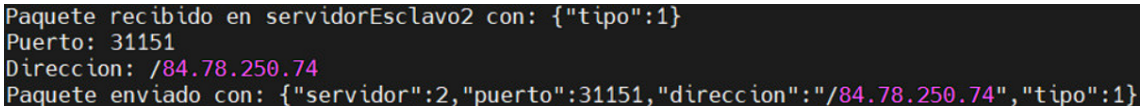

Figura 5.12: Información del servidor esclavo 2 sobre la primera prueba

El servidor maestro, como es lógico, obtendrá la información enviada desde los dos servidores esclavo en el formato de las imágenes anteriores. Con estos datos únicamente queda comparar las IPs y puertos para determinar el tipo de NAT. Cuando resuelve esta duda, el servidor maestro imprime el porcentaje de cada tipo obtenido hasta el momento 5.13.

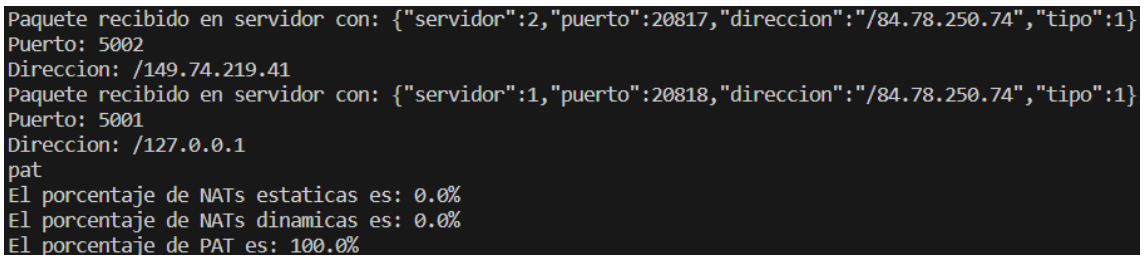

Figura 5.13: Información del servidor maestro sobre la primera prueba

Segunda prueba

Ya sabemos cómo comienza el segundo test, con la solicitud del cliente al servidor esclavo 1 para realizar la segunda prueba. Este servidor avisa al servidor esclavo 2 que ha comenzado la prueba con el primer paquete enviado en la imagen 5.14. El resto de los envíos son intentos del servidor esclavo 1 de descubrir el tipo de capeado NAT.

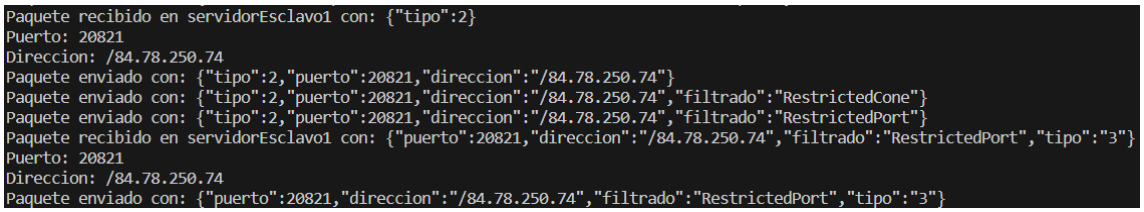

Figura 5.14: Información del servidor esclavo 1 sobre la segunda prueba

Aun así, merece la pena ver qué información muestran los otros dos servidores cuando se ejecutan. El esclavo 2 muestra el paquete mandado por el servidor 1 y el que envía para intentar descubrir un filtrado Full-Cone 5.15. El maestro enseña la confirmación final y los porcentajes de cada NAT analizada 5.16.

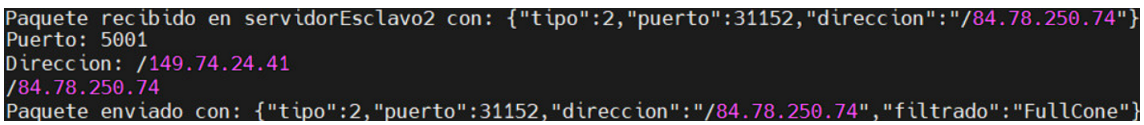

Figura 5.15: Información del servidor esclavo 2 sobre la segunda prueba

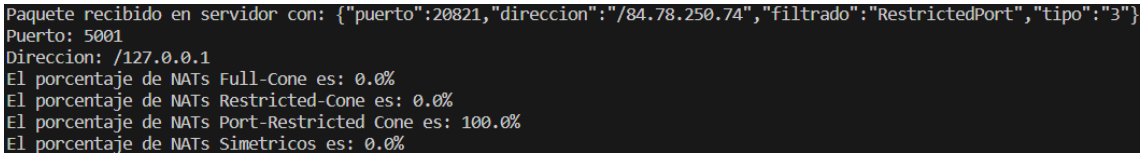

Figura 5.16: Información del servidor maestro sobre la segunda prueba

5.3 Estadísticas obtenidas

Una vez terminadas las pruebas con varios usuarios, tenemos suficientes datos para exponer los resultados y dar una visión global de la situación actual en cuanto a tipos de NAT en la red. El enfoque empleado para la recopilación de datos involucró la operación continua de los servidores durante un período determinado, durante el cual los usuarios remitieron sus paquetes. El servidor principal registró con precisión los porcentajes correspondientes a cada categoría de Traducción de Direcciones de Red (NAT).

Los participantes son individuos utilzando dispositivos móviles que acceden a la Internet a través de conexiones provistas por operadoras de red móvil, o a través de conexiones a sus redes privadas a través de enrutadores. Se considerarán tanto el origen de los paquetes como el proveedor de servicios móviles, ya que las disparidades serán pronunciadas entre redes privadas y conexiones móviles. Es crucial señalar que el tamaño de la muestra comprende catorce usuarios, abarcando una gama significativa de datos cualitativos entre ellos.

Para comenzar, compararemos las diferencias entre los dispositivos con datos móviles y los dispositivos detrás de un router privado. Según la gráfica inferior 5.17, en las pruebas realizadas podemos observar como la totalidad de los dispositivos detrás de un router (cuatro para ser exactos) han utilizado NAT estática. En cambio, la traducción de direcciones de red de sobrecarga (o PAT) es lo más común entre dispositivos con acceso a datos móviles, habiendo un único móvil que ha utilizado NAT estática en esta situación mientras que los otros nueve son de sobrecarga.

El motivo detrás del uso de NAT estática en redes privadas es probablemente la configuración predeterminada de los routers cuando los instalan las compañías telefónicas, ya que deben establecer una NAT estática en la mayoría de los casos (teniendo en cuenta que los usuarios partícipes de esta prueba no han modificado el tipo de NAT).

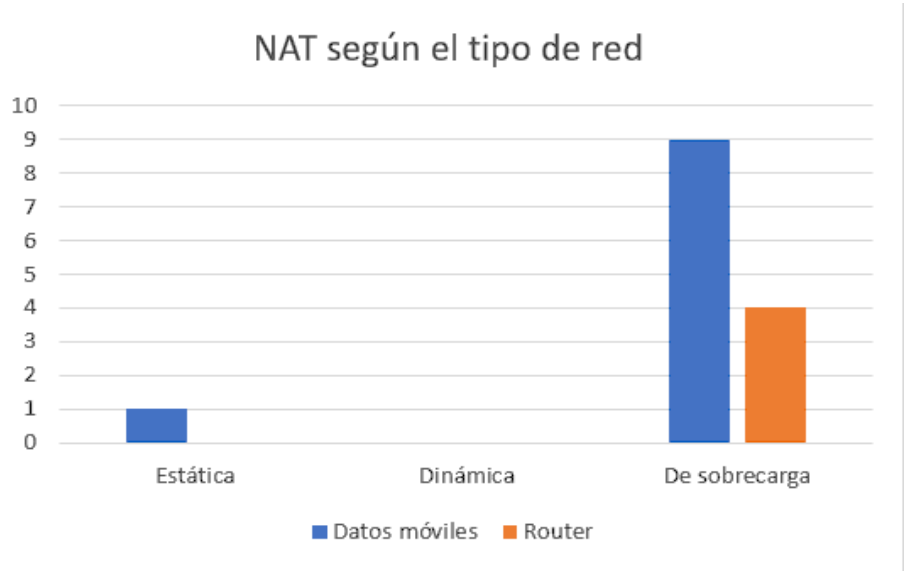

Figura 5.17: NAT independiente del extremo según el tipo de red

Conociendo esta información, podemos avanzar al siguiente gráfico estadístico obtenido al finalizar estas pruebas 5.18. En este caso, se trata de una comparativa entre las compañías telefónicas y el tipo de NAT independiente del extremo. Como ya hemos comprobado antes, la traducción de direcciones de red depende más del tipo de red que de la compañía utilizada, pero existe un caso en el que sí ha dependido de la compañía.

Se trata de Yoigo, donde hemos obtenido NAT estática para un dispositivo que utilizaba datos móviles.

El caso expuesto del dispositivo móvil que ha utilizado NAT estática es también motivo de estudio. Sabemos que su ubicación en el momento de la prueba era una ciudad pequeña comparada con el resto de usuarios, que se encontraban en Valencia. Una compañía telefónica podría tener varios motivos para implementar NAT estática en dispositivos móviles en una ciudad pequeña: La escasez de direcciones IP públicas asignadas a esa ciudad, el ahorro de costos, la gestión de tráfico...

Con estos datos podemos determinar que no todas las compañías utilizarán el mismo tipo de NAT independiente del extremo, cada una utilizará la que mejor se adapte a sus necesidades. Esto se puede aplicar también a la red de la UPV, donde es más factible usar una configuración más manejable como es la NAT estática.

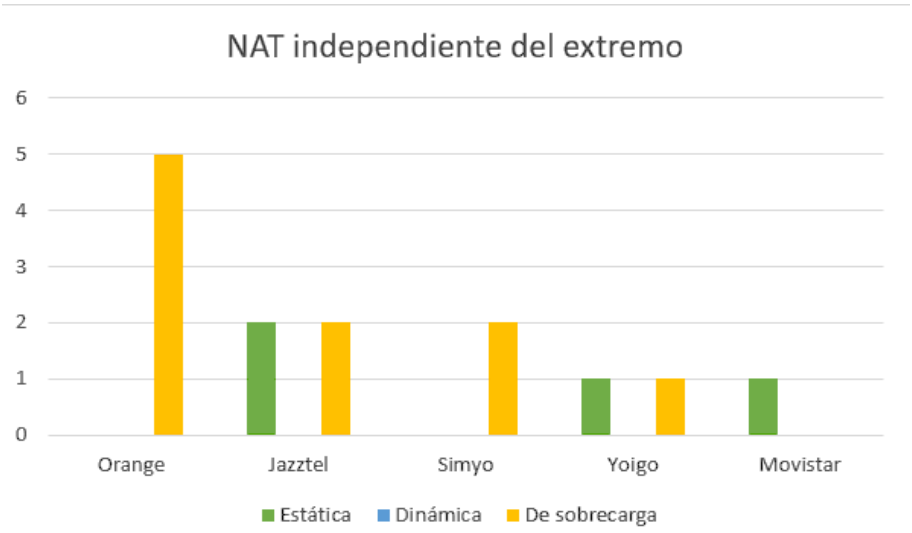

Figura 5.18: NAT independiente del extremo según la compañía

La traducción de direcciones de red independiente del extremo no influye en la seguridad de la red tanto como la dependiente, por eso los resultados de la tabla anterior son ligeramente variables. En cambio, los resultados obtenidos tras analizar el tipo de NAT dependiente del extremo son bastante indicativos de que la mejor elección para este caso es el cono de puerto restringido. Todos los dispositivos estudiados convergen en este tipo de NAT, como podemos ver en la siguiente gráfica 5.19:

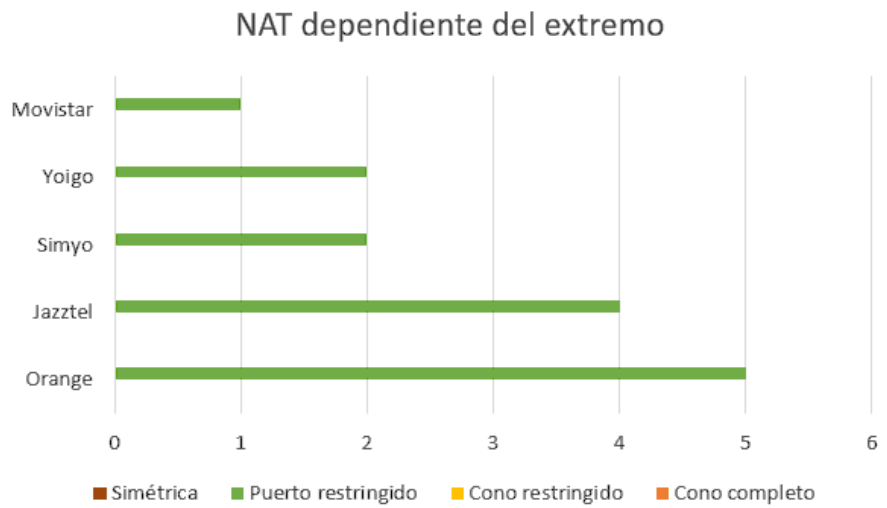

Figura 5.19: NAT dependiente del extremo según la compañía

Considerando las especificaciones inherentes al tipo de Traducción de Direcciones de Red (NAT) conocido como 'NAT de puerto restringido', las empresas aseguran una medida de seguridad al establecer un firewall rudimentario. Surge un interrogante válido en cuanto a por qué no se opta por la modalidad de NAT simétrica, la cual, en teoría, añade un nivel superior de seguridad al asignar una combinación única de dirección IP y puerto para cada conexión.

La razón subyacente a esta elección radica en que esta capa adicional de complejidad conlleva prolongados retrasos durante la transmisión de paquetes, pudiendo deshabilitar mecanismos como el 'Hole-Punching' 2.2.2. Este hecho puede haber afectado también a la decisión de utilizar NAT de sobrecarga para la red de la UPV. Aunque implica cierto grado de complejidad añadida en la red, merece la pena invertir en ello si garantiza cierta seguridad adicional.

En resumen, la elección de NAT por parte de las compañías telefónicas o empresas refleja una adaptación estratégica para optimizar el uso de recursos y la eficiencia de la red en diferentes contextos urbanos o domésticos.

CAPÍTULO 6
Conclusiones

En este estudio, hemos sumergido nuestras miradas en el complejo mundo de la Traducción de Direcciones de Red (NAT), desglosando sus diversos tipos y revelando sus implicaciones en la conectividad cibernética moderna. Al finalizar este recorrido, es claro que hemos alcanzado los objetivos trazados al inicio y hemos logrado una comprensión más profunda de cómo los diferentes tipos de NAT moldean la manera en que los dispositivos se comunican en una red.

Estos objetivos se han logrado tras sobreponerse a obstáculos que aparecían en el camino en cada fase del proyecto. Idear un esquema de red capaz de detectar e informar sobre los tipos de NAT ha requerido un estudio a fondo a cerca de este método y el diseño de una solución a la altura de las necesidades. Además, enfrentarse a nuevas herramientas, como puede ser Android Studio, y diseñar una aplicación móvil es una tarea ardua si no se ha cursado la rama de software.

La síntesis de los hallazgos revela que los tipos de NAT ofrecen soluciones diversas para los retos de direcciones IP en la actualidad. El análisis estadístico ha arrojado luz sobre la prevalencia relativa de estos tipos en redes de diferentes escalas, destacando el dominio de NAT de sobrecarga en redes a gran escala y la superioridad de NAT estática en el entorno de redes domésticas.

Nuestro estudio no solo ha cumplido con los objetivos de investigar y clasificar los tipos de NAT, sino que también ha sentado las bases para futuras exploraciones. Los desafíos aún prevalecen en el ámbito de la seguridad de las comunicaciones a través de NAT, y las implicaciones de adoptar NAT en un mundo cada vez más interconectado son áreas abiertas a futuras investigaciones.

En última instancia, este estudio no solo contribuye a nuestra comprensión académica, sino que también ofrece pistas prácticas para administradores de redes y profesionales de la ciberseguridad. Los tipos de NAT pueden ser más que una simple terminología técnica; son herramientas que moldean la eficiencia, la privacidad y la seguridad de nuestras redes digitales. Este estudio reafirma que un conocimiento sólido de los tipos de NAT es esencial para construir y mantener redes confiables en un mundo en constante evolución.

En conclusión, la Network Address Translation (NAT) sigue siendo una pieza esencial del rompecabezas de la conectividad y la seguridad en el mundo digital actual. Desde su inicio como solución para la escasez de direcciones IP hasta su evolución para enfrentar desafíos de interoperabilidad y seguridad en la era de IPv6 y la nube, NAT ha demostrado su adaptabilidad y relevancia constantes. Aunque enfrenta desafíos en la era de la IoT y la necesidad de comunicaciones directas, NAT continúa desempeñando un papel crucial en la optimización de la distribución de direcciones y la protección de redes en un mundo cada vez más interconectado. Con el continuo avance tecnológico, se espera que

NAT siga evolucionando para enfrentar los desafíos futuros y mantener su importancia en la infraestructura de redes global.

6.1 Relación del trabajo desarrollado con los estudios cursados

En el análisis y el desarrollo de este trabajo hemos podido identificar qué conocimientos y herramientas aprendidas en la carrera han sido utilizadas. Entre ellas encontramos uno de los lenguajes de programación más utilizados como es Java hasta herramientas de análisis de redes como Wireshark que se imparten en diferentes cursos de Ingeniería Informática. Es interesante ver como gracias a ciertas buenas prácticas que adquirimos y una mecánica sistemática de trabajo somos capaces de adaptarnos a nuevos retos.

En mi caso, estudiar la rama de redes me ha aportado cierta visión sobre estructuras de red que he podido plasmar en la solución del proyecto. Gracias a los primeros años en los que se tiene una visión más amplia de la informática, he podido lidiar con el uso de una aplicación como es Android Studio y aplicar buenas prácticas de software o usabilidad de aplicaciones.

A pesar de que la traducción de direcciones de red había sido previamente tratada en dos asignaturas de la rama TI (Sistemas y Servicios en Red, y Redes Corporativas), profundizar en esta materia ha permitido contrastar que seguían habiendo cuestiones que no pudieron ser tratadas.

6.2 Trabajos futuros

Este trabajo tiene algunos aspectos nos hubiera gustado mejorar, pero el alcance de un trabajo de Fin de Grado lo impide :

- **Conseguir una población de datos mayor para el estudio estadístico.** Al tener una muestra tan reducida de usuarios es posible que se nos hayan escapado datos relevantes, no se ha podido disponer de tantas versiones como hubiera sido deseable por referirse únicamente a un entorno reducido propio de un trabajo de fin de grado.
- **Mejorar el aspecto visual de la aplicación móvil y de la aplicación para ordenador.** Se ha utilizado una interfaz bastante sencilla para el móvil con tal de ejemplificar cada prueba de forma clara. Por otro lado, la aplicación para ordenador utiliza únicamente la terminal, pero sería mejor desplegar una interfaz gráfica. Las características propias de un trabajo de fin de grado obligan a centrarse en otros aspectos tratados por ser de mayor interés.

Durante el desarrollo de este proyecto, han surgido ideas sobre posibles ampliaciones o implementaciones de nuestra solución que podrían ser interesantes:

- **Utilizar nuestra solución en una aplicación de mensajería del tipo P2P que implemente técnicas NAT.** Una vez asentadas las bases para asegurar la comunicación entre dispositivos detrás de NAT, la posibilidad de crear una aplicación de mensajería es muy tangible.
- **Diseñar una página web que con un solo clic te muestre el tipo de NAT.** En vez de diseñar una solución basada en la comunicación directa con servidores, una página

web facilitaría el acceso a más usuarios y probablemente sería más simple ya que no haría falta descargar ninguna aplicación.

- **Implementar la solución en Docker para montar los servidores de forma instantánea.** Esta opción sería una clara mejora de portabilidad y eficiencia ya que Docker automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización.

Bibliografía

[1] Fabian Schneider (2011) *NAT Usage in Residential Broadband Networks.*

[2] Bruce Hartpence, Daryl Johnson (2010) *A Re-examination of Network Address Translation Security.*

[3] RFC 791 Protocolo Internet Consultado el 15 de junio de 2023. Visitado en https://www.rfc-es.org/rfc/rfc0791-es.txt

[4] Information on RFC 6598 Consultado el 18 de junio de 2023. Visitado en https://www.rfc-editor.org/info/rfc6598

[5] Guan Xu 2021 J. Phys.: Conf. Ser. 2031 012040 *Research on the Application of the IPv6 Network Protocol.*

[6] Google collects statistics about IPv6 adoption in the Internet on an ongoing basis. Consultado el 8 de julio de 2023. Visitado en https://www.google.com/intl/en/ipv6/statistics.html#tab=per-country-ipv6-adoption

[7] Movistar, primer operador español en desplegar IPv6. Consultado el 10 de julio de 2023. Visitado en https://www.movistar.es/blog/mi-movistar/movistar-primer-operador-espanol-en-desplegar-ipv6/

[8] Amazon AWS da la puntilla a las direcciones IPv4: cobrará por cada IP antigua manteniendo IPv6 gratis. Consultado el 1 de agosto de 2023. Visitado en https://bandaancha.eu/articulos/amazon-aws-da-puntilla-final-direcciones-10612

[9] RFC 2663 Terminología y consideraciones sobre Traducción de Direcciones IP. Consultado el 10 de junio de 2023. Visitado en https://www.rfc-es.org/rfc/rfc2663-es.txt

[10] Dr. Rajamohan Parthasarathy, Ms. Preethy Ayyappan (2020) *Design And Configuration Of Static Network Address Translation Techniques Method Using Cisco Packet Tracer Tool.*

[11] William Wilson (2015) *Network Address Translation and Dynamic Host Configuration Protocol.*

[12] STUN a través de NAT Consultado el 17 de junio de 2023. Visitado en https://www.rfc-editor.org/rfc/rfc3489.txt

[13] Ford, Bryan; Srisuresh, Pyda; Kegel, Dan (2005) Peer-to-Peer Communication Across Network Address Translators.

[14] Jun Bi, Miao Zhang y Lei Zhao *Security Enhancement by Detecting Network Address Translation Based on Instant Messaging.* Lecture Notes in Computer Science, vol 4097. Springer, Berlín, Heidelberg, 2006.

[15] Error de malformación de strings en JSON. Consultado el 17 de mayo de 2023. Visitado en https://stackoverflow.com/questions/59484200/com-google-gson-stream-malformedjsonexception-use-jsonreader-setlenienttrue-t

[16] Definición de actividad principal en Android Studio. Consultado el 125 de mayo de 2023. Visitado en https://developer.android.com/guide/components/activities/intro-activities?hl=es-419

[17] Permisos especiales para Acceder a Internet en Android. Consultado el 10 de mayo de 2023. Visitado en https://stackoverflow.com/questions/56266801/java-net-socketexception-socket-failed-eperm-operation-not-permitted

[18] Cláusula para ejecutar operaciones en MainActivity. Consultado el 15 de mayo de 2023. Visitado en https://stackoverflow.com/questions/25093546/android-os-networkonmainthreadexception-at-android-os-strictmodeandroidblockgua

APÉNDICE A
Objetivos de desarrollo sostenible

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**				**X**
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**			**X**	
ODS 8. **Trabajo decente y crecimiento económico.**			**X**	
ODS 9. **Industria, innovación e infraestructuras.**	**X**			
ODS 10. **Reducción de las desigualdades.**			**X**	
ODS 11. **Ciudades y comunidades sostenibles.**		**X**		
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				
ODS 17. **Alianzas para lograr objetivos.**		**X**		

Tabla A.1: Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

A.1 Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Los avances tecnológicos en redes de comunicación y conectividad están desempeñando un papel fundamental en la promoción de los Objetivos de Desarrollo Sostenible (ODS) al facilitar la colaboración global, el acceso a la información y la superación de barreras geográficas. Estas tecnologías están abriendo nuevas oportunidades en áreas como educación, crecimiento económico, innovación y desarrollo sostenible en general.

Network Address Translation (NAT) es una técnica utilizada en redes de computadoras para traducir direcciones IP entre sistemas de redes privadas y públicas. No está directamente relacionada con los Objetivos de Desarrollo Sostenible (ODS) en sí. Aun así, se puede relacionar indirectamente con redes de comunicación y la conectividad entre ellas. Influye en la eficiencia de la infraestructura de red, la innovación tecnológica, la planificación urbana sostenible y la colaboración global. A continuación, se resumen las posibles influencias de NAT en los ODS 9, 11 y 17:

En el contexto del Objetivo 9: Industria, innovación e infraestructuras, NAT podría afectar la eficiencia y la innovación en la infraestructura de red. La implementación adecuada de NAT puede permitir que las organizaciones optimicen el uso de direcciones IP y gestionen eficientemente el tráfico de red. Una gestión eficaz puede llevar a una infraestructura más escalable y confiable, reduciendo la necesidad de expansión constante y optimizando el consumo de energía. Además, NAT podría influir en la implementación de tecnologías emergentes como el Internet de las cosas (IoT) y la computación en la nube, lo que fomenta la innovación en diferentes sectores industriales. Sin embargo, también es importante abordar posibles desafíos de seguridad y compatibilidad que podrían surgir en un entorno de NAT complejo.

En el ámbito de Ciudades y comunidades sostenibles (Objetivo 11), NAT podría influir en la gestión eficiente de la conectividad en áreas urbanas. El uso adecuado de NAT puede permitir la implementación de soluciones de IoT en ciudades, como el monitoreo de tráfico y la gestión de recursos energéticos. Esto contribuye a una planificación urbana más sostenible al optimizar el uso de recursos y mejorar la calidad de vida de los ciudadanos. Sin embargo, es fundamental considerar la escalabilidad y la adaptabilidad de las soluciones de red para garantizar una infraestructura robusta en el crecimiento urbano.

En lo que respecta al Objetivo 17: Alianzas para lograr objetivos, NAT puede tener un impacto limitado en la colaboración global y la cooperación entre países y organizaciones. Si bien NAT puede afectar la comunicación entre redes y la conectividad internacional, su impacto directo en las alianzas para objetivos no es significativo. Sin embargo, es esencial considerar que la colaboración internacional y la cooperación en la consecución de los ODS dependen en gran medida de una infraestructura de comunicación sólida y segura, en la cual NAT juega un papel indirecto.

En resumen, la Network Address Translation (NAT) tiene implicaciones indirectas en varios Objetivos de Desarrollo Sostenible. En el ODS 9, puede afectar la eficiencia y la innovación en la infraestructura de red, facilitando la implementación de tecnologías emergentes. En el ODS 11, podría contribuir a una planificación urbana sostenible mediante la optimización de recursos y soluciones de IoT. Aunque NAT no tiene un impacto directo en el ODS 17, su correcta implementación es esencial para una comunicación efectiva y colaboración en la consecución de los ODS. En última instancia, es crucial abordar los desafíos y consideraciones técnicas asociadas con NAT para maximizar sus beneficios en el contexto del desarrollo sostenible.